A Systematic Approach towards Detecting, Identifying, and Recovering from Component Failures in Real-Time, Fault-Tolerant Distributed Systems

An SBIR submission to Navy N03-087
Rapid Deterministic Fault Detection in Distributed Systems

by The Open Group

A. Identification and Significance of Problem or Opportunity

A.1. Summary

The capability for recovery from failures is incorporated in almost every computer-based system that exists today. For example, enterprise computer systems utilize ECC memory; microprocessors detect and "vary out" bad data cache lines; the ubiquitous TCP networking protocol detects and retransmits "dropped" packets. As a result, the availability and reliability characteristics of commercial components have improved dramatically, perhaps by an order of magnitude each decade. Yet, the design of fault tolerance mechanisms for highly available, mission critical weapons systems continues to be difficult, requiring specialized skills of highly experienced engineers. In addition, the resulting systems are often "stove-piped," requiring a significant effort to coordinate responses to faults in systems-of-systems environments.

The basic problem is that there is no coordination between the various fault tolerance techniques. In most cases, this is benign and the system behaves reasonably. In some situations, however, the fault tolerance strategies interact with — or even interfere with — each other, possibly leading to a cascade of failures, resulting in overall system failure. Component failures are generally probabilistic, and these cases are rare enough that the problem need not be addressed in typical commercial systems.

These issues must be addressed, however, when modern weapons systems are created from the commercial components. Such systems include two additional fault sources that violate the stochastic model. The first is battle damage: components in close proximity have a common failure mode. The second is real-time deadlines: missing a deadline is a failure that must be handled, but the process of recovering from one such timing failure is likely to induce another timing failure later.

There is a current opportunity to address this problem arising from the confluence of several conditions:

- Commercial enterprise systems are becoming complex enough that they require system-wide resource management, and commercial vendors are adapting their system monitors to provide such management. In addition, vendors are cooperating to create relevant industry-standard models, such as the DMTF's Common Information Model (CIM), and The Open Group's Application Instrumentation and Control (AIC) and Application Response Measurement (ARM) standards.

- Weapons systems are being redesigned so that they can be upgraded incrementally in order to incorporate improvements in COTS components, both hardware and software. This often takes the form of an architecture based on open systems that defines a common system model for several weapons systems. The Navy, for example, has recently introduced the Navy Open Architecture (NOA). These common architectures are a forcing function that encourages the use of industry standards, such as CIM.
- Recent systems research, such as DARPA's Quorum program, has led to insights into systems interaction and the identification of QoS parameters that can be used to characterize the requirements and capabilities of components relative to dependability.
- The Open Group has recently developed a Fast Failure Detector (FFD) component based on the results of the Quorum program. While this component is specialized toward detection of network node failures, the process of extracting the failure detector into a separate component has led to a number of insights about interactions between components with interacting dependability requirements.

We propose to leverage these insights and develop a dependability model for component interactions. We will investigate the use of this model in the context of the Navy Open Architecture with the intent of allowing the use of industry-standard protocols and commercial components within the NOA resource management system. We will investigate the feasibility of managing real-time, mission-critical applications using commercial enterprise resource management systems.

Based on our experience with the FFD component, we believe that the crux of the issue is identifying the characteristics of a failure—the particular guarantee of service that has not been satisfied. The ability to specify this policy statement—with defined, documented, and measurable parameters—will enable an analytic approach to the problem. In the case of FFD, these policies are incorporated as part of its operational parameters, which are established via a QoS Application Programming Interface (API). Thus, we intend to focus specifically on this API as a specific instantiation of a more abstract dependability model.

We propose to use the formulation and definition of this API as a driving function in examining the role of failure detection in mission-critical applications. We will investigate potential implementations of such failure detectors that range from intrusive (built into the algorithms of the application) to passive external observers (no interaction with applications).

Successful completion of this SBIR effort will produce several products for use in real-time mission-critical systems:

- A taxonomy (enumeration) of common failure detection strategies in real-time systems.
- A failure model with QoS characterictics specific to real-time mission-critical systems.
- An API that incorporates those QoS parameters and supports implementations of the common failure detection strategies.
- A set of reusable failure detector components based on that API.
- A set of worked examples that demonstrate the use of these failure detector components in implementing highly dependable, real-time systems. These examples would include scenarios with both commercially available enterprise resource management systems and specialized resource management systems.

A.2. Problem Description

Modern weapons systems are complex and distributed, involving multiple subsystems, each of which typically comprises multiple components. System components receive data from multiple sources of information, fuse that information, select a response, and then put that response into action. The sensors and actuators are often physically remote, connected via interruptible communication paths. And, of course, weapons systems are particularly subject to physical damage.

Modern weapons systems are also mission critical. The overall system must react to environmental events in a timely and reliable manner. Defensive systems, in particular, must respond to unpredictable events, and therefore must provide highly reliable, highly available service even in severe conditions.

A.2.1. Fault Tolerance

Design of fault-tolerant weapons systems is not straight-forward, and a major role of the system engineer is to select the strategy for providing fault tolerance. There are many fault tolerance techniques, and any particular system will usually involve multiple mechanisms, with different choices for different subsystems. In addition, the strategy is usually hierarchical, whereby individual components are themselves fault-tolerant, but there are also mechanisms by which higher level components will recover if those fault-tolerant components should fail.

The rationale for a hierarchy of strategies is that no single system or subsystem can ever provide 100% availability. There are many reasons: First, it is not possible to predict all possible failure modes in a system. Thus, it is not possible to systematically design a system with strategies for recovering from all failures. Second, adding mechanisms for recovery adds complexity and increases the realm of potential failure modes. Perhaps the most important restriction, however, is due to the finite resources available to any project: limited funding, tight timeliness, and a constricted pool of knowledgeable design engineers.

This limitation on resources has led to the increasing utilization of COTS components, and even commodity components. This evolution is occurring simultaneously at many levels in the system. Commercial DSP ICs are incorporated into special peripheral boards, which are plugged into commercial computer workstations and servers that form the core computing infrastructure. Those computers then interact utilizing commercial routers communicating via commercial network products. Software applications are built atop commercial operating systems and commercial middleware, such as CORBA products. They incorporate commercial database systems, such as Oracle. The executable programs are created using commercial compilers that link in commercial algorithm libraries.

Almost by definition, the characteristics of these components are beyond the control of the system engineer. The evolution of a commercial component will be determined by the commercial supplier based upon that supplier's view of the "needs" of the marketplace, which is usually interpreted as the path that provides the most profit to that supplier. As a result, the components are likely to change over time. Some changes will be benign; others will even be beneficial. Some, however, will be detrimental to the weapons system. Often these changes will be subtle, such as timing characteristics, or susceptibility to faults. In many cases, the manufacturer wasn't aware that customers depended on these characteristics, assuming that they were uncomplicated internal engineering design choices.

The system engineer can select from suitable components available at design time. The system engineer can also delay including the modified components into fielded systems, but only to the extent that earlier versions of those components have been warehoused and are available. In addition, this imposition of such a delay must be an explicit action. The need for a delay must be recognized based on detecting changes in components received from the supplier. (Remember that the manufacturer didn't realize that the change was important.) That requires significant and ongoing testing of the system, and that requires resources — which bumps up against the project development resource limits.

Consequently, weapons systems designers must balance many factors in order to create the most effective system possible. The overall system must be usable in the context that it will be manufactured, deployed and maintained. The system must adapt to the changes that will be imposed upon it. Many of these changes are unpredictable and will affect future design changes. To accommodate future changes, a design must be understandable; to be understandable, a design must be simple — or be perceived as not complex. This difference between reality and perception has helped to foster the conversion to use of objects in systems. To the extent that an object, or other component, can be treated as a black box, a designer can ignore the subtleties of that component. This results in fewer variables in the design space, and usually leads to a simpler, more understandable design.

Many components, of course, should not be treated as black boxes. There are the variations in COTS components described earlier. Purpose-built components will experience similar evolution. In addition, there may be knowledge available that improves the ability to select an effective strategy for recovery from failures. For example, an application running on computers that are subject to battle damage might provide overall higher availability when the recovery strategy involves switching to a physically distant backup computer.

A.2.2. Failure Detection

In order to recover from a failure, a system must be cognizant that a failure has occurred. Real-time systems have time constraints. In many cases the timeliness of an operation is more important than the knowledge incorporated into that operation. For example, the signal processing subsystems of distributed radar systems are often organized into processor pools. Processing of individual radar reports will be apportioned among those processors. If one of those processors should fail, the fate of a particular computation operation may be irrelevant. What is important is that the overall system continue operation — that it continue to receive radar sensor reports, that it continue to process most of those reports, and that it continue to produce radar track information in a timely manner.

In order to continue real-time operation in the presence of failures, however, the system must incorporate a strategy for dealing with failures. In this example, for instance, the system must not suspend operation while waiting for the results of all computations that have been assigned to other components. Instead, it might disregard the missing computation result and continue its processing using the information that is available. There is a trade-off here, however. Utilization of more data sources typically produces better information. A multi-sensor radar system should fuse as many of those sources as are available. Thus, timely detection of a failure is important: The earlier that a system knows that a component has failed, the sooner it can abort the wait for data from that component, and the faster it can produce and deliver its own result.

Timely detection of failures may be particularly crucial to certain types of operations. While radar systems may be relatively insensitive to the loss of individual sensor reports, other systems may not be. For example, an offensive weapons system may distribute its computations across a processor pool. In this case, however, each of those results may be necessary for successful launch. Thus, a missing computation due to a failed component must be reassigned to another component. But, the recovery action can not be initiated until the failure has been detected. The longer the period before the failure has been detected, the less time is left for the execution of the recovery action prior to the deadline.

Of course, an alternate strategy would have been to perform each computation in parallel. This is not usually feasible to due the enormousness of the required computation. Even if it were, however, there is still the possibility of both computations failing. This is why selection of failure recovery strategies is so important to the systems engineer. Fault tolerance is a systemic property (an emergent property in complexity theory). Fault tolerance is a distinct engineering discipline. There is a separate body of literature and a separate base of engineering experience. This knowledge base must be effectively integrated into the overall system design.

The dissection of separate, reusable components for failure detection and recovery promotes the effectiveness of that integration. The separation allows the incorporation of specialized knowledge about the application environment and the nature of failures that are probable. Separation also allows the use of detection strategies that were not contemplated as part of the original component. Reuse improves the quality of the component by increasing the resources available for developing, documenting, and testing the component. In addition, reuse leads to the development of an experience base — the existence of knowledge about how to use a component, how it behaves, how it misbehaves, and how its operation can be misunderstood.

A.2.3. False Positives

The requirement for timeliness in real-time systems has a side-effect: sometimes a component has not failed, but is simply slow. This tardiness is often an expected occurence in normal operation. Environmental noise on a communication link can lead to corrupted information and the need for data retransmission. Operating system schedulers include some jitter — even in real-time operating systems. Application algorithms may be unpredictable. Kalman filters, for instance, involve a third power dependency.

The root cause of the problem in generally irrelevant (for purposes of maintaining timeliness.). Sometimes, the failure detector in a real-time system must declare a component to be failed even when it is only tardy. This is a *false positive* condition and must be handled appropriately. If an application has determined that a component has failed and then a result from that component shows up, the application is likely to produce an incorrect result. In addition, there is often a high cost for reincorporating failed components — and even tardy components — back into the normal processing stream.

This leads to a trade-off: the shorter the latency that is tolerated by a failure detector, the more quickly a system can recover from a failure. Conversely, the longer the latency, the fewer false positives that will occur. The choice of the latency period is a Quality of Service (QoS) issue, and the system engineer should be able to treat the selection as an explicit design parameter. In fact, this selection of timeout characteristics is probably the most common criterion used for failure detectors, and incorporating it as a tuning parameter is key to the creation of reusable componentry.

A.2.4. Fault Isolation and Failure Recovery

It is not sufficient just to detect a failure. The system must identify the characteristics of the failure and make a determination about the fault (or faults) that is causing the failure. It is not necessary, however, for a system to fully determine the root cause of a failure. It does not matter, for example, whether the central processor or the system bus of a computer has failed. Applications will not be able to run on that computer, so a recovery strategy must involve the use of a different computer. Similarly, it does not matter whether a remote computer has failed or whether only the communications links to that computer have failed. No data is flowing to or from that computer, so a recovery strategy must involve a different computer using a different set of communications links.

It is important, however, for the system to determine a consistent strategy for failure recovery. Different components in a system may be impacted differently by a particular failure. Accordingly, each of the components might initiate a different recovery procedure, a situation that often leads to chaos and overall system failure. Thus, it is necessary that failure recovery be coordinated. In highly integrated systems, such as the Navy Open

Architecture, this coordination must cover both the real-time and non-real-time portions of the system. It is very beneficial, therefore, for the real-time portions of the system to be integrated with the enterprise resource management system(s) for the overall system.

Most of this coordination must occur prior to the failure. The role (or roles) that each component plays must have been identified as part of system configuration and appropriate information provided to the components. In addition, up-to-date state information must be maintained throughout the system as part of its normal execution. Then, when a failure happens, each component is already aware of the state of other components. This reduces the amount of data that must be exchanged during failure recover to only the information about the failure and its effects on the various components.

A.3. An Analysis of the Problem in the Context of Navy Applications

Although many aspects of military weapons systems are similar to commercial systems, there are numerous differences that can affect the selection of solution patterns for Navy weapons systems.

One distinguishing feature is the real-time nature of the systems. Generally, commercial systems can be real-time or they can be complex, distributed systems. There are few commercial systems that are both. Military systems, on the other hand, are often both real-time and complex, distributed systems. The hallmark of future military systems will be the capability of correctly operating in more complex contexts than the enemy can.

The purpose of offensive military weapons is to disable the enemy's fighting capacity, which is usually performed by destroying the enemy's weapons and often by killing enemy forces. Because the enemy has similar but opposing goals, defensive military weapons systems are usually life-critical. In addition, the stimuli applied to defensive systems are often under the control of the enemy, which can be expected to select times and methods of attack for its own advantage. Thus, it is important that shipboard resources be applied to the most critical tasks. Dynamic changes in resource availability should result in a reevaluation and potentially a reassignment of resources.

Explicit management of resources is so critical that the Navy Open Architecture includes a separate subsystem for this purpose. This subsystem is pervasive and comprehensive: it interacts with and controls components at every level of the system. It must evaluate the overall mission of the ship and assign resources accordingly. The reuse and interoperability aspects are obviously important, and this is one area where the benefits of a common architecture are particularly apparent.

Consider the issue of fault trend analysis, as it might be performed by HP’s OpenView resource management system. Failure detectors for individual components could report “interesting” events, e.g., network packets with bad checksums, or message latencies over 75% of the timeout values. Background trend analyzers could then predict failures. For example, if a particular application is encountering such problems, that application may be misconfigured. On the other hand, if all applications using a particular network are encountering problems, then that network might be overloaded. This analysis can then direct additional analysis. Notice, however, that the “interesting” events are mostly easily identified via specific criteria. The definition and use of mensurable metrics within a system simplify the task of resource management allowing the use of simple analysis techniques rather than heuristics, which are inherently system-specific. The use of commensurable, industry standard metrics enables the reuse of common, perhaps even commercial resource managers. Thus, the identification and specification of common QoS parameters for failure detection is important to the overall problem of system resource management.

A.4. Opportunity

The capabilities of The Open Group offer an excellent opportunity for advancing the effectiveness of failure detection — in both Navy applications and in commercial applications. The Open Group provides experience in QoS-aware components for real-time, fault-tolerant systems, and a history of transferring technology to commercial vendors. In addition, it has existing relationships with defense contractors, such as Lockheed Martin and Raytheon, who can provide a knowledge of Navy applications needs and an understanding of the underlying nature of those applications, resulting in an ability to distinguish between those characteristics that are fundamental to a solution, and those characteristics that are artifacts of the history of an applications domain. These collaborations offer the potential for short-term incorporation of effective solutions into Navy applications and the possibility of possible transfer to commercial vendors and COTS status over the longer term.

A.5. Proposed Solution

A.5.1. Overview

As a result of research sponsored by DARPA, the US Air Force, the US Navy, and several commercial organizations, The Open Group has developed a vision for addressing the special requirements of distributed, real-time, fault-tolerant systems. We are currently developing a framework and tool kit that implements portions of that vision under a Phase II SBIR effort (see Section K). Previously developed components include CORDS/GIPC, a real-time group communication system (see Related Work). During research sponsored under DARPA's Quorum program, the concepts used in the CORDS/GIPC failure detector were extracted and reimplemented in a separate component, the Fast Failure Detector (FFD), which can be used independently of CORDS/GIPC.

The Open Group believes that an opportunity exists to develop a better model of failure identification and detection, and to provide componentized support for real-time, fault-tolerant distributed systems. We believe that this model and these components could be incorporated into shipwide resource management systems based on commercial products, such as HP’s OpenView or IBM’s Tivoli, and propose to investigate this opportunity using Navy Open Architecture weapons systems as prototypical applications.

The effort would focus on the failure detection aspect of the overall problem because of the potential for extracting it to a separate, reusable component that could be inserted into existing applications with minimal impact on the current designs. We would also intend to explore the feasibility of using multicast protocols as a fundamental (but optional) communication concept within the failure management system in future Navy applications. Examples of such multicast protocols include group communications, such as the CORDS/GIPC component, and publish/subscribe protocols.

A.5.2. Fast Failure Detector

The existing Fast Failure Detector (FFD) addresses a problem identified by the HiPer-D group (see Related Work section). The HiPer-D test-bed uses the Ensemble group communication system to provide scalable fault tolerance. One of the tightest real-time constraints within the HiPer-D system is the Aegis AAW (ship self-defense) execution path, which includes multiple applications and has an aggregate required response time on the order of multiple seconds. The HiPer-D prototype system easily achieves this goal during normal operation.

The Hiper-D team determined that the system did not, however, meet this deadline while recovering from component failures. (Note that this is a self-imposed goal within the HiPer-D project. The actual Aegis system does not have this requirement and takes at least an order of magnitude longer to recover from failures.) The basic problem is that the performance scalability of the system is based on the use of virtual synchrony, which requires that multiple nodes coordinate the delivery of messages. The Ensemble group communication system delivers each data message to the nodes of all recipients. The nodes must then exchange additional control messages to determine that all intended recipients have received all required messages. At that point, each node can release the data messages to the application processes. If one node does not acknowledge receipt, the other nodes must delay (on that particular data stream) until the node does respond or until the node is ejected from group membership.

Real-time systems must meet time constraints. Real-time systems utilizing group communication must impose time limits on receipt of message acknowledgements in order to meet those time constraints. Nodes that do not respond must be ejected from group membership in order for the overall application to meet its timeliness goals. A problem arises in that sometimes unresponsive nodes have not failed, but instead are simply slow, i.e., situations where tardy nodes induce false positive conditions. While handled correctly by applications based on group communications, the result is that each tardy application must be reinitialized and then allowed to rejoin the execution group. This is an expensive, high overhead proposition, requiring full reacquisition of application state, and is highly undesired by systems engineers.

A basic design issue with using network communication in fault-tolerant systems is that there is no positive indication that a failure has occurred. Instead, failure is inferred by the absence of anticipated activity. Ensemble's group communication host failure detection mechanism is based on time-outs of heartbeat messages. Individual message delivery time-outs typically must operate an order of magnitude faster than the overall system time constraint. Thus, the AAW execution path would require sub-second time-outs.

Unfortunately, these time-out periods are of the same magnitude as scheduling jitter in non-real-time operating systems and middleware components.

Ensemble's design target was to provide high throughput, not real-time predictability. As such, multiple threads of control are used internally to batch messages in order to optimize network traffic. There is no way to tag particular messages and/or threads as high priority. Thus, the only way to provide higher priority to the heartbeat messages would be to provide higher priority to the entire Ensemble AAW process. Due to its life-critical function, the AAW execution path was already operating at one of the highest priority levels in the system. Thus, there was no way to provide a design-time guarantee that the HiPer-D AAW application could meet its latency performance objectives while recovering from failures.

The existing FFD was conceived as a way to address this deficiency. Recognizing that the crux of the problem was delivery of low latency messages, the FFD component replicates Ensemble's heartbeat function. The FFD component, however, has been developed using real-time design and programming techniques. Thus, system resources can be dedicated to the delivery and processing of time-critical heartbeat messages. If FFD determines that a remote node has failed — or is simply tardy, it notifies the application on the local node, which then initiates ejection of the other node. (The original design called for FFD to inject the failure notification directly into Ensemble, but that proved to require too many changes to Ensemble itself.)

As a result of the development and inclusion of FFD, HiPer-D engineers have reported that they are operating test-bed systems achieving node failure detection times on the order of 100 to 150 milliseconds. This performance is being achieved using Sun Microsystems computers running Solaris, and PC-class systems running Carnegie Mellon University's version of Linux with Resource Kernel enhancements on a Gigabit Ethernet network.

A.6. Technical Approach

Although our technical approach is based on the currently existing Fast Failure Detector, discussions with the HiPer-D team led to insights about potential modifications, extensions, and alternate algorithms. These concepts provide the basis for creating a general API that would enable an object-oriented approach to mission-critical applications across a broad range of problem areas, including Navy weapons systems.

A.6.1. Current Fast Failure Detector (FFD)

The current FFD has several advantages. First, it has a simple, easily understandable interface. Essentially, the API is a list of network hosts and a maximum latency time. The FFD generates heartbeat messages to announce its liveness to other FFD components; simultaneously, it checks for heartbeat messages received from other instances of FFD on other hosts. If no messages are received from a host for a period longer than the maximum allowed, the application is notified by a callback or a local message (depending on the selected communication structure).

Second, the FFD is a separate component that performs only failure detection. Thus, it can be inserted into other applications without perturbing the design strategies of those applications. This also simplifies testing. Many "fault-tolerant" applications aren't: Although the designs were intended to provide fault tolerance, there are often design or implementation errors. Because of limited testing time and facilities, many systems cannot perform exhaustive testing of faulty component configurations. We have been able to deploy and test FFD simply and effectively using fault-insertion techniques as well as by explicitly taking nodes off-line. In addition, the use of the FFD API simplifies testing of the application by reducing the number of external fault signals, as well as providing a convenient point for initiating fault insertion into the application.

Another advantage is that the FFD has been designed using real-time design techniques. Each function within FFD is performed within a single dedicated thread, allowing precise allocation of resources to each function. The locks are organized into a simple hierarchy, allowing the effective use of operating system locks that support priority ceiling or priority inheritance algorithms. Finally, FFD has low overhead: its CPU usage is below the threshold of measurement (on Solaris and Linux) in the existing test-bed configurations.

The current FFD is also capable of enhancing the real-time performance of a pre-existing component that was not designed for use in real-time systems. By amplifying the capabilities of such components, FFD increases the number and variety of components suitable for use in real-time, fault-tolerant applications.

A.6.2. Alternate Fast Failure Detector Implementations

Our experience with the current FFD both in The Open Group's facilities and in the HiPer-D test-bed at NSWC Dahlgren has identified several enhancements and alternate implementations that we believe would prove useful in other applications, such as Navy weapons systems. The enhancements and alternate implementations generally would not alter the API for applications using the existing FFD capabilities.

The design of the current FFD was undertaken to address a specific deficiency in failure detection identified by the NSWC HiPer-D team — the lack of rapid, reliable detection of failures of remote components on networked hosts. The HiPer-D applications already had rapid failure detectors for other failure modes, most notably the detection of process failures via the use of the UNIX *waitpid*(3) and the Windows NT Event wait system mechanisms. Internally, however, the HiPer-D applications make little distinction between the two failure modes. Generally, when a process/component fails, the failed component is immediately excluded from current processing and a pre-assigned backup process is activated and brought into the processing configuration when convenient for the application. The design of the applications would be simplified if both types of failures were handled in a similar fashion and if failure notifications for both local and remote components were received via the same mechanism. Thus, one can see the utility of developing a failure detector with a similar interface to the current one but that instead responds to process termination notifications.

In fact, there are additional alternate implementations that appear to be useful. One would detect liveness of threads within a multi-threaded application. For a variety of reasons, individual threads will often get "hung." Many real-time applications are event driven and implemented as an event loop. Once each time through the event loop, each thread could invoke a software component that updated a mailbox in shared memory. A single such "dead-man timer" failure detector could effectively monitor the operation of many such threads. Such an implementation would have the benefits of a common interface along with very inexpensive operation in the real-time threads.

Other implementations might take advantage of knowledge of characteristics of the underlying infrastructure. For example, the current FFD utilizes the IP multicast protocol to reduce the amount of network message traffic. While IP multicast effectively leverages the bus nature of coaxial-based Ethernet, many higher speed networks such as optical switches do not natively provide it. Thus, an FFD that could aggregate ("piggy-back") messages might cause less interference with normal application data traffic. Also, some switch technologies provide switch-specific information that could be used.

There is a significant body of academic literature of failure detection algorithms and particularly on consensus using failure detection algorithms — although there is little emphasis on real-time systems. We would investigate the literature for algorithms suitable for Navy systems.

Other implementations of a failure detector might differ primarily in the interface to the underlying communication structure. For example, the existing FFD operates at the socket level, using IP multicast. A failure detector in a CORBA-based environment might use the same failure detection and prediction algorithms, but it might communicate using CORBA invocations, or a CORBA event channel, or even the recently introduced CORBA Multicast capability. Such a component might also be expected to interface with the CORBA ORB — or provide input into a CORBA Multicast component. CORBA timeouts might provide additional (perhaps secondary) means of failure detection. Other implementations might not be network based at all, but would use alternate mechanisms such as UNIX process mechanisms (e.g., *waitpid*(2) as mentioned earlier), or explict messages from components that explicitly terminate after internally detecting inconsistencies.

A.6.3. Enhanced Fast Failure Detection Capabilities

Once we have identified alternate implementations with identical APIs, it becomes useful to consider mechanisms for aggregating indications of failure. Instead of having to communicate with multiple failure detectors, an application could register its interest in multiple failure conditions and interact with only one failure detector. It is not yet clear whether such a capability would best be provided via a special failure detection aggregation module, or via another alternate implementation that just happened to interact with multiple other failure detectors, or via some other mechanism. We would investigate this capability as part of the proposed effort.

Separate failure detector components can also assist with failure prediction. The current FFD, for example, monitors the latency of heartbeat messages. When latency is low, failure due to network or processor overload is unlikely. Increased latency, however, is indicative of increased loading on the network and/or the processor, and latencies that are approaching the maximum allowed are a predictor of failure. While the FFD generates information about the latency of messages, it is not yet apparent how best to make this information available to the applications. Indeed, this may be one of the areas where the capabilities of existing commercial enterprise resource managers can be utilized. We propose to investigate this issue as part of the proposed effort.

A.7. Benefits of Approach

The benefits of our approach accrue primarily due to the componentization of the functions supporting fault tolerance — the failure detector over the short term; failure identification and recovery, and use of commercial and specialized fault management and enterprise resource managers over the longer term. The particular advantages over the continued enmeshment in applications include:

- Reduction in development costs and time due to reuse of components
- Improved mission effectiveness due to better tested, more dependable components
- Reduced costs for testing due to more effective unit testing and less interaction between defects in fault tolerance components and overall system operation
- Less dependence on design decisions made many years before deployment. A system engineer can alter parameters for failure recovery strategies after system integration — when their interactions can be experimentally measured
- More effective field upgrades. Modifications to adapt weapons systems to new battlefield situations can be performed with significantly less development cost — and particularly less time — because many of the modifications can be performed by adjusting component parameters. Even those that require software development will be simplified because of the separation of components
- Reduced upgrade costs due to less dependency on the particular character of particular hardware and/or software components
- Improved potential for ability to use COTS fault tolerance components

B. Phase I Technical Objectives

The technical objectives of the Phase I effort will be to:

1) Produce failure models that can be used to characterize the requirements of real-time, mission-critical applications. The failure model will include multiple strategies and QoS metrics.
2) Identify a set of candidate commercial enterprise resource managers that could be used with componentized failure detectors in real-time mission-critical systems.

3) Produce a set of scenarios wherein the commercial resource managers can utilizing information from failure detectors to assist the process managing the shipboard reosurces to meet Navy mission requirements.
4) Specification of an object-oriented API for controlling alternative implementations over a broad performance range. The API would also provide an interface to the enterprise resource managers.
5) A set of prototype components that conform to the API, including failure detectors based on FFD.
6) A simple demonstration application that exercises the prototype components. (The role of commercial resource management components may be simulated.).

C. Phase I Work Plan

For Phase I, we propose the following activities to create a technical design to support failure detection in targeted real-time, fault-tolerant applications:

Application Analysis

We will identify and investigate several Navy-related candidate applications. We will draw upon Navy and Navy contractor resources as well as other defense contractors in The Open Group's membership base. We will also investigate civilian applications. We will primarily draw upon The Open Group's membership base, including participants in The Open Group's Quality of Service Task Force[1] and the Real-Time and Embedded Systems Forum.[2] Candidates SIAC[3], The European Union's Netframe project, as well as other banks and other financial institutions.

Based on our existing experience, we will start with the Aegis AAW Auto-Special Doctrine. For each application, we will identify the failure dependencies and identify the requirements for each. For example, we know that the timeout latencies for the radar processing path in the AAW application are much lower than those on the resource management path. This AAW application is also interesting because there are at least two different implementations (one in the existing Aegis fleet, and one in the HiPer-D test-bed). It should be noted that the performance characteristics of this application are classified. We do not anticipate a need to understand the actual performance requirements, only the impact of varying the QoS parameters.

[1] http://www.opengroup.org/qos/

[2] http://www.opengroup.org/rtforum/

[3] The Securities Industry Automation Corporation (SIAC) operates the computing infrastructure for the New York Stock Exchange (NYSE).

Failure Detection Models and API

We will use the results of this analysis to create simple models of the failure detection aspects of the applications, and we will identify QoS metrics that are common to the failure models of these applications. We will define an API that incorporates those QoS metrics and provides control over the failure detection strategies that were identified earlier.

Analysis of Commercial Enterprise Management Systems

We will investigate the capabilities of commercially available enterprise management systems to determine their applicability to real-time mission-critical systems with particular emphasis on the Navy applications identified earlier. We will develop scenarios that demonstrate the use of the products in Navy systems such as the Navy Open Architecture.

Prototype Components and Demonstration Application

We will develop prototype components that implement the API identified earlier as well as a demonstration application based on one of the scenarios that demonstrate the use of the commercial enterprise resource management system. Because the use of a commercial product would entail significant resources, both product cost and training time, we expect to simulate these capabilities.

Evaluation

We will analyze the results of this effort and document the results of the investigation and the proposed design. We will discuss these results with Navy personnel and defense contractors as well as other interested parties.

Option

If the option is approved, we would extend the investigation to explore the instantiation of the proposed API and components within the context of existing commercial systems. Candidates include the OMG’s Real-Time CORBA suite, which is currently developing a standardized interface for reliable multicast communication; the DMTF’s Common Information Model (CIM), which is specifying parameters that are common across industry segments; and The Open Group’s Application Instrumentation and Control (AIC), which is identifying related sets of application measurements. Based on this investigation, we would formulate a plan for commercializing these failure identification concepts and components.

— end —