COMPUTING SCIENCE

Building Fault-Tolerant Distributed Computing Systems using Standard Components

P.A. Barrett, S.K. Shrivastava, N.A. Speirs and A. Waterworth

TECHNICAL REPORT SERIES

No. 449 November, 1993
Building Fault-Tolerant Distributed Computing Systems using Standard Components

P.A. Barrett, S.K. Shrivastava, N.A. Speirs and A. Waterworth

Abstract

The aerospace industry is making increasing use of computers in the implementation of safety-critical systems. New generations of airliners (Airbus A320/330/340, Boeing 777), for example, are using digital computers in their primary flight control systems - systems upon whose integrity and availability depends the safety of the aircraft and its passengers. Such systems are required to be fault-tolerant so that they can continue to function correctly in the presence of a finite number of component failures. Current generations of fault-tolerant computers for safety-critical applications tend to make extensive use of special-purpose hardware, and are thus expensive and inflexible. This paper investigates the possibility of constructing fault-tolerant computer systems using standard hardware components, replicated to an appropriate degree and communicating via special-purpose software protocols. The Voltan family of fail-controlled nodes is introduced and described, and ways of incorporating Voltan nodes into Integrated Modular Avionics (IMA) architectures are presented. Means of overcoming the potential drawbacks of such nodes are discussed. In particular, possible extensions to IMA gateway modules in order to provide communications and data validation services in support of Voltan nodes are described.

This paper was presented at the Computing in Aerospace 9 conference in San Diego in October 1993, and has also been submitted for journal publication.

© 1993 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne, Computing Science, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

BUILDING FAULT-TOLERANT DISTRIBUTED COMPUTING SYSTEMS USING STANDARD COMPONENTS

Building Fault-Tolerant Distributed Computing Systems using Standard Components
(By) P.A. Barrett [and others]
(University of Newcastle upon Tyne, Computing Science, Technical Report Series, no. 449)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
BARRETT, Peter Adrian

Abstract

The aerospace industry is making increasing use of computers in the implementation of safety-critical systems. New generations of airliners (Airbus A320/330/340, Boeing 777), for example, are using digital computers in their primary flight control systems - systems upon whose integrity and availability depends the safety of the aircraft and its passengers. Such systems are required to be fault-tolerant so that they can continue to function correctly in the presence of a finite number of component failures. Current generations of fault-tolerant computers for safety-critical applications tend to make extensive use of special-purpose hardware, and are thus expensive and inflexible. This paper investigates the possibility of constructing fault-tolerant computer systems using standard hardware components, replicated to an appropriate degree and communicating via special-purpose software protocols. The Voltan family of fail-controlled nodes is introduced and described, and ways of incorporating Voltan nodes into Integrated Modular Avionics (IMA) architectures are presented. Means of overcoming the potential drawbacks of such nodes are discussed. In particular, possible extensions to IMA gateway modules in order to provide communications and data validation services in support of Voltan nodes are described.

About the author

Peter A. Barrett is a Principal Research Associate at the University of Newcastle upon Tyne.

S.K. Shrivastava joined the Department of Computing Science in August 1975, where he is a Professor.

Neil A. Speirs is a Lecturer in the Department of Computing Science at the University of Newcastle upon Tyne.

Adrian Waterworth is a Research Associate in the Department of Computing Science at the University of Newcastle upon Tyne.

Continued
Suggested keywords

FAIL SILENCE
FAULT TOLERANCE
FLIGHT CONTROL SYSTEMS
INTEGRATED MODULAR AVIONICS
SAFETY CRITICAL SYSTEMS
VOLTAN

Suggested classmarks (primary classmark underlined)

Dewey (18th): 621.381958 001.64404 629.135
U.D.C. 681.322.02 519.687 629.7.067
BUILDING FAULT-TOLERANT DISTRIBUTED COMPUTING SYSTEMS USING STANDARD COMPONENTS

Peter A. Barrett, Santosh K. Shrivastava,
Neil A. Speirs, Adrian Waterworth

The BAE Dependable Computing Systems Centre and Department of Computing Science,
University of Newcastle upon Tyne, U.K.

Abstract

The aerospace industry is making increasing use of computers in the implementation of safety-critical systems. New generations of airliners (Airbus A320/330/340, Boeing 777), for example, are using digital computers in their primary flight control systems – systems upon whose integrity and availability depends the safety of the aircraft and its passengers. Such systems are required to be fault-tolerant so that they can continue to function correctly in the presence of a finite number of component failures. Current generations of fault-tolerant computers for safety-critical applications tend to make extensive use of special-purpose hardware, and are thus expensive and inflexible. This paper investigates the possibility of constructing fault-tolerant computer systems using standard hardware components, replicated to an appropriate degree and communicating via special-purpose software protocols. The Voltan family of fail-controlled nodes is introduced and described, and ways of incorporating Voltan nodes into Integrated Modular Avionics (IMA) architectures are presented. Means of overcoming the potential drawbacks of such nodes are discussed. In particular, possible extensions to IMA gateway modules in order to provide communications and data validation services in support of Voltan nodes are described.

1. Introduction

Digital computers are in widespread use throughout the aerospace industry. In the civil aviation sector, for example, Flight Management Systems (FMSs) have been present on the flight deck for many years and, more recently, the European Airbus have seen the introduction (in the A320) of computers into primary “fly-by-wire” flight control systems. Such systems will normally be considered safety-critical, and will, therefore, be required to use fault-tolerance techniques in order to enable them to continue to deliver timely and correct responses despite the occurrence of component failures.

Current generations of aircraft use highly specialised computing systems, designed specifically for the tasks which they are to undertake. Although effective, such special-purpose systems are extremely expensive, and more cost-effective ways of achieving similar levels of dependability are required. Thus, future generations of aircraft are likely to move towards integrated distributed systems architectures capable of supporting all or most of the computing functions currently implemented using special-purpose units. The development of integrated computing architectures for aerospace applications is proceeding via initiatives such as Integrated Modular Avionics (IMA). 1

Equally desirable for future applications would be the capability to implement systems making use, where feasible, of standard “off-the-shelf” hardware and software components replicated to an appropriate degree. This is a challenging task, centering around the problem of providing timeliness of service despite the overheads imposed by the use of replica management protocols.

This paper describes the authors’ on-going research work, which explores the use of replication at various levels in distributed system architectures and investigates the potential for building fault-tolerant, distributed computing systems using standard components. We begin with a review of system fault tolerance techniques, including the concept of fail-controlled nodes. A distinction is made between hard and soft fail-controlled nodes, and the Voltan soft fail-controlled node is introduced. Having described Voltan in some detail, we go on to consider how fail-controlled nodes in general, and Voltan in particular, may be integrated into IMA-type architectures.

2. System Fault Tolerance

Replicating processing on distinct processors, and employing mechanisms such as voting on results, provides a means whereby outputs from faulty processors may be prevented from appearing at the application level. Such techniques (referred to as active replication) provide a practical means of constructing systems capable of tolerating fail-uncontrolled (arbitrary) processor failures.

This approach is exemplified by the Airbus A310 slat/flap control system, in which the use of replicated processing and dual independent channels is used to ensure that the possibility of incorrect control surface actuations may be ignored, and the only additional mechanism required is one by which the control surfaces may be locked in position should both channels fail, or a mechanical failure cause asynchronous deployment of the control surfaces. In general-purpose distributed computing system architectures, such as IMA, application-level fault-tolerance mechanisms may become extremely complex if arbitrary failure modes must be tolerated at that level. More desirable would be a systematic means of incorporating processor-level redundancy in such a way as to provide fail-controlled nodes as building blocks.

A node is an element of a system which provides processing resources; typically a box containing processor, memory, I/O devices and interconnections, etc. A fail-controlled node is a node designed in such a way that its behaviour on internal component failure will follow a pre-defined pattern with very high probability. A fail-controlled node may, like a nor-
mal node, be a physical entity (e.g. a box containing replicated processors, memory, etc., and a voting mechanism for validating results and masking errors), or it may be a logical entity comprising a number of fail–uncontrolled processors, a communications mechanism, and some appropriate software protocols. Depending on the design of a node, various qualities of fail–controlled behaviour may be provided.

The use of fail–controlled nodes can greatly simplify the task of providing fault–tolerant applications, since the consequences of arbitrary failure modes which lead to erroneous outputs are eliminated at the node level, leaving the higher levels to deal only with errors of omission (i.e. the disappearance of services).

2.1 Fail–Controlled Nodes

A fail–controlled computer (or node, or host) contains sufficient internal self–checking mechanisms for us to be able to make assumptions regarding its failure behaviour. A number of different classes of fail–controlled behaviour have been specified, including:

- Fail–silence.

A fail–silent node would cease to communicate with other entities within the system if it should detect itself to be in error. In a typical fail–silent node, two processors run in lock–step, with each processor having its own memory and other peripherals. The outputs of the two processors are compared by a special–purpose comparator circuit, and are only permitted to be propagated if they agree with one another. The comparator may also be used to provide functions such as failure indication, and to shut down or re–initialise the faulty processor.

- Fail–signal.

In addition to failing silently, the node will generate an explicit "node failed" signal which may be used to initiate appropriate actions at the system or application level in order to compensate for the loss of the failed node.

- Failure–masking.

A failure–masking (or N–Modular Redundant – NMR) node is able to continue to function correctly in the presence of up to \( m \) processor failures (where \( N \) is the number of individual processors in the node, \( = 2^m + 1 \)). The mechanism is similar to that of fail–silent nodes, however, by using more than two processors, failures may not only be detected, they may be masked as well.

The failure behaviour of a node dictates the application–level fault–tolerance techniques which may be used in conjunction with it. An application executing over failure–masking nodes will not require any mechanism for tolerating processor failures. If, however, the application executes on fail–silent nodes, replication of application–level processes, or a mechanism such as checkpoint with re–start, will be required. Such mechanisms are, however, relatively easy to construct when compared with a system based on fail–uncontrolled nodes.

2.2 Hard & Soft Fail–Controlled Nodes

Fail–controlled nodes may be categorized as either hard or soft, depending on the approach taken to their implementation, with each technique having its advantages and disadvantages.

Hard Fail–Controlled Nodes

Hard fail–controlled nodes adopt a primarily hardware–oriented approach to ensuring predictable failure behaviour. Typically, two or more identical processors (each with its own memory and other peripherals) execute the same software in lock–step after being initialised to identical states. A single, dependable clock is used to drive all processors, thus ensuring that their states remain identical. The outputs from each processor are fed into a special–purpose comparator, where any discrepancies in the outputs indicate that a failure has occurred. On detection of a failure, the comparator will follow a pre–defined course of action, possibly masking the failure if a majority of the processors remain in agreement, or causing the node to stop if a consensus can no longer be obtained. Comparators may also implement functions such as node–reinitialisation, fault diagnosis or failure notification.

Hard fail–controlled nodes are able to offer very high levels of dependability. No failure–control mechanism is perfect, however, and there remains the possibility that uncontrolled failures may occur under certain circumstances. For example, the comparator represents a potential single point of failure, although comparators are usually designed to be very simple and inherently fail–controlled with high probability. Further, common node failures may cause multiple processors to produce similar, but nonetheless incorrect results. Often such failures will be the result of errors in the design or implementation of a processor, however, they may also occur under other circumstances (e.g. a power surge or brown–out affecting processors in an identical manner).

Hard fail–silent nodes are already widely used in aerospace applications, and represent an effective means of achieving fail–controlled behaviour in safety–critical systems. Due to the need for special–purpose hardware, and the fact that hard–fail–controlled nodes must currently be designed specifically for a given processor, however, hard fail–controlled nodes are costly. (This may change as future generations of processors begin to take account of support for dependability in their designs, as does, for example, the Intel 960.)

Soft Fail–Controlled Nodes

A soft fail–controlled node is a logical entity achieved by running special software communications protocols on standard, fail–uncontrolled processors. Application processes run in parallel on two or more processors within the node, with message redundancy (in the form of checksums or digital signatures) being used to authenticate the outputs of process replicas. The mechanism ensures that any invalid messages produced by a node may easily be detected to be invalid and discarded by a recipient.

The main advantage of this approach is that it can make use of standard host computers, with no special–purpose hardware, and is thus a much lower cost solution than hard fail–si-
lence. In addition, it may be possible to employ design diversity techniques (using processors of different types and/or multiple versions of the application software within a node) in order to reduce the potential for common-mode failures, and the fact that the processors within a fail-controlled node are only loosely synchronized (compared with many hard fail-silent designs) makes soft fail-controlled nodes more robust in the presence of transient faults.

There are, however, some drawbacks to the technique, which may make it inappropriate for some safety-critical applications, or applications with very demanding timing constraints. The main disadvantage is that, since no special fail-silent hardware is employed, a failed processor (or its network interface) can, in principle, flood the network with erroneous messages. Although these messages will be rejected by any recipient as invalid, this may well lead to a reduction in available network bandwidth to an extent which may lead to missed deadlines. In contrast, hard fail-controlled nodes contain hardware-level measures intended to prevent such behaviour. This drawback of soft fail-controlled nodes may, however, be countered through the use of a limited degree of special-purpose hardware. Section 4 gives an example of such a mechanism in the context of IMA.

3. Soft Fail-Controlled Node Design

In those systems where soft fail-controlled nodes are to be employed, the main problem which must be addressed is that of synchronizing the execution of the replicas within a node and ensuring that their internal states remain consistent. In a hard fail-controlled system, this problem is solved by maintaining a tight degree of synchronization between the processors (which run in lock-step), however, in a soft fail-controlled node, this degree of synchronization is not feasible, leading to the possibility that different replicas may perform different computations and exhibit divergent behaviour. To prevent this, specialized communications protocols must be used to ensure that the processing carried out by different replicas remains consistent, and the individual replicas themselves must conform to an appropriate model of computation consisting of processes, each structured as a state machine, communicating via messages.

The State Machine approach is one of the most widely used models for active replication. A state machine consists of state variables which encode its internal state and commands that transform that state and produce output. Commands must be implemented by deterministic programs and the execution of any given command is atomic with respect to all others. Given these properties, active replication can then be supported by adopting communications mechanisms which ensure that all replicas process the same set of input messages in the same order. Thus, the communications system must possess the properties of agreement (all non-faulty replicas receive identical input messages) and order (all non-faulty replicas receive these messages in the same order).

3.1 The Voltan Fail-Silent Node

A Voltan fail-silent node is composed of two conventional (fail-arbitrary) processors and it implements soft fail-silence: that is, it either produces correct messages which can be verified as such by the receivers, or it ceases to produce new correct messages. This behaviour is guaranteed so long as no more than a single processor in a node fails. Further, any spurious messages produced by the failed processor in a failed node can be detected as such and discarded by all correctly functioning receiver nodes. Each processor in a Voltan fail-silent node runs identical copies of the application processes assigned to that node, along with a number of system processes that help maintain replica synchronization and provide necessary communications support. The individual outputs of the replicated application processes are compared and a valid, combined output message only generated if the comparison succeeds.

Voltan assumes a computational model that is largely based upon the state machine. It is assumed that computations are composed of a number of processes that interact only via messages, and that the computation performed by a process on a selected message is deterministic. However, since practical distributed programs often require additional functionality such as timeouts (when waiting for messages) or the ability to handle asynchronous events, Voltan also includes mechanisms for dealing with such potential sources of non-determinism.

It is assumed that the originator of a message can be authenticated by a non-faulty receiver (using, for example, a digital signature mechanism) and all nodes in the system are presumed to possess a unique identifier (a number). Each group of duplicated computational processes is also presumed to possess a unique group identifier and each individual process is assumed to maintain a local counter variable (initialised to zero) which is used for the generation of sequence numbers for the messages produced by that process. A process generates a sequence number for a message by concatenating its host identifier, its group identifier and its local counter value, after which the counter is incremented by one. Hence, correctly functioning replicas of a process will produce messages with identical sequence numbers (making message selection for comparison relatively easy) while the sequence numbers for different messages (either from the same processes or different ones) are guaranteed to be unique.

The aforementioned system processes that are run on each individual processor in a Voltan fail-silent node are as follows:

(i) **Diffuser Process**; This process takes the messages produced by the computational (application) processes running on a processor, signs them and sends them to the other processor in the same node.

(ii) **Receptor Process**; This process accepts only authentic messages from the network for processing. Messages with single signatures are sent to the local comparator process, while messages with two distinct signatures are sent to the local order process for distribution to the appropriate local application processes.

(iii) **Comparator Process**; The comparator processes sign and verify local messages coming from the receiver process (i.e., potential output messages generated by the other processor in the same node). If the contents of such a
message are identical to those of the corresponding message produced on the comparator's own processor, then the message is countersigned and sent to the local transmitter process for transmission over the network to its final destination. However, if the message cannot be matched, then it is not countersigned and the comparator process halts itself in an attempt to prevent any further doubly-signed messages being produced. It follows that all correct (valid) messages issuing from a node will be double signed and any message which is not double signed can be treated as erroneous. (This allows receiving nodes to identify and trap malicious messages produced by a single failed processor in any other node since such messages cannot be double signed.)

(iv) **Transmitter Process**: This process transmits valid messages coming from the comparator process to their destinations.

(v) **Order Process**: This process executes a special order protocol with its counterpart running on the other processor in the same node. This permits the order processes to construct identical queues of valid messages for processing, thereby ensuring that application process replicas will make a consistent choice of request to execute.

These processes are illustrated diagrammatically in Figure 1, which shows the overall software architecture of a Voltan node. The major system components that run on an individual processor in a Voltan fail-silent node and their interactions are all shown. There are two main layers: the replication layer ensures that all message interactions of the application processes are agreed and ordered at the replicas, while the communications layer provides the atomic broadcast and other message communications facilities.

The various message pools and queues shown in the diagram are:

(i) **Received Message Pool (RMP)**: Contains valid received messages intended for ordering.

(ii) **Processed Message Pool (PMP)**: Contains unsigned output messages produced by application processes. These messages must be validated by the comparator before transmission to their final destination.
(iii) External Candidate Message Pool (EMP): Contains singly signed messages that have been received for validation.

(iv) Internal Candidate Message Pool (IMP): Contains unsigned messages, each waiting for a signed message with identical sequence number to arrive in EMP.

(v) Delivered Valid Message Queue (DMQ): An application process is only ever presented with a single choice of message to process, the one at the head of its DMQ. The order processes running on a node then ensure that the DMQs of replicated application processes running on the processors in that node remain identical. Hence, replicas will process the same set of requests in the same order.

(vi) Broadcast Message Queue (BMQ): The broadcast message queue is maintained by the order process and is an ordered queue of requests to be delivered to the appropriate DMQs of the application processes. (Note that the BMQ may contain duplicates, but that the order process will deliver a duplicate message when it attempts to deliver it from the BMQ to the appropriate DMQ. Such duplicates are discarded.)

The fail-silent Voltan node is only one of the possible Voltan architectures. By replacing the simple comparator process described earlier with a more sophisticated one that is capable of majority-voting a set of outputs, a failure-masking, TMR Voltan node (containing three processors rather than two) can be constructed. Such a processing node will be capable of tolerating one processor failure without suffering any loss of service. However, note that the order protocol required will be more complex, being based on the concept of fault-tolerant message broadcast. This is because the TMR versions of the protocols need to be able to continue operation in the presence of a failed processor, whereas the versions used in the fail-silent node are only expected to work in the absence of failures.

4. Fail-Controlled Nodes and Integrated Modular Avionics (IMA)

4.1 Overview of IMA Hardware Architecture

IMA is not itself an architecture. Rather, it defines a set of components and methods of interconnection which a designer may use in the construction of an architecture appropriate to his requirements. (See references 1, 8, 9 for further information on IMA.)

A typical IMA architecture comprises a number of computing resources, located throughout an aircraft and interconnected via a power bus and an ARINC 629 data bus. "Smart" sensors and actuators may be connected directly to the 629 data bus, whilst other devices may be connected to the system via utilities data busses and remote data concentrators. The IMA hardware components are as follows (see Figure 2):

- A Power bus, which distributes power to the computing resources.
- ARINC 629 data bus, which interconnects computing resources, remote data concentrators and smart sensors and actuators.
- Computing resources, each of which comprises:
  - A power supply module which draws power from the power bus and distributes it internally to the computing resource via a 609 power bus.
  - A 659 backplane bus which interconnects gateway, I/O and core modules.
  - A gateway module which provides the interface between the computing resource and the 629 data bus.
  - An I/O module which provides an independent I/O facility to the computing resource.
  - A number of "core" modules providing the processing capability of the computing resource. Each core module contains processor, memory, clock, I/O and fault detection and correction components.

![Figure 2. IMA Hardware Components](image)
Smart sensors and actuators. These devices connect directly to the 629 data bus.

- Remote data concentrators. The remote data concentrators provide the interface between the IMA system (i.e., the 629 data bus) and devices other than the smart sensors and actuators. Each remote data concentrator provides a utilities data bus, to which devices (e.g., analogue and digital sensors, panel switches and indicators, power switching actuators, system sensors etc.) are connected via “remote terminal” modules.

4.2 Voltan and IMA

IMA systems are constructed from computing resources (multi-processor nodes) interconnected by a local area network (the 629 data bus). Internally to each node, individual processors communicate with one another over a backplane bus, and with the network via a gateway module (see Figure 3).

This structure is sufficiently close to that described in Section 3 for the techniques presented there to be applied to IMA. IMA nodes may contain more than 2 processors, however these processors may be logically paired in order to provide the abstraction of a number of fail-silent nodes contained within each computing resource. (In order to avoid confusion, in this discussion of IMA the term node will be applied to a logical fail-silent node, whilst the term computing resource will be used to apply to the IMA component of that name. It will be assumed that each node is contained within a single computing resource, thus minimising traffic on the 629 data bus.) An example of the resulting fail-controlled IMA architecture is shown in Figure 4.

The protocols used to implement fail-silence are described in Section 3. Where application processes are replicated on a number of fail-silent nodes (to achieve availability as well as integrity), messages between nodes will have to be sent across the network using a reliable multicast protocol in order to ensure that either all destinations of a message receive that message, or none do. Reliable multicasting may be implemented in conjunction with the above fail-silence protocols (see, for example,10) using mechanisms such as the following:

In the absence of failure, two copies of each message will be transmitted across the network to arrive at their destination nodes. Should a failure occur within the sending node whilst a message is being transmitted, some or all processors may fail to see one or both of the copies of the message. Given that nodes are fail-silent (to the extent that the messages they send are either correct or detectably incorrect) the following situations are possible:

- No node receives either copy of the message. No problem here; the sender failed silently before the message was transmitted.
- All destination nodes receive both copies of the message. Again, no problem. The multicast has completed successfully despite the failure.
- A subset of destination nodes has received one copy of the message, but not the other. Under these circumstances, it is possible that other destination nodes may have received neither copy, and the conditions for reliable multicast have not been fulfilled. In this case, the nodes receiving only one copy of the message will take it upon themselves to complete the multicast by transmitting a further copy of the message themselves. This protocol will be repeated (if necessary) until all destination nodes have received two copies of the message.

IMA systems will be used to support a wide range of services aboard an aircraft. Some of these services may be safety critical, others less so. Some may have demanding timing constraints, whilst others should be restrained in their use of system resources. The attributes which may be required of an application therefore include:

- Very high levels of dependability.

One of the potential drawbacks of the Voltan approach is that Voltan nodes are not, in fact, “fail-silent”. What they actually provide is an abstraction of fail-silence in which any messages sent are either correct, or are detectably incorrect. Thus, there remains a remote possibility that a faulty node may transmit large numbers of incorrect messages to other nodes, thus increasing the load on those nodes and the network.

IMA, however, provides a means of addressing this problem through the introduction of appropriate intelligence into the computing resources’ gateway modules. The possibility that significant numbers of erroneous messages may be generated by a failed node may be countered simply by having the gateway modules detect and discard erroneous messages, thus ensuring that such messages never leave the computing resource in which they were generated. In this way, a worst-case processor failure could affect processing in the rest of its computing resource, but not elsewhere in the system. If critical application processes were to be replicated on nodes in different processing resources, the application itself could survive such failures.

Clearly, given their critical role in interfacing computing resources to the network, the gateway modules them-
selves must be made dependable, and must be implemented using fail-controlled techniques (see Section 4.3).

- Ability to meet strict timing constraints

The ability of the IMA/Voltan architecture to meet strict timing constraints will be limited by a number of features of the architecture:

- Both transmission and reception of messages requires the collaboration of both processors of the source / destination nodes respectively. Since nodes are only loosely synchronised, this introduces worst-case delays equivalent to maximum possible processor desynchronization at both sender and receiver.

- The reliable multicast protocol described relies on one or more receivers of a message seeing only a single copy of that message, assuming that the sending node is faulty, and completing the protocol themselves. Timeouts, which must exceed the maximum possible de-synchronization of the sending node, are required to detect that a copy of the message has been lost. Note that if multicast communications were to be provided via a different mechanism this problem could be circumvented.

- Following a failure during the multicasting of a message, each of the receivers of that message who have seen only a partial multicast will attempt to complete the protocol. If the number of receivers was large, this could well result in several receivers all trying to complete the multicast, thus increasing the load on the network and on the receivers themselves. Again, this problem could be avoided by adopting a different approach to providing multicast communications.

Thus, although we may reasonably expect to be able to bound message delivery times, these bounds may be excessive for certain applications demanding very fast responses unless an efficient multicast communications mechanism employing special-purpose hardware (such as that used by Delta-4 XPAH) is provided.

- Use of environmental resources.

This is unlikely to be a significant issue in IMA applications, which will typically be installed in relatively large commercial aircraft. It should not be completely overlooked, however. The IMA/Voltan mechanisms proposed above would seem to make fairly efficient use of resources, and there are no major concerns on this front.

Thus, it would appear that Voltan-like soft-fail-controlled mechanisms may be incorporated into IMA-style systems with relative ease, and would function well in that environment. Whilst there may be doubts about using soft-fail-controlled nodes for safety-critical applications, the flexibility of the Voltan approach is likely to find favour for some of the less critical, but still important, applications which must be provided on a modern aircraft. For critical applications, Voltan-style mechanisms may still be employed if an enhanced IMA gateway module is used to provide the required dependable communications and error-detection services.

Figure 4. Example Fail-Controlled IMA System
4.3 The IMA Gateway Module

The gateway module in IMA provides an interface between the 629 data bus and the backplane bus of a computing resource. It is thus ideally placed both to offer communications services and to provide a check on the data being transmitted onto the 629 bus by a computing resource. This section examines some of the ways in which the functionality of the gateway module may be extended in order to enhance the performance of an IMA system.

Communication Services

One way in which the gateway module may be exploited to the benefit of the IMA system is to regard it as a provider of communications services. IMA applications will usually be replicated for availability, with replicas running in different computing resources in order to allow the application to survive the failure of a computing resource. If the application is such that it, or its constituent processes, must communicate with processes elsewhere in the system, some form of multicast protocol will be required in order to ensure that all replicas receive all messages addressed to the application. Each application could, of course, implement the communications mechanisms it requires itself, however, this is inefficient and unnecessarily complicates the applications. It is generally better that the communications services required are provided by the “system”, and for IMA the gateway module is an obvious candidate for this role.

Data Validation

The second service which may be provided by the gateway module is that of validating the data placed onto the 629 bus by the processes running within a computing resource. There are several ways in which the gateway module could help in this regard:

- Signature/checksum validation. Messages will normally be signed (or check-summed) at source in order that any message corrupted in transmission may be detected as invalid at its destination. For any messages transmitted via the services of the gateway module, these signatures/checksums may be checked before the message is placed onto the network, thus preventing invalid messages from being transmitted. This check would eliminate one of the potential problems with soft-fail-controlled node techniques; that failed nodes may continue to generate (detectably) invalid messages, thus consuming system resources. If such messages could be trapped within the gateway module their potentially damaging effects would be restricted to a single computing resource.

- Replicate message elimination. Applications executing on soft-fail-silent nodes containing two or more loosely-synchronised processors will generate replicated outputs which must be reduced to a single message at some point in their transmission. Elimination of duplicate messages can be carried out on reception at their destination, however, where replicated outputs take the form of messages to be sent across the 629 data bus, the gateway module may be used to filter out the replicas, deleting them without tying up resources on the network and at the destination.

- Watchdog function. The gateway module may also be used to implement a range of watchdog functions for a computing resource. For example, core elements may be connected directly to the 629 data bus; there is no requirement for communications to go through the gateway module. Under such circumstances, the gateway module cannot directly validate messages before transmission. It can, however, monitor the network for incorrect or untimely transmissions and carry out remedial procedures should any such transmissions be detected.

Using the gateway module to provide message validation and multicast communications functions allows the protocol required to implement soft-fail-silence to be much simplified, resulting in a significant reduction in end-to-end message delivery times.

Gateway Module Dependability

One effect of enhancing the functionality of the gateway module as described above is to increase its importance to the correct functioning of the system as a whole; it becomes essential that the gateway module itself exhibits fail-controlled behaviour with very high probability since its failure could potentially have implications for the integrity of the entire system. One way of achieving this would be to implement the gateway module as a hard-fail-silent component. The result is something of a cost trade-off; increased complexity (and thus cost) in the gateway module allows the use of fail-uncontrolled processors within computing resources, and increases the scope and flexibility of the system for supporting dependable applications. A similar approach was adopted by the Delta-4 project with its Network Attachment Controller, and has been shown to work well in practice.

Alternatively, soft-fail-silence techniques may be applied to the gateway module in the same way as to a normal processing node. Used in isolation, such an approach would not prevent the possibility of a failed gateway module processor flooding the network with erroneous messages. Used in conjunction with a mechanism which detects gateway module processor failures, however, the problem of network flooding can be avoided. Such mechanisms will, of necessity, include a minimal amount of special-purpose hardware. They may comprise a means of monitoring the messages placed onto the network by the individual processors of a soft-fail-silent gateway module, disconnecting the gateway module from the network should any anomalies be detected, or they may involve a mechanism whereby a non-faulty gateway module processor may detect the failure of another gateway module processor (through anomalies in the stream of messages received from that processor), shutting down or disconnecting the gateway module should such a failure be detected.

5. Conclusions

We have described mechanisms for controlling the failure characteristics of a computing node using replication. The principles for building soft-fail-controlled nodes are applicable to general purpose hardware. Hence, the work described here can be regarded as representing the limits of
what can be achieved using standard “off the shelf” components to construct highly reliable nodes as the building blocks of distributed systems.

In section 4 we discussed the applicability of this approach to IMA architectures. It is shown that the structure of soft fail-silent Voltran nodes is quite compatible with planned avionics architectures. We have also shown how judicious use of the IMA gateway module can enhance the reliability and performance of such systems. This approach follows that adopted by the Delta-4 project. In the Delta-4 architecture, software techniques are combined with a small amount of special purpose hardware (the Network Attachment Controllers) in order to produce reliable distributed systems. Here, we have software implemented fail-silent nodes with gateway modules requiring some degree of specialised hardware support. This approach would appear to be a reasonable compromise between hardware and software techniques for increasing the reliability of avionics architectures.

Acknowledgements

This work has been supported by the Science and Engineering Research Council and British Aerospace.

References
