Fault-Tolerance in Air Traffic Control Systems

FLAVIU CRISTIAN
UCSD
and
BOB DANCEY
SAIC
and
JON DEHN
Lockheed/Martin Air Traffic Management

The distributed real-time system services developed by Lockheed Martin's Air Traffic Management group serve as the infrastructure for a number of air traffic control systems. Either completed development or under development are the US Federal Aviation Administration's Display System Replacement (DSR) system, the UK Civil Aviation Authority's New Enroute Center (NERC) system, and the Republic of China's Air Traffic Control Automated System (ATCAS). These systems are intended to replace present en route systems over the next decade. High availability of air traffic control services is an essential requirement of these systems. This article discusses the general approach to fault-tolerance adopted in this infrastructure, by reviewing some of the questions which were asked during the system design, various alternative solutions considered, and the reasons for the design choices made. The aspects of this infrastructure chosen for the individual ATC systems mentioned above, along with the status of those systems, are presented in the Section 11 of the article.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*reliability, availability, and serviceability*; D.2.5 [**Software Engineering**]: Testing and Debugging—*error handling and recovery*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; *hierarchical design*; *real-time and embedded systems*; H.1 [**Information Systems**]: Models and Principles; J.7 [**Computer Applications**]: Real Time—*air traffic control systems*

General Terms: Design, Reliability

Authors' addresses: F. Cristian, Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114; email: flaviu@cs.ucsd.edu; B. Dancey, SAIC, 10260 Corporate Point Drive, San Diego, CA 92121; email: bob_dancey@cpqm.saic.com; J. Dehn, Lockheed/Martin Air Traffic Management, 9231 Corporate Blvd., Rockville, MD 20872; email: dehn@lfs.loral.com.

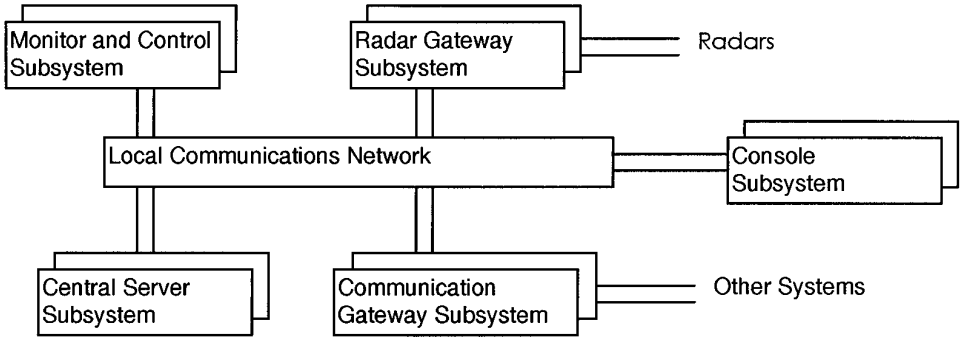

Fig. 1. En route system components.

Additional Key Words and Phrases: Exception handling, failure, failure classification, failure masking, failure semantics, fault-tolerant systems, group communications, redundancy, server group, software robustness, system architecture

1. INTRODUCTION

The goal of an en route system is to provide Air Traffic Control (ATC) services to its end-users: the air traffic controllers. (Aircraft are under control of an *en route* system from soon after takeoff until shortly before landing.) Some of these services, such as radar data reception and display, are critical, in that if they are not available the controllers cannot carry out their duty normally. Typical en route system requirements specify that critical services should not be unavailable more than three seconds per year. Other services may have a limited unavailability of up to 156 seconds per year. Coupled with these availability requirements are performance requirements to handle anticipated air traffic workloads for the next few decades (several thousand radar reports per second, with each report being processed and displayed within one second or less of receipt at the system boundary). Since the unavailability of existing commercial single-fault-tolerant systems is measured in tens of minutes or hours per year, the above very stringent availability requirements make these en route systems beyond the current state of the art. To achieve these requirements, new design techniques, which enable the system to automatically mask *multiple* concurrent component failures, had to be used.

The infrastructure was originally developed for the FAA and has subsequently been reused in other air traffic control systems. This article describes the infrastructure in general terms, not necessarily as it is being used on any particular system. For a more detailed look at a particular use of this infrastructure, see Debelack et al. [1995]. Although developed for use in en route air traffic control systems, the underlying infrastructure could be reused in other application domains as well.

2. AIR TRAFFIC CONTROL SERVICES AND SUBSYSTEMS

An en route airspace controlled by a single facility is administratively divided into sectors. As an example, the US airspace is serviced by 20 facilities, each with a design limit of 118 sectors per facility. Each facility receives aircraft surveillance and weather data from radars and communicates with other external systems such as other en route facilities, terminal area facilities (controlling airspace around airports), and airport towers (controlling aircraft at the airports themselves); see Figure 1. All facilities adhere to the standard time broadcast by the WWV radio station of the National Bureau of Standards. The ATC infrastructure services described here are scaleable to any facility requiring highly available end-user services with stringent response times, from systems with a handful of nodes to those with several hundreds of nodes.

Air traffic control services are geographically partitioned among the various subsystems mentioned above. For example, the Surveillance Processing service (which receives radar data and correlates it with individual aircraft tracks) is hosted by radar gateway processors with direct physical access to radar input. CPU and storage-demanding ATC services, such as Flight Plan Processing and Conflict Alert, are hosted by centralized processors. The Display service (which inputs and interprets commands entered by air traffic controllers, manages all data characterizing the state of an air traffic sector, and displays portions of it as directed) is hosted by console processors directly connected to the physical displays and keyboards used by controllers.

Redundancy of components is used inside each subsystem to ensure that it provides its specified services despite failures or changes affecting individual subsystem components. The goal is to mask such events from subsystem users. The remainder of this article discusses the issues that had to be addressed in managing this redundancy. Since presently there is no agreement on terms when it comes to system structuring and fault-tolerance concepts, we first define the basic architectural building blocks of the ATC infrastructure, classify the failures that these building blocks can experience, and introduce the general approach to fault-tolerance adopted. We then investigate the issues that had to be addressed to provide high availability at the various abstraction levels of an en route system, from hardware to the software applications.

3. BASIC SYSTEM STRUCTURING CONCEPTS

The ATC infrastructure services are structured in a “depends” relation as shown in Figure 2. This acyclic graph shows the various resources that make up and are managed by the infrastructure. This definition uses the concepts of servers, services, and “depends” relation as described in Cristian [1991]. The infrastructure allows the hardware to exhibit crash/omission semantics, as discussed in Section 5.2 (Cristian [1991] also defines various failure semantics, such as crash, omission, and performance, used in this article).

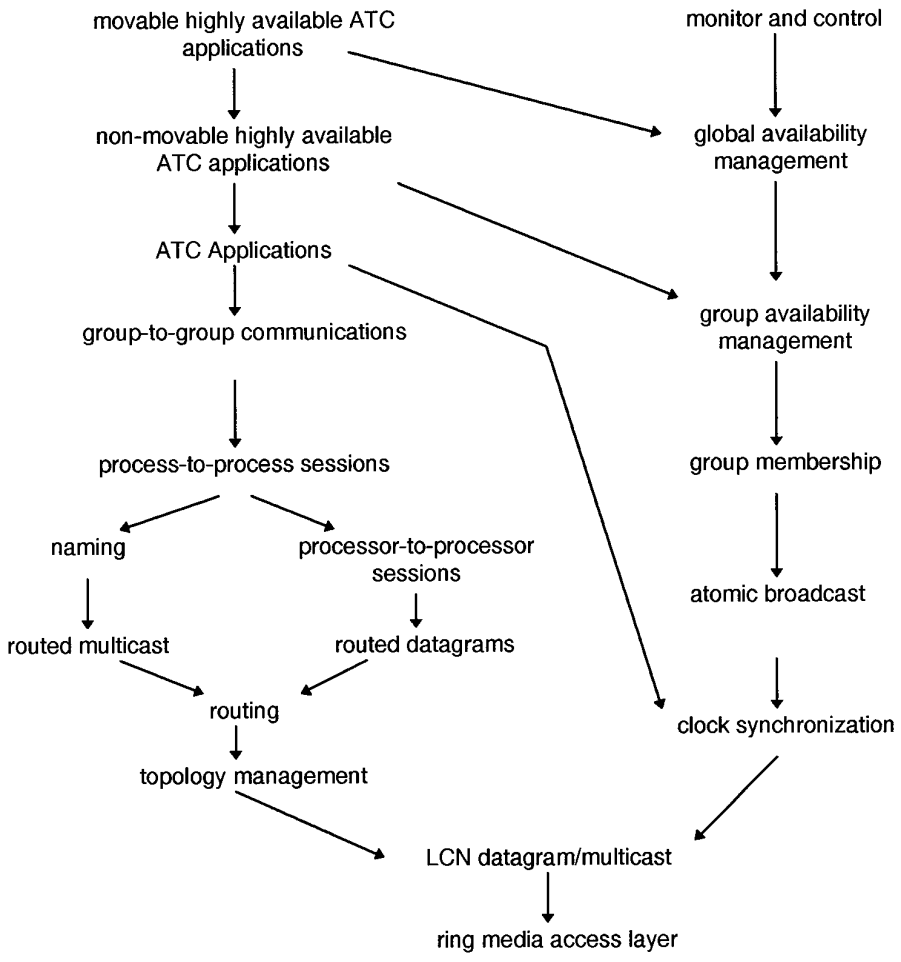

Fig. 2 Depends-on hierarchy.

Note that in Figure 2 the ATC *applications* are shown generically. For each implementation of an air traffic control system that uses this infrastructure, other "depends" relations are created for the specific set of applications required in any specific system.

To ensure that a service remains available to clients despite server failures, the ATC infrastructure implements the application services in *groups* of redundant, physically independent servers. If some of these servers fail, the remaining ones provide the service. These primary/standby groups of k servers with crash/performance failure semantics, with members ranked as primary, first backup, second backup, . . . , $(k-1)$th backup, can mask up to $k - 1$ concurrent member failures and provide crash/performance failure semantics. (Concurrent in this sense must take into account the mean time to repair a failed group member. If a second member fails before the first member is repaired, the second failure is considered to be concurrent with the first.)

The specific mechanisms needed for managing redundant server groups in a way that masks member failures and at the same time makes the group behavior functionally indistinguishable from that of single servers depend critically on the failure semantics specified for group members and the communication services used. The stronger the failure semantics of group members and communication is, the simpler and more efficient the group management mechanisms can be.

Since it is more expensive to build servers with stronger failure semantics, but it is cheaper to handle the failure behavior of such servers at higher levels of abstraction, a key issue in designing multilayered fault-tolerant systems is how to *balance* the amounts of failure detection, recovery, and masking redundancy used at the various abstraction levels of a system, so as to obtain the best possible *overall* cost/performance/dependability results. Deciding to use too much redundancy, especially masking redundancy, at the lower levels of abstraction of a system might be wasteful from an overall cost/effectiveness point of view, since such low-level redundancy can duplicate the masking redundancy that higher levels of abstraction might have to use anyway to satisfy their own dependability requirements.

4. HOW SHOULD FAILURES BE HANDLED IN ATC SYSTEMS?

This global question can be broken down into two orthogonal questions. First, at what abstraction level should replication be used to automatically mask component failures to human users? And second, what class of component failures should the system be able to mask automatically?

4.1 At What Level of Abstraction Should Automatic Failure Masking Occur?

To mask software as well as hardware server failures in a uniform way to end-users, one has to use redundancy at the highest level of abstraction: the application level. Otherwise, a common software fault could manifest itself on all hardware servers simultaneously, leading to a complete failure of the end-user service. The idea is to implement any service that must be available to its users despite hardware or software failures by a redundant software *server group* whose members run on distinct hardware processor hosts and maintain redundant information about the service state. When a group member fails (either because of a lower-level hardware or software service failure or because of a residual design fault in its program) the surviving group members have enough service state information to continue to provide the service without interruption. This complicates somewhat the development of applications, especially when they make use of persistent data. However, since the specification for ATC systems requires that the system be able to automatically mask both hardware and software failures [Avizienis 1987], and the very stringent availability requirements practically forbid users from observing situations in which there exist no operational servers for a critical ATC service, the decision was taken to use application server groups and to rely on systematic hierarchical methods

for exception handling [Cristian 1989c] to mask lower-level failures to application servers whenever economically appropriate.

4.2 What Failure Classes Should Be Automatically Masked?

The redundancy management mechanisms designed for managing server groups with arbitrary-failure semantics are not only more costly and complex than those designed for managing groups with stronger failure semantics, such as crash, omission, or performance, but also lead to higher run-time overheads and slower response times.

Statistical evidence [Gray 1986] indicates that group-masking mechanisms designed for software servers with crash/performance failure semantics can be effective in providing tolerance to server failures caused either by hardware server failures or by residual design faults left in production-quality software after extensive reviews and testing. Less than 1% of the failures reported by Gray [1986] were double server failures, that is, group failures. A plausible explanation for this phenomenon is that most transient faults affecting physically independent processors occur independently, and hence cause any affected software servers to crash independently. Similarly, most residual software design faults manifest themselves intermittently in rare limit (or exceptional) conditions which result in time-dependent synchronization errors or in a slow accumulation of resources acquired for temporary use and never released. Such errors and residues seem to accumulate at different rates in operating system tables and group members running independently on different processors and eventually lead individual group members to fail at different times.

Faced with the choice of adopting either redundancy management mechanisms for servers with arbitrary-failure semantics or mechanisms for servers with crash/omission/performance failure semantics, in view of the accumulated experience described above, the decision was taken to go with the latter alternative. With this choice, we provide automatic masking at a reasonable cost for the vast majority of component failures likely to occur during the lifetime of the system. As highlighted by Gray [1986], such mechanisms can also be effective in providing tolerance to a significant portion of, although not all, arbitrary failures which are likely to occur because of imperfect coverage by the hardware error-detecting codes and the software checking used. The hopefully rare arbitrary application failures not detected and masked automatically by these mechanisms will ultimately be detected by the human users: the air traffic controllers who know the physical laws of the airspace environment they are controlling and can diagnose abnormal system behavior. In such a case, a controller can unilaterally decide to switch to conservative aircraft separation standards (informing pilots of this decision through a physically separate ground-to-air voice system) or to switch to an alternate service provided by a *backup system* of a different, very simple design. This alternate system is capable of providing *emergency* service until the faulty components of the *main* en route system that caused the user-visible failure behavior are repaired. This backup system is also used when it is necessary to

upgrade the software or hardware components of the primary system. More about this backup system will be given in Section 10. Although no attempt is presently planned for automatically masking all arbitrary application server failures to users, many of these failures are detected automatically. Detection of such a failure results in the shutdown of application servers which display the errant behavior. This decision is consistent with a growth-oriented system design philosophy, of implementing efficient automatic means for detecting design defects and offering system maintainers help for diagnosing and correcting problems without causing service interruption to the end-users.

Our decision to base the automatic redundancy management mechanisms on the hypothesis that software servers will suffer crash/performance failures results in two subsidiary system-wide objectives: ensure that (1) the hardware which will be used has, with high probability, crash/omission failure semantics and (2) the programs which implement software servers are partially correct [Cristian 1989c]; that is, in the absence of failures of the servers they depend on, these programs can only crash or be slow, but the probability of a program providing erroneous answers is negligible. To prevent arbitrary failures of software servers, extensive use is made of modern software engineering practices, such as pursuance of simplicity in design, hierarchical decomposition of the software by using information hiding and abstract data types and objects, disciplined detection and handling of all exceptions, and extensive design and code inspections. In particular, design and code inspections focus on those areas that may lead to a common failure in all members of the server group (specifically, areas where the same input is processed in parallel against a common view of the server's state) and in areas that may result in slowly degrading behavior (such as memory "leaks"). A suitable programming language (Ada) was selected to enforce as much error checking as possible at compilation and execution time, and the standard compilation facility was augmented by additional code analysis tools.

4.3 What Mechanisms Are Needed for Implementing Redundant Application Software Server Groups?

First, one needs to decide what *processor services* should be provided to application servers, so as to satisfy both their response time requirements under the specified maximum load and the requirement that processor services have crash/performance failure semantics. To make the behavior of server groups indistinguishable from that of nonredundant servers one has to provide group members with communication primitives enabling them to communicate with other groups and reach agreement on the state of the service they implement despite random communication delays and failures. Additional services are provided on top of the communication primitives, such as a service that allows a new instance of a program to join a software server group and acquire the group's current state. The semantics of the ATC applications requires that group communication be of two kinds: either group-to-group request/reply or one-group-to-many groups multicase ( ). (See Figure 3.) An important issue is that of how to decide on the

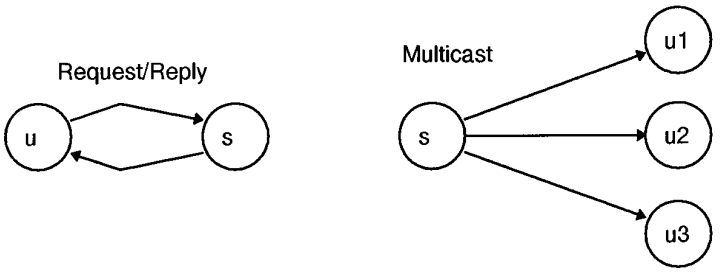

Fig. 3. Group communication paradigms.

availability policy to be enforced for each software server group, i.e., how *closely synchronized* should the local states of members be, and *how many members* should each group have? A related issue is to design *group availability management* mechanisms that will automatically enforce the availability policies defined for application groups.

We list below some of the key choices that were made when designing the automatic fault-tolerance mechanisms that are employed at the various abstraction levels of an ATC system.

5. THE PROCESSOR SERVICE PROVIDED TO APPLICATIONS

In view of the load and response time requirements specified for ATC systems and the requirement that off-the-shelf basic hardware and software components be used extensively to run all specified ATC applications, one of the earliest decisions taken was to use RS/6000 processors for all processing nodes. The RS/6000 family supports a wide range of external attachments and processor speeds, so that the diverse requirements of the external system gateways, the central servers, and the workstation processors can all be accommodated within this processor family. In practice, the system design isolates hardware dependencies to the lowest level possible so that other POSIX-compliant processors and operating systems could be substituted with minimal effort.

These processors are interconnected by a local communication network (LCN) based on IEEE 802.5 token rings. Since an en route ATC system is required to accommodate more processors than a single token ring can support, classes of similar processors are clustered by attaching them to distinct horizontal *access token rings* (Figure 4). The set of processors attached to the same access ring forms a *cluster*. Clusters are connected among themselves by a vertical *backbone* ring structure (Figure 4). Specialized hardware servers, called bridges, are used for relaying messages between access and backbone rings. In Figure 4, bridges are shown as either hollow or filled squares. The "filled" bridges are normally actively passing data between access ring and backbone ring. The "hollow" bridges are redundant components that will be used in the event of a bridge, backbone ring, or access ring failure.

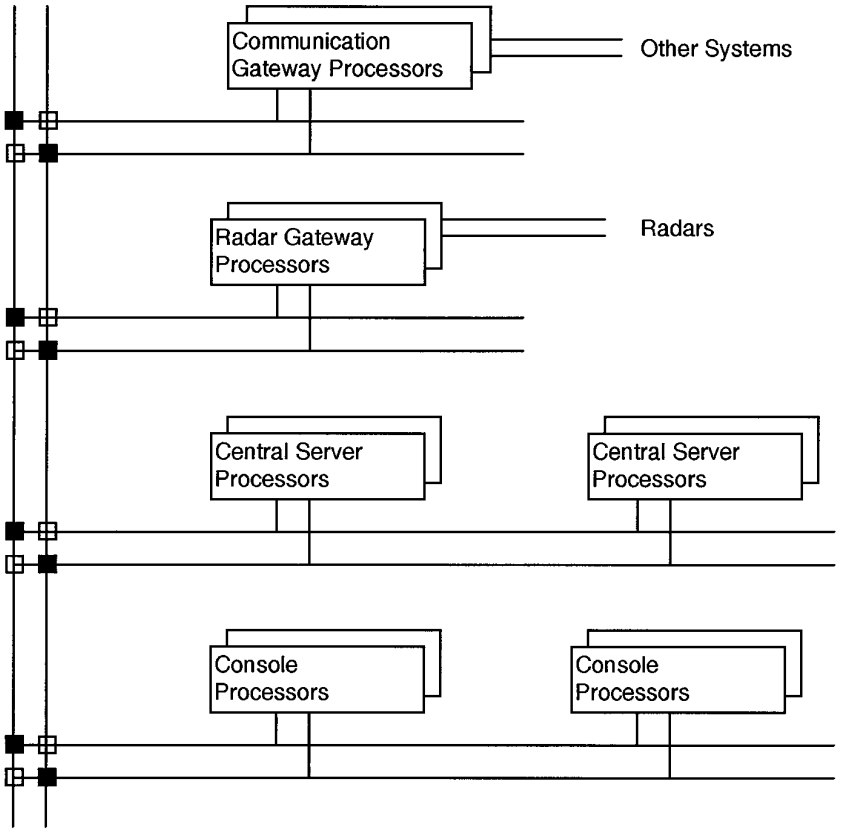

Fig. 4. En route system network topology.

Using this configuration of bridges allows message traffic to be routed around any single communication subsystem component failure and around many multiple failures.

The chosen hardware can not only handle the specified peak ATC load, but can also interface with a variety of specialized I/O servers required for the ATC applications, such as large workstation displays required for this application, radars, and foreign communication networks. The hardware, which in the past has been characterized by significant mean times between failures and observed crash/omission failure behavior, was considered to provide an acceptable cost/performance ratio. The adoption of a POSIX-compliant UNIX operating system (AIX), including the real-time features necessary to meet the system's performance requirements, allows the customer to take advantage of the growth in reliability that comes from many users and the software tools already existing for these systems.

The choice of physical media (in this case, token rings) is also isolated to a few number of services. Other network technologies such as FDDI or ATM could be substituted with no change to the applications.

5.1 The Hardware-Replaceable Units

By a *replaceable hardware unit* we mean a physical unit of failure, replacement, and growth, i.e., a unit which fails independently of other units can be removed without affecting physically the behavior of other units and can be added to a system to augment its performance, capacity, or availability. The different processor types have a considerable variety of physical resources attached to them such as disks, tapes, printers, console displays, keyboards, radar adapters, external WWV time radio receivers, and specialized I/O adapters. For a small processor, a failure of the physical access path to such a resource or the failure of the resource itself results in an entire processor failure. For example, the failure of the display attached to a console processor results in a failure of the entire console processor. Thus, an entire small processor with all its computing, storage, user interface, and communication resources is a replaceable unit. For central server processors, concurrent maintenance of certain attached resources is possible while the processor continues to function perhaps with degraded capacity. In this case the hardware unit of failure/replacement is finer than an entire processor. For the LCN component, the atomic units of failure/replacement are the cables, the bridges between token rings, and the ring adapters used to attach processors to access rings.

5.2 Ensuring that the Hardware-Replaceable Units Have Crash/Omission Failure Semantics

The extensive error-detecting/correcting mechanisms embodied in the chosen processors, in the IBM 802.5 token ring and bridges, justify to a high degree—although not entirely—our goal of providing hardware with crash or omission failure semantics to applications. An implementation of raw processor services with crash failure semantics by using pairs of RS/6000 processors which execute in lock-step and systematically compare their results, as described in Taylor and Wilson [1989], was rejected as being too expensive and redundant with the run-time checking that the error-detecting/correcting codes used in these processors do anyway. The use of processors from other vendors which implement crash failure semantics by lock-step duplication and comparison was also considered and rejected because of problems related to lack of sufficient computational capacity, lack of real-time executives designed for such processors, and cost reasons. Moreover, it was felt that processors were just one of a larger hardware chain that contains a variety of hardware servers, some of which use error-detecting codes (e.g. memories, busses, disks) and some of which do not (e.g. electronic displays, radars), and that using duplexing in just one link of the larger chain was not economically justifiable.

5.3 How Much Hardware Redundancy Is Needed for Achieving the Required Availability Goals?

Assuming independence of the hardware-replaceable units and the failure data accumulated from past field experience for the chosen hardware, exten-

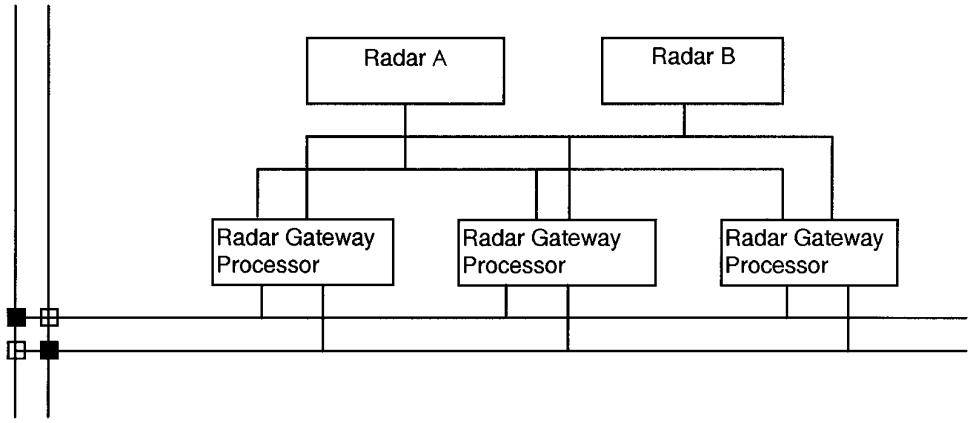

Fig. 5. Radar subsystem interconnections.

sive stochastic modeling studies were used to determine the number of hardware servers needed in the various subsystems to ensure that there exist enough working processors for each ATC service at any time and that timely communication is possible among these processors with very high probability. In order to predict *system* availability, these models take into account both hardware and software failure rates and mean times to repair. For software, the failure rates are estimated based on predicted number of faults left in the code at system deployment and the rate of execution of the software. The mean time to repair for software is equivalent to the software's coldstart time; consistent with our assumptions in Section 4.2, we assume there are not any permanent failures in a software module. This model can be used to determine the level of redundancy needed for specific system requirements; if two systems have a different availability requirement, they may differ in the amount of redundancy needed. Figure 4 illustrates the redundant LCN topology chosen following these studies: a set of two physical backbone token rings, represented vertically, is used to interconnect several horizontal access rings, consisting themselves of up to two physical rings. The number of token rings per ring set can be increased as traffic volume or traffic separation requirements dictate.

Figure 5 illustrates the hardware redundancy that could be employed in a radar gateway subsystem: each radar connected to three different radar gateway processors forming a radar gateway processor group. (This level of redundancy would be required for radars, since the service provided by the radar gateway processors is involved in the critical function of providing air traffic controllers with a current picture of his or her airspace.) In this way the group can mask the failure of any two of its members and still provide the hardware resources necessary for running the Surveillance Processing application. Redundancy is employed in a similar way in the Communication Gateway, Monitor and Control, and Central Server subsystems, by grouping the processors in these subsystems into *redundant processor groups*.

Hardware redundancy is used in a similar way to ensure the availability of processor services needed for running the ATC services responsible for monitoring a sector of airspace: from one to three console processors (with their attached displays and keyboards) can be grouped together into a *sector suite* processor group. The configuration choice is made by the user based on human factor considerations for the management of the air sector and a plan for redundancy. When a console is lost, the display real estate, and the associated processing workload, is redistributed among the surviving console processors. If the number of working console processors in a sector suite group drops below a user-specified threshold, the set of ATC services controlling that air sector can be *moved* to a spare sector suite processor group. This *mobility* of air sector monitor/control services contrasts with the immobility of all the other ATC services, which are bound to their original host processor groups, the only ones with physical access to the resources needed for providing these services. For example, the Surveillance Processing service responsible for receiving radar data from a given radar and associating tracks with that data is *bound* to the processor group whose members are physically attached to the monitored radar.

6. WHAT AVAILABILITY POLICIES SHOULD BE DEFINED FOR THE VARIOUS APPLICATION SERVICES?

The availability policy for a certain server group prescribes how closely the local states of group members should be synchronized and how many members should the group have, so as to meet the specified response time and service availability requirements.

6.1 Application Server Synchronization

For any service, the *server group synchronization* policy prescribes the degree of local state synchronization that must exist between the servers implementing the service. There are two extreme synchronization policies with a continuum of mixed policies in between. Stated simply (see Cristian [1991]), closely synchronized group members all process the same input against the same state, and therefore every correct member changes state and produces correct outputs. With loosely synchronized group members, one member processes the input against the group's state and forwards the resulting state updates to the other members.

Some ATC services are implemented by closely synchronized server groups, while others are implemented by loosely synchronized groups. The system design must specify the type of synchronization used in each case, based on performance and availability requirements for the ATC services. Each choice will affect network loadings, processor loadings, and recovery times. In general, close synchronization involves less communication overhead, but since all servers in the group consume the same input, it provides more opportunity for concurrent failures of all servers in the group.

For example, the Surveillance Processing service for a radar is implemented by three closely synchronized servers that receive input in parallel

from the radar and independently associate aircraft tracks with the radar data. The incoming data rate from any single radar is relatively high (several hundreds of target reports/second). To minimize multicast traffic, only the *highest-ranking* server multicasts the current position of aircraft to console processors. (The group availability management service (see Section 8.1) provides each server with a unique rank.) The Flight Planning service hosted by the central server subsystem is an example of a loosely synchronized service, implemented by a group of two servers. This arrangement reflects the fact that response time requirements for this service are less stringent and that the incoming data rate is much less than for the Surveillance Processing service.

6.2 Choosing the Number of Redundant Servers for an Application

For any given service, the *replication policy* prescribes how many redundant servers should ideally be used to implement the service. This is determined based on the hardware and software MTBFs and MTTRs. Based on hardware alone, a replication policy of more than 2 is rarely needed. When adding the effect of software this may grow to 3 for the most stringent situations. Additional factors may call for more replication than is strictly required—for example, if the optimal view layout for a single controller requires three display surfaces. The synchronization and replication policies selected for a service constitute the *availability policy* for that service. The Surveillance Processing service is implemented by a group of three closely synchronized redundant servers; in this way the service of periodic aircraft track data broadcasting remains available to ATC services running in console processors despite any two concurrent server failures. Most other ATC services are implemented by primary/standby groups (e.g., Flight Plan Processing, Tactical Conflict Detection/Resolution, Weather Processing). Some services are required to have servers on any correctly functioning host in a processor group. For example the Display Management service is required to have closely synchronized servers on all working console processors of a sector suite group.

7. HIGHLY AVAILABLE COMMUNICATION SERVICES PROVIDED TO APPLICATION SERVER GROUPS

In order to utilize the redundant communication paths, one first needs to find out the current *topology* of correctly working LCN components. Once the topology of active components is known, point-to-point and multicast *routes* that avoid failed LCN components can be computed. Second, if servers for a service can move from one processor to another, another important issue is that of *naming*: how do users address their service requests to server groups? Third, assuming that users are able to address service requests to server groups, one also needs protocols for controlling the *sequencing and delivery* of messages between client groups and server groups.

7.1 Maintenance of the Communication Network Topology

A *network topology management* service depends on LCN-wide connectioned as well as connectionless point-to-point message communication and

multicast services (Figure 2) to maintain up-to-date knowledge of the physical topology of the LCN network at any point in time. The topology consists of the following basic elements: processors, processor-to-ring adapters, rings, and bridges. The LCN-wide point-to-point and multicast message services depend in their turn on the Media Access Layer services defined for each individual ring. The topology management service is implemented by a group of up to three servers, which maintain redundantly three copies of the LCN topology image. The leader of this group is chosen in the same way as all group leaders: by the group availability management service. The topology database can change dynamically as processor, adapters, bridges, and rings join or leave the active LCN topology (because of failures or maintenance actions). Events which affect the LCN topology are detected either by servers outside the topology manager group (e.g., bridges detect adapter insert/deinsert events on the rings to which they are attached) or by the leader of the topology management group, which periodically tests via special "circuit test" messages components such as bridges and rings. Topology updates detected by the leader are reliably checkpointed to all its active peers.

7.2 Ensuring the Availability of Communication Routes

To enable quick masking of a message loss due to the failure of a route between two processors, the decision was taken that two independent routes should be maintained between any two processors that need to communicate by a *route management service*. In this way, if the failure of a route to a target processor is detected by a sender processor (by the absence of timely acknowledgments for messages sent along that route), the sender processor can use the alternate route to resend the unacknowledged messages to the target processor. If the route failure is diagnosed as being permanent by the topology management service, the route management service computes and sends an alternate route to the sender processor so as to maintain two routes between any two processors which need to communicate. The redundancy inherent in the LCN topology allows the routes (i.e., sequences of adapter and bridge addresses) between any two processors to be physically disjoint, so that no single hardware failure affects both routes simultaneously. The route management service is implemented by a group of up to three route servers. Communication costs between topology management servers and route management servers are minimized by letting the latter run in the same address spaces as the former. The route managers depend on the topology service to compute a pair of routes whenever they are requested by a processor which wants to establish a session with another processor (Figure 2). Such sessions are used to multiplex several higher-level sessions between processes running on the two processors. To avoid sending unnecessary requests to the route management servers whenever new process-to-process sessions must be open, processors cache such routing information locally. The topology service informs the route management service about any topology changes, so that

new routes which avoid failed components or take advantage of repaired components can be computed and propagated to all affected processors. Inconsistencies between the locally cached routing information and the route servers are resolved in the usual way: when a processor detects that its cached information is out-of-date, either because it has missed a route update message from the route servers or because it detects a route failure, it requests new routes from the route server group.

Dual routes are not only maintained for point-to-point communication, but also for multicast communication. Although the token rings allow all-rings multicast in addition to routed multicast, the former form of multicast is not used for bandwidth reasons. To ensure that the periodic multicast communication pattern of Figure 3 (implemented by using the token ring functionally addressed multicast service and the dual multicast routing service provided by the route managers) is available despite component failures, a reliable periodic multicast protocol based on the use of negative acknowledgments was designed. This protocol relies on the presence of message traffic in the normal case and does not have any message overhead in the absence of failures.

7.3 Communication Services Provided for Application Server Groups

In addition to the one-to-many multicast services mentioned in the previous section, another important class of communication services is provided to enable client groups to request services to server groups and match replies with requests. These request/reply services among process groups depend on lower-level process-to-process session services, which in their turn depend on location-transparent naming and processor-to-processor session management services (Figure 2).

A *processor-to-processor session management* service uses dual routes as mentioned before to guarantee sequenced and reliable delivery of messages. A server for the distributed processor-to-processor session management service exists on each processor. A *naming service* makes use of the routed multicast service (Figure 2) to record the current host processor for the highest-ranking server of each ATC server group into a table which maps service names to processors. Fragments of this names mapping are cached in each processor, so that these need to access the name service only at initialization or when detecting cache inconsistencies. The naming service is implemented by up to three replicated *name servers*, colocated for efficiency in the same address spaces as the topology and routing manager servers mentioned previously.

The *process-to-process session managers* depend on the naming (Figure 2) and the processor-to-processor session services to provide *location-transparent sessions* for ATC applications in the following way. When a server s for an ATC service S on processor p becomes the highest-ranked member of the group which implements S, the group availability manager informs the name service of the fact that service requests for S must now be addressed to p. To any subsequent request that a server t running on a processor q

makes for opening a session with S, the name server leader responds by giving the process-to-process session manager on q two low-level routes (i.e., sequence of adapter and bridge addresses) between p and q. All service S requests are routed by this session manager to p without t knowing anything about where s resides. If s fails, and a new server s' for S on another processor p' becomes the highest-ranking S member, the group availability manager on p' will inform the name manager leader that it now services requests for service S. The name manager uses an unacknowledged inexpensive multicast to inform all session managers which had sessions with s that now they should use two new routes for sending service S requests to s'. If each session manager that maintains sessions with S stores all S requests sent and not yet acknowledged, these can be resent by using the new route. In this way, the crash of the highest-ranking server s for S and the fact that service S requests are subsequently being processed by an alternate server s' for S remain transparent to all S users. The clients experience the failure as a mere delay in getting their answers. The time between the detection of the failure of s and the moment communication is reestablished with an alternate server s' will be minimized if the ATC service is implemented by groups of at least two active servers. This avoids a start of s' after the failure of s, which would delay the replies to service S users. ATC services are generally implemented by groups of two servers, allowing the delay in responses after server failures to be in the range of 0.5 to 1.5 seconds, as dictated by ATC systems response time requirements.

The point-to-point location-transparent process-to-process sessions services are in turn used to implement application-specific request/reply protocols among ATC application groups. These protocols provide an "at-least-once" semantics (when the operations exported by servers are idempotent) in order to minimize the bookkeeping necessary during steady-state processing. Some send messages in parallel to all members of a group, some only to the highest-ranked members. This system-wide decision is made based on LCN and processor performance considerations and is coordinated with the synchronization policies of the receivers.

8. ENFORCING THE SPECIFIED AVAILABILITY POLICIES FOR APPLICATION SERVICES

The mechanisms which ensure internal synchronization of the local member states in a server group depend on the synchronization policy prescribed for the service implemented by the group. If group members are required to be closely synchronized, it is sufficient that they receive the same sequence of inputs to stay synchronized, and a member need communicate with other members only when joining to get the group state [Cristian 1988]. If group members are loosely synchronized, the checkpointing which goes on among members at join as well as periodically is very much application dependent and is left to the service implementors (common synchronization services are provided to applications so that they can

efficiently implement their chosen technique). In this article we focus on the complementary issue of how to ensure that, for all services of a service group, there are enough members with the right rankings (if applicable) despite failures and joins.

To force a clear separation between the notion of a service availability policy and the mechanism used for automatic policy enforcement, a unique *group service availability management* service (gSAM) was implemented. This requires ATC servers to implement only the application-specific get-state and checkpoint procedures needed for local state synchronization at join and checkpoint time. The gSAM service learns of the availability policies intended for the ATC services in its service group through a set of startup parameters. The gSAM service then handles the tasks of detecting server failures, coordinating promotions, and enforcing replication policies automatically, without need for human intervention.

8.1 The Group Service Availability Manager

A group service availability management service must ensure that, for each service in the group, the prescribed availability policy is enforced automatically despite processor and server departures (due to failures and scheduled shutdowns) and processor and server joins (due to recovery from failures, changes, or growth). To achieve this objective, the gSAM servers must agree on a processor group state despite random communication delays and failures. The group state consists of the set of group processors that are correctly working, called the *processor group membership* [Cristian 1988], as well as of several mappings (or tables) recording what availability policies have been defined for the various services hosted by the group, what servers run on what processors, and what rankings exists among the members of the running server groups. The problem of building a generic gSAM service for any processor group was broken down into three subproblems: first, build a *processor group membership service* which ensures agreement on processor group membership, second, build an *atomic broadcast service* which ensures that all updates to the state of the mappings mentioned above are applied by all gSAM servers in the same order, and third, *synchronize* the states of the gSAM servers so as to ensure that they react as promptly as possible to operator commands and failures.

8.1.1 *Reaching Agreement on Processor Group Membership.* The ATC infrastructure implements synchronous group membership protocols as described in Cristian [1988]. These protocols ensure a number of *safety* properties, such as "at any time, all members of a group agree on the group membership," and *timeliness* properties, such as "there is an upper bound on the time it takes to detect group member failures or joins." Synchronous processor group membership protocols satisfying such safety and timeliness requirements depend critically on whether communication partition failures, which prevent group members from communicating in a timely manner, can occur or not. The ATC infrastructure design minimizes partition failures by using real-time operating systems capable of guaranteeing

bounds on message delays under worst-case system load conditions and by sending messages along multiple physically independent paths to mask individual path failures. In spite of these steps, network partitions may rarely occur. In such cases, the partitions are recognized (the clock synchronization protocol can recognize when it does not find an existing time source), and manual intervention is then required to bring the partitioned resources back into the operational system. The synchronous approach [Cristian 1990] was chosen based on the requirement of providing bounded failure detection delays and by the availability of a real-time version of the AIX operating system that promises to bound message transmission delays on correctly functioning processors and token rings.

The speed at which these protocols detect member failure is inversely proportional to the communication traffic generated by the protocol, and therefore the network performance and recovery time requirements must be balanced. Typical ATC implementations detect member departs in less than one second.

8.1.2 *Reaching Agreement on a Processor Group State.* Since the operations that affect the global state of a gSAM group are in general not commutative, gSAM servers use a *group atomic broadcast* communication service to broadcast global group state updates among themselves. The ATC infrastructure uses a synchronous group atomic broadcast protocol which ensures that (1) each message whose broadcast is attempted by a group member is either accepted by all correct members or by none of them (atomicity), (2) all broadcasts accepted are accepted in the same order by all correct group members (order), and (3) there is an upper bound on the time necessary for a correct group member to broadcast a message to all other correct group members (termination) [Cristian 1989a]. The protocols are optimal in their failure-free message cost: to tolerate up to f concurrent component failures, they use $f + 1$ messages to broadcast a group state update atomically, independently of the size of the group participating in broadcast.

A clock synchronization service (Figure 2), which synchronizes clocks to external time signals received by radio, is used by both the synchronous atomic broadcast service and some ATC applications. This service is implemented by following a probabilistic approach to synchronizing clocks [Cristian 1989b]. This approach can achieve with very high probability constant maximum deviations from external time of less than 10 milliseconds despite the fact that the uncertainty in LCN message delays (due to transient conditions on the token ring and scheduling delays within the processor) is higher than 100 milliseconds.

8.1.3 *Synchronizing the Local States of Group Service Availability Managers.* To ensure that the specified availability policies are enforced for all services hosted by a processor group for all possible group memberships, gSAM servers are replicated on all group members. Following Cristian [1985], the gSAM service is viewed as being an interpreter for two classes of concurrent events: human commands input by operators and component failure occurrences caused by the adverse group environment.

The service is fault-tolerant because it undergoes specified state transitions not only in response to human operator commands, such as join processor or join server, but also in response to any number of crash/performance server and processor failures. The failure of a server running on a processor is detected either by the operating system running on that processor or by the local gSAM server via periodic tests. In all cases the local gSAM server notifies the other gSAM servers about the failure (which changes the group state) via a group atomic broadcast. The failure of a processor is detected by the processor group membership service. The failure notification sent by this service to the surviving gSAM servers looks to them just the same as a group atomic broadcast. In this way all events affecting the group state are seen in the same order by all correct group members.

Previous experience with the design of a service availability management service within the Highly Available Systems project at the Almaden Research Center indicated that close synchronization among the servers implementing the service yields a simpler overall design than loose synchronization, with no loss of performance. The reasons for this are that in a closely synchronized group there is no need to program the (often complicated) promotion protocols that must be executed when a server becomes the highest-ranking member after the previous leader fails. Moreover, since in a loosely synchronized approach it is the leader's role to synchronize all concurrent events affecting the group, special point-to-point communication mechanisms are needed to enable ordinary group members to inform the leader about all events that they detect. This not only leads to more complicated code, but also to longer delays in reconfiguring the group after component failures.

In view of the above experience, the decision was taken to use close synchronization in the server groups that implement the gSAM service for all processor groups. To give an idea of how a service availability management server group works, we illustrate below how a sector suite gSAM server group reacts to a server or processor failure. Consider a sector suite group composed of three console processors p, q, r, with servers for the Aircraft Track Management (ATM) service running on all three processors. These servers are ranked: the highest ranked is on processor p, the next on q, and the last on r. If the group service availability manager on p detects the failure of the local ATM (primary) server, it atomically broadcasts the news about the crash to all gSAM group members. The algorithm executed by an arbitrary group member x when it receives the broadcast is the following: if there is a local ATM server, and its rank is lower than the rank of the failed server, x promotes its local ATM server to the next higher rank; otherwise, if there is no local ATM server, then x starts one with the lowest possible ranking. In this way the ATM server on q becomes the highest-ranking server in the ATM group; the one on r becomes the next; the server on p becomes the last in the ranking group; and the handling of the original server failure on p is considered terminated. A processor failure is handled in a similar manner by being interpreted at each surviving processor as being the failure of all ATC servers that were running on that processor.

9. MASKING FAILURES OF A MOVABLE SERVICE GROUP

For nonmovable service groups (such as those bound to radar, communication, or central server groups) the failure of all servers in the group implementing the service results in a service failure. This can in its turn result in a service group failure or at least a degradation of the level of service provided by that group. This need not be the case if an air sector surveillance service group hosted by a console processor group fails. Indeed, as mentioned earlier, this service group is movable from one console processor group to another one, since each console processor group has all the physical resources needed for running the surveillance services for *any* air sector. In the ATC infrastructure there is a special service responsible for ensuring the availability of movable ATC service groups: the *Global Availability Management Service* (GSAM). Besides this role, the GSAM service also plays the important role of *interface* between an ATC system and its human operators and administrators. To this end it must collect status data about the various hardware and software system components on a periodic basis to be able to display it to operators on demand. The GSAM service also transmits operator commands to processor groups. For example a command such as "enable the surveillance-processing service for a certain radar R to be provided" is entered by an operator from a keyboard attached to a GSAM server, and it is the GSAM server which monitors whether the command is successfully executed by the gSAM servers in the radar gateway processor group with access to radar R.

9.1 The Global Availability Management Service

Like the gSAM service, the GSAM service is implemented by a closely synchronized group of up to three servers. The group keeps track of the status of all ATC service groups as well as of a pool of spare sector suite processor groups defined for each facility. The highest-ranked gSAM servers for all processor groups of a facility report periodically to all GSAM servers the status of the processor group to which they belong by using a routed multicast communication service. The periodicity of status reporting makes the GSAM group *self-stabilizing*: the group state (which should reflect the current facility state, including load) is completely defined when the "last" broadcasts from all processor groups are received in a timely manner. If d is the upper bound on the time needed by a group status report multicast to reach all GSAM group members in the absence of failures, and the multicasts occur every p time units, then in the absence of failures the local state of a GSAM server is up-to-date $p + d$ time units after it starts. A processor group failure can be detected if a GSAM server does not receive a status report multicast from that group within $p + d$ time units from the last broadcast. Given the massive redundancy used in the LCN such events should be very rare. If the group has crashed and is movable, it will be moved by the GSAM servers to a spare sector suite. If the group is not movable, but the clients of the group can continue to function in a degraded service mode without the group's services, the GSAM servers will wait until the group comes back again. If the service

group is essential for its clients then a switch to an alternate communication mechanism described below can occur.

10. WHAT IF COMPONENT FAILURES OUTSIDE THE CONSIDERED FAILURE HYPOTHESES OCCUR?

The radars can be connected to console displays not only by the LCN network described previously, but also by a backup local area network Ethernet, with enough capacity to transport essential, periodically generated radar data to all consoles. If an air traffic controller detects that the main ATC system experiences what to him or her looks like a failure, he or she can unilaterally switch to receive radar data about the aircraft in his or her sector from the backup network. The backup data is rather low level and only enables a controller to continue a job under an "emergency" work mode, until the normal, higher-level service through the main ATC system is restored. Very few hardware and software components are shared between the primary ATC radar data path and the backup Ethernet radar data path, to ensure a high degree of physical independence and design diversity between the two paths. The expectation is that a high degree of physical independence and design diversity will cause the ATC primary radar distribution path and the Ethernet backup radar distribution path to fail independently.

Having an independent backup system that provides enough functionality to be used for short periods of time also provides a means to gracefully introduce changes (hardware or software) to the primary system. Operations can temporarily be carried out on the backup system while the primary system is upgraded and then restarted.

11. ATC SYSTEMS USING THIS INFRASTRUCTURE

Three en route systems currently use this infrastructure. Taiwan's ATCAS has been in operation since the spring of 1996. The US DSR system is planned for delivery to the FAA in spring of 1997. The UK NERC system is planned for delivery to the CAA in the summer of 1997. All the services shown in Figure 2 except "movable highly available ATC applications" are used by all three systems. (Only the NERC design calls for a movable application. In that system design, the applications running in a group of workstations supporting a single sector can be moved to an unused group in the event of group failure.)

The systems vary by the set of applications and by the system size. Using Figure 1 to define the generic set of air traffic control applications, ATCAS implements all but the communications gateway subsystem. ATCAS is a fairly small system, deployed in sites with from 10 to 20 nodes. The DSR system, to be deployed at all 20 US en route centers, uses existing FAA systems for the radar gateway subsystem and the communications gateway subsystem and will range in size from 100 to 150 nodes. The NERC system includes all subsystems defined in Figure 1 and will be deployed in a configuration of approximately 200 nodes. The DSR system is the only one of these en route implementations to contain the backup LAN Ethernet as discussed in Section 10.

12. OTHER QUESTIONS AND ISSUES

Although the work described in this article was done in connection with the design of a particular class of fault-tolerant distributed air traffic control systems, we believe that many of the system-structuring concepts used are quite general. For example, the notions of service, server, and "uses" relation, the nested failure classification and the notion of server failure semantics, the methods of masking server failures through server groups, and the associated redundancy management and group communication mechanisms provide powerful conceptual tools which can be used in other fault-tolerant system designs as well. We believe in particular that any team which designs a distributed system with high-availability objectives has to confront the same design choices as mentioned in this article. Thus, this article can be useful as a guide through the labyrinth of fault-tolerant system design.

A number of important questions outside the scope of this article must be addressed when designing fault-tolerant distributed systems. What is the failure semantics specified for the various application services, and what methods are used to implement this semantics? How have the applications been structured hierarchically, and how is exception handling and crash recovery performed in this hierarchy? How is checkpointing being implemented in ranked groups, and what specific communication protocols are used for communication among application groups? How will a very quick recovery after a total system failure be possible? How will we accurately predict and measure the availability of application services? With a robust infrastructure in place, these higher-level questions can then be answered.

REFERENCES

AVIZIENIS, A. AND BALL, D. 1987. On the achievement of a highly dependable and fault-tolerant air traffic control system. *IEEE Comput. 20*, 2 (Feb.).

CRISTIAN, F. 1985. A rigorous approach to fault-tolerant programming. *IEEE Trans. Softw. Eng. SE-11,* (Jan.).

CRISTIAN, F. 1988. Reaching agreement on processor group membership in synchronous distributed systems. *Distrib. Comput. 4,* 4.

CRISTIAN, F. 1989a. Atomic broadcast for redundant broadcast channels. *J. Real-Time Syst. 2,* 195–212.

CRISTIAN, F. 1989b. Probabilistic clock synchronization. *Distrib. Comput. 3,* 3, 146–158.

CRISTIAN, F. 1989c. Exception handling. In *Dependability of Resilient Computer,* T. Anderson, Ed. Blackwell Scientific, Oxford, U.K.

CRISTIAN, F. 1990. Synchronous atomic broadcast for redundant broadcast channels. *J. Real-Time Syst. 2,* 195–212.

CRISTIAN, F. 1991. Understanding fault-tolerant distributed systems. *Commun. ACM 34,* 2 (Feb.).

DEBELACK, A. S., DEHN, J. D., MUCHINSKY, L. L., AND SMITH, D. M. 1995. Next generation air traffic control automation. *IBM Syst. J. 24,* 1, 63–77.

GRAY, J. 1986. Why do computers stop and what can be done about it? In the *5th Symposium on Reliability in Distributed Software and Database Systems.*

TAYLOR, D. AND WILSON, G. 1989. The Stratus system architecture. In *Dependability of Resilient Computer,* T. Anderson, Ed. Blackwell Scientific Publications, Oxford, U.K.

Received January 1991; revised January 1996; accepted April 1996