# Report on the
# Fourth ACM SIGOPS European Workshop
## Fault Tolerance Support in Distributed Systems

*Özalp Babaoğlu*

Department of Mathematics
University of Bologna
Piazza Porta S. Donato, 5
40127 Bologna, Italy
`ozalp@dm.unibo.it`

## Introduction

Continuing the tradition of Zürich (1984), Amsterdam (1986) and Cambridge (1988), the 1990 edition of the ACM SIGOPS European Workshop was held in Bologna, Italy from 3 to 5 September.

As the theme of the workshop, "fault-tolerance support in distributed systems" was selected. The motivation for this choice was the observation that distribution and fault tolerance can be argued to be two faces of the same medallion—any system that can tolerate failures must be distributed, and any system that is distributed must be capable of surviving partial failures. To provoke further investigation of this duality, the Call for Participation included the following questions:

- How reliable should distributed systems be for applications like telephone switching, funds transfer, air-traffic control, etc.?
- What formal methods are appropriate for specifying the fault tolerance requirements for applications?
- What are appropriate models (transactions, process groups, client/server, etc.) for fault-tolerant computations in a distributed system?
- What are appropriate paradigms for supporting fault-tolerant applications and how can they be implemented efficiently?
- How should fault tolerance primitives be packaged? As new language constructs, extensions, libraries, etc.?
- To what extent can fault tolerance be retrofitted into existing applications automatically?
- How do hard real-time and fault-tolerance requirements interact or conflict in a distributed system?
- What lessons can be learned from existing implementations of fault-tolerant and distributed systems?

Based on the 60 position papers that were received, the Program Committee consisting of Özalp Babaoğlu, Jim Gray, Hermann Kopetz, Roger Needham, Fabio Panzieri, Fred B. Schneider and Marc Shapiro selected 54 authors to be invited to the workshop. The actual attendance (70 persons; 26 from North America, 44 from Europe; total of 12 countries) was slightly higher than usual due to the high acceptance rate and the fact that several position papers had multiple authors present.

The Program Committee asked 25 authors to make presentations to start off discussion in 9 sessions. The format also included a panel discussion and an afternoon session were 12 participants made short presentations. Reports of the sessions that follow have been prepared mostly by the various Session Chairs with valuable assistance from scribes Aleta Ricciardi, Amitabh Shah and Pat Stephenson. In addition to this report, SIGOPS has graciously agreed to have the OSR publish, in several installments, all of the position papers that have been presented at the Workshop.

The meetings were held in Palazzo Malvezzi (the 16th century residence of the family Malvezzi; now housing the Law School) of the University. I am most grateful to Hewlett-Packard Europe, the University of Bologna, the National Research Council of Italy (CNR) and AICA for their gracious support of the Workshop and to Italiana & Co. for the flawless local organization.

## List of Participants

Emmanuelle Anceaume, INRIA, France
Özalp Babaoğlu, University of Bologna, Italy
David F. Bacon, IBM Thomas J. Watson Research Center, U.S.A.
Mary Baker, University of California, Berkeley, U.S.A.
Rodric Bark, Hewlett-Packard Laboratories, United Kingdom
Thomas Becker, University of Kaiserslautern, Germany
Yolande Berbers, K. U. Leuven, Belgium
Andrew Birrell, Digital Equipment Corporation, U.S.A.
Andrew Black, DEC Cambridge Research Lab, U.S.A.
Guillaume Brès, Laboratories de Marcoussis – CGE, France
Richard Carr, Tandem Computers, Inc., U.S.A.
Liam Casey, Bell-Northern Research Ltd., Canada
Flaviu Cristian, IBM Almaden Research Center, U.S.A.
Jos Coenen, Eindhoven University of Technology, The Netherlands
Robert Cooper, Cornell University, U.S.A.
George Coulouris, Queen Mary and Westfield College, United Kingdom
Dean Daniels, IBM Almaden Research Center, U.S.A.
Jean Dollimore, Queen Mary and Westfield College, United Kingdom
Nigel Edwards, Hewlett-Packard Laboratories, United Kingdom
Jean-Luc Gaudiot, University of Southern California, U.S.A.
Vittoria Gianuzzi, Università di Genova, Italy
Michel Gien, Chorus Systèmes, France
Gita Gopal, Bellcore, U.S.A.
James Gray, Tandem Computers, Inc., U.S.A.
Frédéric Hermann, Chorus Systèmes, France

David B. Johnson, Rice University, U.S.A.
M. Frans Kaashoek, Vrije Universiteit, The Netherlands
James Jay Kistler, Carnegie Mellon University, U.S.A.
Hermann Kopetz, Technical University of Vienna, Austria
Winfried Kühnhauser, GMD, Germany
Rivka Ladin, DEC Cambridge Research Lab, U.S.A.
Spiros Lalis, ETH Zentrum, Switzerland
Gérard Le Lann, INRIA, France
Letizia Leonardi, Università di Bologna, Italy
Barbara Liskov, Massachusetts Institute of Technology, U.S.A.
Mirek Malek, Office of Naval Research, United Kingdom
Edoardo Marelli, European Space Agency, The Netherlands
Lindsay F. Marshall, University of Newcastle upon Tyne, United Kingdom
Keith Marzullo, Cornell University, U.S.A.
Pascale Minet, INRIA, France
Christine Morin, IRISA, France
Sape Mullender, Centre for Mathematics and Computer Science, The Netherlands
Walid Najjar, Colorado State University, U.S.A.
Juergen Nehmer, University of Kaiserslautern, Germany
Edgar Nett, GMD, Germany
Fabio Panzieri, Università di L'Aquila, Italy
David Powell, LAAS-CNRS, France
Xavier Rousset de Pina, Bull-IMAG, France
Aleta M. Ricciardi, Cornell University, U.S.A.
Jerome H. Saltzer, Massachusetts Institute of Technology, U.S.A.
Henk Schepers, Eindhoven University of Technology, The Netherlands
Richard D. Schlichting, University of Arizona, U.S.A.
Fred Schneider, Cornell University, U.S.A
Hermann Schmutz, IBM European Networking Center, Germany
Ralf Schumann, GMD, Germany
Amitabh Shah, Cornell University, U.S.A.
Marc Shapiro, INRIA, France
Santosh K. Shrivastava, University of Newcastle upon Tyne, United Kingdom
Luca Simoncini, Università di Reggio Calabria, Italy
Carla Simone, Università degli Studi di Milano, Italy
Pat Stephenson, Cornell University, U.S.A.
Lorenzo Strigini, I.E.I. – C.N.R., Italy
Andrew S. Tanenbaum, Vrije Universiteit, The Netherlands
Brendan Tangney, Trinity College, Ireland
Arjen Uittenbogaard, HCS Industrial Automation BV, The Netherlands
Tullio Vardanega, Intecs Sistemi, Italy
Paulo Veríssimo, INESC, Portugal
John Warne, STL/ISA, United Kingdom
William Weihl, Massachusetts Institute of Technology, U.S.A.
John Wilkes, Hewlett-Packard Laboratories, U.S.A.

## Session 1: Paradigms for fault tolerance support

**Chair and Rapporteur:** Fred B. Schneider

*Making Real-Time Reactive Systems Reliable* by Keith Marzullo and Mark Wood (Cornell University). Talk presented by Marzullo.

The thesis of this paper is that an architecture suitable for the design of real-time reactive systems can also be appropriate for other distributed application management problems. In both environments, sensors and actuators are convenient abstractions for controlling the environment, be that environment a process control system (with real hardware sensors and actuators) or some system software in a distributed system. The Meta system, which runs on top of ISIS, implements a data model for interacting with sensors and actuators and, therefore, supports the development of real-time reactive systems. The software includes a fault-tolerant averaging scheme, based on a simple but reliable approximate agreement protocol, so that faulty sensors can be tolerated. Meta also supports a data language, called Lomita, to allow interaction with a temporal object-oriented database that saves sensor data. The monitoring of rather complex "watch" conditions can be specified automatically by writing formulas in Lomita. Lomita will monitor for such conditions (in a fault-tolerant manner, by using replication) and guarantee that exactly one version of the rule will react when a condition is satisfied.

*Lazy Replication: Exploiting the Semantics of Distributed Services* by Rivka Ladin (DEC/CRL), Barbara Liskov, and Liuba Shira (MIT). Talk presented by Ladin.

Most replica management schemes require that all replicas of a service process requests in the same order. This has costly implementation consequences and is not always necessary for correct operation. Lazy Replication allows reordering of client requests based on the semantics of the service being replicated. As a result, different replicas process client requests in different orders, but the resulting service still appears indistinguishable to the client from a centralized service. With Lazy Replication, ordering constraints on the operations are defined when the service is instantiated, based on the semantics of the application. The designer of the replicated service can, therefore, ignore complications associated with replica management. Lazy Replication can provide good response time and high availability in the presense of node and communication failures. The implementation has a low overhead in stored state and message exchanges. The talk described a protocol to preserve causality. This protocol is based on a vector-timestamp scheme to keep track of potential causality among client requests. The scheme ensures that operation re-orderings occur (only) when potential causality (i.e. Lamport's "happens before" relation) is not violated. The talk also discussed some preliminary performance data for a prototype implementation. Surprisingly, the data indicate that the replicated system has a larger operation processing capacity than a non-replicated system that supported comparable response time.

Excerpts from the discussion following the presentations:

Verissimo: ISIS broadcasts are not real-time, correct?

Marzullo: Yes.

Shrivastava: Do you support time-outs?

Marzullo: Yes, through timers I can specify time.

Shrivastava: Can the timers be replicated?

Marzullo: We have not yet addressed this.

Le Lann: Elaborate on the ISIS-like broadcasts that you use in Meta.

Marzullo: We really need a real-time broadcast and they are not yet supported.

Le Lann: How can you distinguish between loose and tight timing constraints?

Marzullo: This requires a long answer and I would rather not get into it now.

Malek: How do you evaluate improvement in availability?

Ladin: Client-ordered updates are available as long as there is one available copy. However, there may be a response-time degradation for queries.

Verissimo: How do you decide how many copies to maintain?

Ladin: We haven't done any measurements to address this issue. It will depend on the availability requirements of each particular service and the cost in reduced performance that the application is willing to tolerate.

Malek: Does anyone have such measurements?

Black: There are two aspects to this question. First, the number of copies; second, how fast do you push out updates.

Liskov: Malek's question is fundamental.

Shrivastava: Do you allow servers to call others?

Ladin: Yes.

Shrivastava: So, it's replicated clients calling replicated servers?

Ladin: Yes.

Schneider: Clearly, there is availability only until too many failures have occurred. In light of this, I have two questions: 1. How do you bring a new replica on-line? 2. Can this be done without incurring some sort of blocking and performance degradation?

Marzullo: We follow the ISIS way. One way is to start a new process and do a state transfer. Note, this transfer would be asynchronous.

Cooper: We do not assume anything about state transfer. ISIS provides resiliency during state transfer – hence blocking is necessary, especially for transferring large amounts of information.

Ladin: Unlike other systems, a failure of a replica does not necessarily require a view change and therefore having a failed replica rejoin the group can be done painlessly by using gossip messages to bring it up-to speed. However, since some replicas fail "forever" and thus new ones must be added, we provide a reconfiguration protocol. The protocol consists of two phases: first, the coordinator of the protocol (the initiating replica) proposes the new configuration and waits for approval by all members in the new group. In reply, a replica sends its vote and its current state. Second, after receiving replies from all members in the group, the coordinator commits the new configuration by sending commit messages that include the merged state to all members including the new one.

Casey: How do you manage labels?

Ladin: To correctly interpret timestamps (labels), each configuration has a version number and we associate the version number with each timestamp. Only timestamps that correspond to the same configuration can be compared. Timestamps that span configurations are handled specially.

## Session 2: Fault tolerance through/despite file systems

**Chair and Rapporteur:** Sape J. Mullender

*Specifying Data Availability in DataMesh* by John Wilkes and Raymie Stata (HP Laboratories). Talk presented by Wilkes.

While the performance of processors has always increased steadily and rapidly, the performance of disks has only increased in terms of capacity and reliability. Disk speeds have increased only very little. As a result, disk speed has become an important limiting factor in overall systems performance. The DataMesh project aims at using three techniques to close the disk/processor performance gap: Better usage of disks, multiple disk devices and *active devices*.

The model is basically to build a processor into every storage device, to connect the storage devices in a fast, dense mesh network and to use processor cycles to get to the data faster. With this model, the goals of DataMesh are to achieve very large storage structures (0.1 to 10 terabytes) which are still faster on average than local disk and highly functional: highly available, scaleable, structured data with client-specified properties.

To satisfy the I/O bandwidth of 50 workstations in the mid-1990s, one needs some 400 disks which, on average, with a MTBF of 150,000 hours, will cause a failure every two weeks. It thus is evident that much attention needs to be paid to fault tolerance.

The user will play an important role in specifying the degree of fault tolerance required. The system will then decide on ways to realize these wishes. If a user specifies, for example, that the maximum unavailable time must be less than 30 seconds, that performance during failure must be at least 40% of normal, that no data may be lost, and that disasters (loss from multiple simultaneous failures) may not happen more than once every hundred years, then the system might choose to use a disk array with parity (multiple disks with one disk used as a parity disk), because other solutions do not give the desired functionality.

Naturally, more reliability, better performance and higher availability all cost more. Money should be used as a deterrent for users to ask for a better quality of service than is actually needed.

*A Highly Available and Reliable Unix File System for use via NFS* by Barbara Liskov, Robert Gruber, Paul Johnson, and Liuba Shrira (MIT). Talk presented by Liskov.

The Programming Methodology Group at MIT is working on the implementation of a reliable file server accessible via NFS which will provide exactly the same semantics as normal NFS servers, only with much higher reliability and availability.

The server has three replicas, each with disks and an uninterruptible power supply. Each file is stored at two replicas. Each file operation (e.g., reads or writes) is carried out as a single transaction. The server can be used by normal NSF clients. The only modification required is that some code must exist in the clients to switch to another server when one crashes. This can be done by using a reliable multicast protocol instead of the RPC protocol currently used.

The replication technique uses a primary copy method which makes use of the simplifying circumstance that, although each operation must be carried out atomically, there is no need for combining several operations into one atomic action.

During normal operation, a file system is assigned to two servers, the primary and the backup. The primary fields client requests and communicates with the backup only for write operations.

The primary maintains a time-ordered log in primary memory for modifications. The log entries are periodically transmitted to the backup, which carries out the modifications and returns acknowledgements. The primary keeps a pointer to the last-acknowledged log entry. The primary acknowledges to the client after the arrival of the backup's acknowledgement. Log entries can be deleted when modifications have been written to both primary and backup disk.

If a server fails, or the network partitions, a *view change* occurs. At this point the third server becomes involved as a *witness*. The remaining server becomes the primary and the witness becomes the backup. Since the witness does not have a copy of the file system, it merely writes the log entries it receives to disk. In any case, the new view must contain a majority of the servers.

The implementation is done in C on BSD 4.3 Unix running on MicroVax 3's connected by Ethernet. A future implementation will run on DECstation 5000s.

After these two presentation, a discussion started, which mostly consisted of grilling the speakers.

Marshall: What is the probability of sudden change, i.e. severe disaster scenarios?

Wilkes: You will need replication of data to tolerate such failures, however small the probability.

Le Lann: What is the difference between max. down time and max. repair time?

Wilkes: Down time is failure detection time plus repair time. The user only gets to specify the max. down time.

Tanenbaum: NSF keeps the "time last read" for each file which essentially results in file modification on reads. The MIT server does not keep track of this time accurately. Where else do you cheat?

Liskov: Time-last-read and switching over to a new server on failure are the only two client-visible differences.

Marzullo: There are file systems other than NFS that provide location transparency, but not failure transparency. Do you?

Liskov: We have not looked at other file servers. NFS is well accepted, and hence was our choice. Our hypothesis is that it will not change very much with other file systems. The basic techniques remain the same.

Then a discussion started on the assumption of clock synchronization.

Cristian: The correctness depends on clock synchrony. How do you do that in Unix?

Liskov: We do have clock synchrony — accurate to less than 100 ms.

Cristian: However, there is no guarantee known a priori.

Liskov: Clock synchronization will become easier.

Birrell: You only need rate synchronization, not clock synchronization.

Liskov: Are they not the same?

Birrell: Absolutely not!

Le Lann: Birrell says that you only need synchronized rates, not clocks. Now since there are other problems like Atomic Broadcasts that use clock synchronization (to guarantee rates?), isn't this a tautology?

Schneider: You can implement one with the other. It is a question of methodology. Barbara gives one such protocol. Note that the Fischer, Lynch, Patterson result says that if there is no agreement on rates then there can not be any agreement on clocks.

Kopetz: Is it correct that the server only tolerates crash failures?

Liskov: The server certainly does not tolerate Byzantine failures. However, message losses are OK.

Gray: It is a common occurrence that logs get filled up. Do you really need to keep the logs? What about reading backups?"

Liskov: Andrew's Echo system does not keep logs. We feel that we need to keep the logs. However, I don't know what is the right thing to do. I have no idea about reading backups.

Saltzer: Do you still need to backup the system if it performs as well as you hope?

Liskov: Unfortunately backups will always be needed, even if it were only to safeguard against operator error.

## Session 3: Transactions

**Chair and Rapporteur:** Barbara Liskov

*Shared Logging Services for Fault-Tolerant Distributed Computing* by Dean Daniels, Roger Haskin, Jon Reinke and Wayne Sawdon (IBM Almaden Research Center). The talk, given by Dean Daniels, discussed a logging service for use within the Quicksilver system.

*Stable Transactional Memories and Fault-Tolerant Architectures* by M. Banâtre, P. Joubert, C. Morin, G. Muller, B. Rochat and P. Sanchez of IRISA. The talk, given by Christine Morin, described an approach to providing fault tolerance and atomicity by means of a special hardware stable storage device.

*Understanding Transactions in the Operating System Context* by Andrew Black (DEC CRL).

The talk began by discussing the "ACID" properties of transactions (A stands for all-or-nothing, C for consistency, I for isolation (concurrency atomicity), and D for durability), and argued that C was really different from the others since it required a user-supplied "meaning" function in its specification. Then he noted that modern operating systems are concerned more with providing a framework that allows services to be provided and used than with providing the services themselves, and questioned how this applied to transactions, i.e., what tools should be provided by an operating system so that applications can make effective use of transactions.

Andrew Tanenbaum opened the discussion by pointing out that A. Black's statement about an operating system providing a framework for services rather than the services themselves represented a paradigm shift. The question was: is the shift inherent in the move from centralized to distributed systems, or is it simply a new point of view? Black answered that it seemed to be both: the viewpoint was necessary in distributed systems, but was also useful in centralized systems, e.g., Mach.

Paulo Verissimo asked C. Morin whether their approach tolerated arbitrary failures; Morin replied that they tolerated only a single hardware fault.

Fred Schneider wanted to know whether A. Black would still believe that transaction property C was different from the others (A, I, and D) if it were possible to specify it without a meaning function. Black replied that C was fundamentally different because the client must enforce it whereas the transaction system can enforce the other properties automatically. Hermann Kopetz remarked that this view of the I property was somewhat low-level; if viewed at a higher level, I might involve C. Black replied that preserving consistency was stronger than I; all I is concerned with is selecting a serial order. Jim Gray pointed out that C had to do with the net effect of a group of actions, whereas A, I, and D are properties of transactions that support C provided the group of actions is chosen properly; also, C can be defined at many different levels, but not A, I, and D. Flaviu Cristian pointed out that C means that invariants are preserved. I and D do not do this, but if they do not hold, neither does C, i.e., (not(I) and not(D) implies not(C)).

Richard Carr asked C. Morin what was meant by the claim that their STM architecture was an open architecture. Morin replied that you can plug different architectures into STM, but you would need to change STM operations. Le Lann asked about the benefits and problems of using Mach. Morin replied that Mach allowed them to use existing software and the modularity of the code helped them to change the kernel more easily.

Jim Gray asked Daniels to elaborate on his statement that their IPC was twice as fast as Mach's. Daniels replied that the details could be found in their TOCS paper[1].

## Session 4: Data availability

**Chair and Rapporteur:** Özalp Babaoğlu

*Transparent Disconnected Operation for Fault-Tolerance* by James Kistler and M. Satyanarayanan (CMU). Talk presented by Kistler.

Kistler described the design and initial experience with the mechanisms incorporated in the Coda file system that permit file operations to continue despite network partitions. Coda is a descendent of the Andrew file system and achieves high availability by replicating files. In Coda, file replication can be explicit at the server level or implicit due to the caching of files at the clients. One of the goals of Coda is to permit file operations to continue as long as any copy is accessible, even in the presence of network partitions. Allowing disconnected operation is seen as a way of integrating personal computers into a network. The main technical problems have to do with re-integration of a client after disconnected operation. The choices made in Coda try to strike a balance between availability and consistency.

*Fault-Tolerance in Very Large Archival Systems* by Jerome Saltzer (MIT).

Saltzer began his presentation by noting that trends in mass-storage technology and economics are pointing towards less-than-one-dollar costs for digitally storing the contents of a typical book by the end of the century. At these costs, large scale digital archiving of textual information by libraries becomes very attractive. Once, however, data is committed to digital form, care would have to be taken to guarantee its integrity. Data would have to be refreshed periodically to guard against obsolete media or access hardware. Fault tolerance could be added to the achieve by replicating at

[1]Haskin, R., Malachi, Y., Sawdon, W., and Chan, G. "Recovery Management in QuickSilver," *ACM Trans. on Computer Systems 6, 1* (February 1988), 82-108.

several physically-distant sites. Saltzer then went on to identify some potential problems that have to be addressed in realizing such very-long-term and very-large-scale archives.

*One-second Fault Tolerance* by Andrew Birrell (DEC SRC).

In the final presentation of the session, Birrell advocated one-second recovery as a desirable and feasible mechanism for fault tolerance. In retrospect, this topic was somewhat different than his position paper and would have been more appropriate for the "fault tolerance through recovery" session. Birrell's thesis is that if we could build one-second (or less) recovery mechanisms into distributed applications, the result would be perfectly acceptable as a fault tolerance strategy as the user would perceive the system as slightly slow. He then cited a few examples from the SRC environment including Autonet (a fault-tolerant network), a fault-tolerant name service and Echo (a replicated fault-tolerant file system). Birrell argued that certain building blocks (replicated file systems, replicated networks, reliable power supplies) for realizing such systems are already well understood—what is not is how to put it all together and achieving one-second recovery across all levels of the system.

During Saltzer's talk, Jim Gray questioned whether his bits/$ projections were accurate. Saltzer responded that with high probability, the density projections will be available, perhaps not in 10 years but within 15 years.

Gita Gopal asked Birrell if he had considered all of the interactions between the various layers of his system and how close the recovery time was to one second. Birrell responded that they were clos but still over.

Rick Schlichting wondered whether CD-ROM technology was applicable to Saltzer's really-long-term archival storage. Saltzer was not sure how to make effective use of this technology since the current CD-ROM hardware cannot respond quickly to multiple conflicting reads (slow seeks).

Robert Cooper asked Birrell whether he had considered hardware multicasts, to which Birrell replied no because multicast was "evil."

Andy Tanenbaum asked Kistler if they had considered using prediction algorithms to decide which volumes to keep and which to junk. Kistler responded that they had not basically because it was very hard.

Babaoğlu remarked that to satisfy Birrell's desire for fault-tolerant applications, one has to view fault tolerance as an end-to-end argument—a fault-tolerant communications network is no guarantee that a file system built on top of it will be fault tolerant. One has to worry about fault tolerance at all levels of the system. Birrell agreed.

Sape Mullender asked Kistler why it was desirable to have transparent transition from connected to disconnected operation, reasoning that by informing users that they are disconnected, they could prepare for the worst. Kistler agreed that warning the user may be good idea but that they did not want the user work to be disrupted.

Santosh Shrivastava asked Birrell if failure detection through "pinging" could be effective in achieving the one-second fault tolerance. Birrell was skeptical that this mechanism could be the basis for a fault detector for his needs.

Lindsay Marshall expressed concern that in Saltzer's vision, reliable scanning of text could be a serious problem. Saltzer replied that the issue was outside the scope of his presentation, and that in any case, librarians would probably be using microfilm in addition to electronic archival of books.

Schneider argued that a more self-contained medium than disks might be better suited for Saltzer's needs. He noted that it would be difficult to read in bed with a terminal on one's lap and that such a self-contained medium would also alleviate the refresh problem. Saltzer responded that what he had in mind was more a "research library" than an "entertainment library." Schneider replied that this is what he had in mind as well in that it is precisely the user of a research library who wants the guarantee that the information he is getting is authentic and could not have been altered. Physical isolation and self containment are ways towards achieving this property. Saltzer basically agreed.

Finally, Bill Weihl asked Kistler why they chose an optimistic approach in handling disconnected operation. Kistler responded that they could have adopted a pessimistic strategy which would have made re-integration simpler but one needed to know the degree of write-sharing of files to make a better comparison.

## Session 5: Group communications

**Chair:** Richard Carr. **Rapporteur:** Özalp Babaoğlu

This session, informally called "The Battle of the Broadcasts", contained four presentations.

*Fault Tolerance Using Group Communication* by Frans Kaashoek and Andy Tanenbaum (Vrije Universiteit, Amsterdam). Talk given by Kaashoek.

After arguing the advantages of reliable multicasting as a programming paradigm, Kaashoek went on to describe the implementation being carried out at the Vrije Universiteit. This implementation, based on a broadcast network, guarantees a global order for the receipt of messages and has cost comparable to a RPC. He then presented a parallel implementation of a solution to the Traveling Salesman Problem were fault tolerance is obtained for free through the use of reliable multicasts.

*Fast Causal Multicasts* by Pat Stephenson and Ken Birman (Cornell University). Talk given by Stephenson.

In this talk, Stephenson described a new protocol for efficiently implementing reliable, causally ordered multicasts. The protocol was developed in the context of the ISIS system as a replacement for the more costly CBCAST protocol for a majority of cases. Communications that satisfy the requirements for the new protocol *bypass* the old ISIS system and incur little overhead. Communications that cannot be bypassed are handled in the old way. The new protocol (FCBCAST) can handle multiple, overlapping process groups and has message overhead propostional to the current number of members of a process group. Stephenson also argued about the appropriateness of reliable causal multicasts as a programming paradigm for fault-tolerant distributed applications.

*Reliable Atomic Broadcast in Distributed Systems with Omission Failures* by G. Bres (CGE) and G. Le Lann (INRIA). Talk given by both authors.

The authors presented an algorithm and argued its correctness that is able to achieve reliable atomic broadcast in the presence of receive-omission process faults. They argue that this failure model is more realistic in in that it includes both process and communication failures. The requirements for the problem that the authors formulate are also stronger than usual. The algorithm is based on quorums and global sequencers.

*Atomic Broadcast in One Phase* by Pascale Minet and Emmanuelle Anceaume (INRIA). Talk given by Minet.

Minet described an environment based on multiple broadcast channels and special hardware network interfaces called *Network Attachment Controllers* (NAC) such that atomic broadcast is achieved in a single phase in the failure-free case. The NACs are assumed to be weak fail-silent while the broadcast channels exhibit omission failures.

At the conclusion of the presentations, there was an attempt (mainly by Fred Schneider) to put the algorithms presented in context with earlier theoretical results. In particular, there was concern that some of the claims made were inconsistent with certain lower-bound results, or in some cases, there were earlier results that subsumed those presented. This inquiry met with limited success. A few of the exchanges follow.

Schneider addressed a question to the entire panel of speakers about why the Fischer, Lynch, and Patterson impossibility result does not apply. Schneider restated the result that distributed consensus is impossible in asynchronous environments in the presence of even one crash failure. Only Frans Kaashoek got the opportunity to respond.

Kaashoek: We are working in a synchronous environment so the impossibility result does not apply.

Schneider: How do your algorithms relate to the lower-bound time complexity result $min((t+1),(f+2))$ rounds, where $t$ is the maximum number of failures and $f$ is the number of actual failures?

Kaashoek and Tanenbaum: We optimize for the no failure case.

Thomas Becker: Why are there different functions for sending and receiving? Why not integrate to one primitive?

Kaashoek: To allow concurrency.

Cristian: How does your (Kaashoek and Tannenbaum) algorithm compare to that of Chang and Maxemchuck?

Kaashoek: It is similar but they require more interrupts, and use negative acks and their recovery is totally different.

Verissimo: Our system/approach (as INESC) looks at receive-omission failures. Can you comment on or compare your approach to ours?

Le Lann: Our properties P1 and P4 are different. P1 is a Toueg-style requirement on uniform agreement. P4 is eventual termination with the proviso that our assumptions on the partion are met.

Carr: (To Minet) How do you implement the long channel-busy required by a failed sender?

Minet: In hardware.

Warne: (To Stephenson) Could you compare/contrast FCBCAST (bypass) to Psync (of the x-kernel)?

Stephenson: Bypass maintains causality in overlapping groups.

Weihl: (To Kaashoek) Do you allow overlapping groups?

Kaashoek: No.

Schlichting and Pat Stephenson then had a general discussion on what groups were (Pat said distributed objects) and whether hierarchically-structured groups presented any problems (Pat said none).

Black: Which is more powerful? ABCAST or CBCAST?

Stephenson: You can build ABCAST out of two CBCASTs so ABCAST is more powerful.

Andy Tannenbaum then got back on the"What is causality" thread to which there was a general raucous with many rabid participants shouting and frothing at the mouth. The most important aspect of this was that it ended. Also important was Pat stressing that causality need not be consistent with the 'real world'.

Cristian: What happens if sender $a$ crashes and $a \rightarrow b$ and process $p$ gets $b$ but not $a$?

Stephenson: There's piggybacking, message logging and then garbage collection.

## Session 6: Fault tolerance through recovery

**Chair and Rapporteur:** Jim Gray

The session began with three presentations:

**Transparent Recovery in Distributed Systems** by David Bacon (IBM Thomas J. Watson Research Center).

David Bacon presented his ideas on optimistic recovery. He began by explaining that transactions are insufficient because "they only recover data, not process states". A second problem is that the programmer must design his application as a series of transactions, and must handle errors — it would be nice if fault handling were transparent to the application. These two assertions gave rise to lively debate later in the session. First, the point was made that process state *is* data, persistent programming languages make it transactional. Second, there was concern about the fault model: applications must handle faults in application inputs and application state, so it is little extra burden to handle the relatively rare hardware and systems faults.

Bacon rejects both of these viewpoints — he wants to provide a fault-free execution environment. The easy way to do this is to occasionally checkpoint each process state, and to keep a log of all its inputs and outputs. If a process fails, recreate it at the checkpoint state and replay all subsequent inputs to reconstruct its most recent state. This is the approach taken by Auragen and subsequently by Nixdorf. Maintaining the log can be quite expensive. Bacon proposes an optimistic recovery scheme which reduces the number of messages that must be logged, and which allows logging to be done asynchronous to the computation. The penalty for these two optimizations is that a causal dependency graph must be maintained by each process (versions of each process sending an incoming message), and one process failure may cause "sympathetic" rollbacks of other processes. A commit mechanism is introduced to synchronize all processes and so avoid such sympathetic rollbacks. A second optimization, called instant logging, reduces the need to wait for most such commit steps.

The optimistic recovery algorithm has been implemented for Mach tasks (without threads or multiprocessing).

*Transparent Optimistic Rollback Recovery* by David Johnson and Willy Zwaenepoel (Rice University). Talk presented by Johnson.

Johnson reported on work similar to Bacon's but using the V Kernel. They have a formal model for optimistic recovery and dependency vectors. Based on this model, they show that in case of a failure, there is a unique "maximal" consistent system state. Optimistic logging, checkpointing, and recovery have been implemented in the V Kernel. They add about 10% overhead to message sending. Checkpointing a process takes about 3 seconds/megabyte (so about 30 seconds). During this time, the process is suspended. Recovery time is about 1.5 seconds/megabyte (so about 15 seconds) plus the replay time (about 1/2 of the checkpoint interval). An instance of a large scientific calculation (with little communication) was run with 4% overhead. All these are times on a Sun 4.

The Rice implementation supports multi-threaded processes by falling back on a synchronous (pessimistic) message logging technique in which each thread dispatch is a message. At the next checkpoint the process can return to optimistic message logging. Commitment of messages to external agents forces the message log of the sending process and all processes that it depends upon for inputs.

Johnson and Zwaenepoel are generalizing their implementation to support a dynamic choice between checkpointing and message logging, and also to support distributed shared memory.

*Availability in the Sprite Distributed File System* by Mary Baker and John Ousterhout (University of California, Berkeley). Talk presented by Baker.

Baker presented the approach the Sprite file system at UC Berkeley is taking towards fault tolerance. They observe that availability is MTBF/(MTBF+MTTR). So, if MTTR (Mean Time To Recover) is instant, one has very high availability. At present, Sprite and most other Unix file systems take a long time to repair, the run a scan of the entire disc. This can take hours. In addition, Sprite is a "stateful" client-server file system to get improved performance. But reconstructing this state takes a long time (about 15 minutes at present). The initial Sprite goal is a 90 second recovery time. The long term goal is a 10 second recovery time.

One must appreciate that the Unix world has a fairly relaxed view of file integrity. If you update and close file and if the system does not fail for a minute or so, then your updates are likely to be there the next time you look at the file. Otherwise, none, some, or all of the changes may be present at a later time. Clearly, Unix is more concerned with performance and price/performance than with correctness in case of failures. Given these relaxed correctness criteria, there is little Unix interest in duplexed discs, or replicated file servers. Rather the interest is in exploiting new cheap disc arrays and massive main memories.

The "modern" approach to quick is to use database logging techniques to maintain the directories (not the actual data). These techniques give server restart times measured in seconds since the disc directories and free space can be brought up to date by applying the log of recent directory changes. IBM's AIX implementation for the RS/6000 is the best example of this. Sprite takes a different approach: The new Sprite log structured file system treats the entire file system as a sequential log. As data is changed or created it is written at the end of the log. The log is mapped to the disc as a ring buffer. Once the log wraps, the "old" disc data is read and compressed (dead data is discarded) while live data is added to the end of the log. With this logic, the server can support very high read and write data rates, and can recover the disc state in a few seconds by finding the current end-of-log.

Once the server has recovered, it must communicate with the clients. Since they are stateful and have cached data, the clients are all waiting for the server to return so that they can reopen files and send in cached updates. This creates a "recovery storm" in which the server becomes overloaded by the clients, the clients time out and decide the server has crashed again, and so the clients reinitiate recovery. To reduce these recovery storms, a NACK (negative acknowledge) was added to the server to let it tell the client to slow down. This reduced client- server state reconstruction from 900 seconds to 100 seconds. Clients typically reopen over 200 files. By batching client 300 requests in one message, considerable RPC and server time is saved. Also, read-only cache entries need not be recovered immediately, and other cache entries can be delayed until needed. The net of these optimizations should be a further reduction by a factor of 4. This should meet the 90 second repair time for the file server.

The discussion that following Baker's presentation asked what happened to files that were incrementally updated. If a whole-file is changed, then the old file is dead and the new file is stored contiguously at the end of the log. But, if only a single page of a file is changed, then that page and the file map (inodes) are added to the end of the log, but the rest of the file is left unmoved. So incrementally updated files will become declustered on disc. Declustered files have poor read performance if there is logical locality of reference. It seems that the Sprite file system is designed for applications which rewrite whole files. This is typical of edit-compile-go applications, but not of database applications.

The discussion concerning optimistic recovery techniques was heated but unfocused. The theme of all the questions was that optimistic recovery did not have an end-to-end model of faults. It seems to focus on faults below the OS interface and ignore faults above it. These days, most software is above the operating system and most faults are software faults. To give a sense of the discussion:

Christian: What is the failure model of optimistic recovery?

Bacon: Lost messages and failed processes.

Johnson: Partitioning is another failure mode.

Danniels: Concurrency failures such as message deadlocks or resource timeout are an unavoidable form of failure.

Bacon: Deadlocks are not failures.

This lead to the following discussion which can be summarized approximately as:

Liskov: Transactions give an application programming style for thinking about faults and handling them. Optimistic recovery does not seem to help the application programmer handle the inevitable fault above the process- message level.

Bacon and Johnson: We provide a fault-free execution environment. Faults above the interface are not faults in our sense.

## Session 7: Practice and experience with fault-tolerant systems

**Chair and Rapporteur:** Hermann Kopetz

In this session, four papers about fault tolerant distributed application in the real world have been presented. Three out of these four applications have been involved with real-time requirements.

*The ISIS Project: Real experience with fault tolerant programming concepts* by Ken Birman and Robert Cooper (Cornell University). Talk presented by Cooper.

The presentation focused on those properties of the ISIS system that have been most successful in the real world. At the top of this list stands the "virtual synchrony model" that makes it possible for a process to infer the state and actions of remote processes using local state information and events that have been locally observed. Using this model, many users have produced robust distributed applications with no prior experience. Since users need interoperability with existing data-base systems, software packages and other communication mechanism, the decision to implement ISIS on top of UNIX has been appreciated. However, UNIX performance is now hurting because of high kernel overhead and poor scheduling of "server" processes.

*Lessons from Norstar's Distributed Call Processing System* by Liam Casey (Bell-Northern Research).

Casey described the architecture of the distributed Norstar PBX system. The call processing software is structured as a number of independent tasks—extended fine state machines—operating in parallel, and communicating by means of periodic functional messages on a broadcast channel. There is no acknowledgement or retransmission. Such a broadcast can be viewed as nothing more than a notification of state change. No entity can force another to act. This puts the emphasis on task autonomy and helps to enforce fault containment. Whenever a fault is detected, Norstar consistently adopts forward error recovery of a peer entity, since it does not make sense to go back to an "old" state in a real time application.

*Software Fault Tolerance in Telecommunication Systems* by Gita Gopal and Nancy D. Griffeth (Bellcore). Talk presented by Gopal.

The talk reported the new requirements in telecommunication systems. In the traditional approach, the tight coupling between application software and system software with an integrated error handling strategy, resulted in a tremendous amount of code for fault tolerance and network management ( in a typical example 5000 lines of code for call processing versus 200,000 lines of code for fault tolerance an network management). In the future, the increasing complexity of the application services, the heterogeneity of the hardware and the distribution of services requires a new strategy. The open telecommunication architecture of the future starts looking like a large distributed system running applications. It consists of an extremely reliable kernel that hides the heterogeneity of the hardware and provides standard interfaces to the applications. Since the applications in the telecommunications context are not independent of each other, a new paradigm for interacting applications in a switching system similar to the transaction paradigm is required—it is called a clan.

*Fault Tolerance in the Advanced Automation System* by Flaviu Cristian (IBM Almaden Research Center).

Cristian portrayed the architecture of the future US air traffic control system that is being designed by IBM. Each local computer complex is structured hierarchically by using a small number of primitive concepts, such as service, server and the "depends" relation. A server implements a service without exposing to the clients the internal service state representation and operation implementation details. To ensure that a service remains available to clients despite server failures, a number of redundant, physically independent servers are combined to form a server group. Failures of group members can be tolerated as long one member of the server group remains intact. This requires that server groups are closely synchronized and that the servers have omission failure semantics.

The following discussion centered on the talk by Flaviu Cristian. Santos Shrivastava asked whether the use of the ISIS architecture was considered. Cristian replied that they needed synchronous communication and ISIS only provides asynchronous communication. Fred Schneider was intrigued by the primary-backup scheme and asked about the application of time-outs and the possibility of confusion by the parallel operation of primary and backup servers. Flaviu answered that a mutual surveillance scheme can detect failures in less than 100 msec and that temporary confusion is preferable to a loss of availability. The more general question by Fred Schneider "If you had a choice to make one small hardware modification to make your job easier, what would that be?" was answered by Robert Cooper "Make the hardware failstop and fail fast", by Liam Casey "Distributed switching, so there is no single point of failure" and by Flaviu Cristian "There is no hardware better than IBM hardware".

## Panel discussion

**Moderator and Rapporteur:** Marc Shapiro

A panel consisting of Michel Gien, Andy Tanenbaum, Rick Schlicting, Bill Weihl, David Johnson and Jim Gray was asked to answer the question: "In designing a new operating system, what two primitives or abstractions would you include explicitly for support of fault-tolerant applications?"

*Michel Gien* advocated micro-kernels in general, and Chorus in particular. Micro-kernels provide component isolation and explicit communication. This leads to a better understanding of each component and the relationships between them. It also allows for distribution and redundancy. Chorus are adding a group communication mechanism based on ISIS to their kernel. With the addition of message logging and replay, they will develop a fault-tolerant distributed Unix.

*Andy Tanenbaum*: the classical mechanisms mask hardware errors. But hardware errors are rare; it's the software that's rotten. The goal of OS designers should be to make the whole system reliable, not just mask hardware errors. Andy suggests two general design principles for operating systems:

- Minimal kernels — everything in user space.
- Statelessness: use RPC without connections to communicate; stateless file servers; eschew internal table state; damn and condemn internal timers.

Reluctantly, he admits the usefulness of reliable multicast in the kernel.

*Rick Schlicting* spoke next. The kernel should have communication support, and structuring support. Communication should adapt to application requirements: e.g. real-time is harder. Psync is lightweight, supports group multicast (a fundamental concept), preserves causality order, and is reliable. Structuring means the OS is a basic infrastructure for providing various features. It is easy to compose protocols in x-Kernel; this allows to separate atomicity from, say, ordering.

*Bill Weihl* would provide at-most-once RPC, and some form of group management, including membership protocols, multicast and "anycast". He suggests unbundling the various components of a transaction system (logging, recovery, undo, commitment, and concurrency control), and allowing the programmer to access them separately. So someone constructing a file system could use a logging primitive, for example.

*David Johnson*'s system does transparent, asynchronous message logging on receipt, and occasionally checkpoints processes. Their protocol allows to save uncoordinated state information but restore it consistently. By switching between logging and checkpointing, they can support non-deterministic processes. Message logging is also useful for debugging, giving an additional reason for providing it in the kernel.

*Jim Gray*: An OS is a big thing, because users need those services. Being big they contain a lot of bugs; the "Resource Manager" model (mediating between the application and the transaction manager) helps make the system fault-tolerant. Jim's two features for fault-tolerance are "persistent processes," recreated on failure, and pairs of widely distributed processes (two servers 1000 miles apart), the only way to get better than 10-year MTBF.

The moderator remarks that an OS is a kernel plus a set of services. Most panelists talked about kernels (only); Jim Gray talked about services (only). Any comments?

Weihl: What are the common services? Users shouldn't have to decide, re-implement, or in fact, build an OS.

Gien: The microkernel is what is common to all our, different, types of users. They can build what they need on top of that. The application/OS border is disappearing. It is important to note that the user space vs. system space decision is now a *configuration* problem, not a *design* problem.

Gray: Kernelizing didn't solve the fault-tolerance problem. Kernelizing was tried in the 60s and didn't stick around to the 70s. It sure won't make it in the 90s.

Lindsay Marshall: All I really care about is getting my applications to run. I want a transparent, easy-to-use interface. It should tolerate software failures.

After this discussion, Andrew Black remarks that Tanenbaum's statelessness begs the question. After all, people buy computer systems to store state. "The whole point is that we want persistent state, persistent even when there are faults." Jim Gray responds: "It already exists—it's called a database."

¿From the floor, Mirek Malek proposes to build fault-tolerance mechanism from more basic abstractions. At the lowest level, synchronized clocks give synchronization. From this we can construct reliable broadcast. Then consensus. And finally recovery.

The moderator asks: can reliability really be transparent?

Johnson: With optimistic techniques, fault-tolerance can be transparent to applications. Servers can be more special-purpose: for example, one should not blindly checkpoint a 12-megabyte block cache.

Schneider: Transparency is naive. A historical comparison: for a while people believed in programming languages that would prevent synchronization errors. What works much better is to provide a nice way to pass the problem up a level. For example, sometimes telling the user that I/O didn't occur. This is another instance of the "end-to-end argument".

Marshall: Look, all I want is to know when something has gone wrong, and be given an opportunity to save my file. Instead of masking the fault, give me a fault diagnostic so I can fix it. Once again, software faults are the real issue.

Everybody agrees that it is desirable to provide different degrees of transparency and confinement, as well as a choice of mechanisms.

Jim Gray advocates "defensive programming" techniques against software faults. Make everything fail-fast, and put in short-latency exception handling mechanisms, e.g. redundant data structures. Every module should start by checking its parameters and invariants; if there is a problem, raise an exception and clean up. In ESS, so-called "audit" programs check the consistency of data structures in the background (look for dead pointers, check data integrity at the end of transactions, etc.).

Luca Simoncini: We are providing mechanisms, not policy. We should not provide automatic checkpointing or transactions. But we should provide failure detection.

Hermann Kopetz: We should explicitly state the fault hypotheses our OSes will tolerate. Suggests "fail-complete nodes": if you suspect something has gone wrong, do you keep going?

¿From here a long discussion started about real-time.

Gray: Real time is a very fuzzy term. Existing hardware can very well deal with the time constraints of "real-time" applications ($30\mu s$ and up). So-called real-time systems have ordinary problems.

Edoardo Marelli: Real-time is not a matter of performance; it's a matter of *predictably* meeting deadlines.

Le Lann: Real time is not MIPS, it's a matter of latency. RT programs are task sets with timing requirements. We need smart scheduling strategies.

Keith Marzullo: Real time is really resource-constrained programming. Programs themselves run quickly. Resource contention is the bottleneck.

Kopetz: Real time programs have different characteristics—they deal with the external world, no rollback is possible, there are hard deadlines.

The moderator puts the original question to the real-time specialists: "What two primitives are needed for fault-tolerant *real-time* applications?" In unison, Kopetz, Marzullo, Le Lann, Malek respond: "First, synchronized clocks; and second, synchronized clocks."

Tanenbaum suggests that radio-signal time receivers will provide synchronized clocks easily. Flaviu Cristian and Andrew Black point out that a time signal receiver is $1000. In large installations, that's a big problem. Without them, your system is only as synchronous as your worst clock.

Fred Schneider remarks that synchronized clocks are useful, and not just in real-time; see Liskov's PODC invited speaker talk in which she uses clock synchronization to optimize protocols, etc. Clock synchronization should be low down in the system (in the kernel to minimize variance), for instance based on the Kopetz chip.

Kopetz: it can keep the clocks of 60 nodes synchronized to within $5\mu s$ on a standard Ethernet.

The panel concludes:

Weihl: More than ever convinced that transparency won't work.

Johnson: It works for hardware, not software.

Tanenbaum (the project leader in charge of saying "no" gives in): I'd add reliable multicast and clock synchronization to my list.

## Session 8: Short Presentations

**Chair and Rapporteur:** Özalp Babaoğlu

The following five-minute presentations were made:

F. Cristian, *On Choosing Failure Semantics*
W. Najjar, *Scalability and Reliability of Large Systems*
A. Ricciardi, *The Group Membership Problem*
S. Lalis, *Structuring Distributed Applications*
R. Schlichting, *IPC Support for Fault Tolerance*
L. Leonardi, *The Object Paradigm is to be Reconsidered for Distributed Systems*
J. P. Warne, *Concurrency Control in ANSA Conformant Systems*
M. Shapiro, *Kernel-Level Support for Objects*
V. Gianuzzi, *Interface Problems Between Fault-Tolerant Applications and Operating Systems*
W. Künhauser, *A Kernel Abstraction for Fault Tolerance Support*
W. Weihl, *Fault-Tolerant Parallel Computing*
P. Verissimo, *Atomic Multicasting in Delta-4*

## Session 9: New architectures

**Chair and Rapporteur:** Fabio Panzieri

*Portable Fault-tolerant Systems* by Dan McCue and Santosh Shrivastava (The University of Newcastle upon Tyne). Talk presented by Shrivastava.

Santosh Shrivastava described a portable programming environment, named Arjuna, that can support the development of object oriented distributed applications. The principal features of Arjuna include the provision of program interfaces for the implementation of (i) atomic actions on persistent objects, (ii) concurrency control and recovery management, and (iii) access to and manipulation of persistent objects. In addition, Arjuna supports location transparency and migration and replication of objects.

*Fault-tolerance in Delta-4* by David Powell Marc Chérèque and David Drackley (LAAS-CNRS). Talk presented by Powell.

David Powell described the fault tolerance approach developed within the Delta-4 project. This is an EEC funded project being carried out as part of the ESPRIT program. Delta-4 investigates the relationship between distribution and fault tolerance in order to offer distributed architectures (e.g. CIM, office automation, integrated information processing, process control) with built-in fault tolerance features. In essence, applications supported by Delta-4 can be made incrementally fault-tolerant on a service- by-service basis; in particular, the application designer can choose which services he wishes to make fault-tolerant, and to which degree, at application configuration time.

*A Fault Tolerance Approach for Distributed ISDN Control Systems* by Thomas Becker and Jurgen Nehmer (University of Kaiserslautern). Talk presented by Becker.

Thomas Becker introduced the design of a distributed control system for ISDN telecommunication services. This system, being developed as part of the ESPRIT project DAMS, is required to provide a high level of availability, and to meet (soft) real-time requirements. Fault tolerance and availability requirements are met via replicated data management.

*Some Ideas on Support for Fault Tolerance in Comandos/Aonhgus* by Brendan Tangney et al. (Trinity College, Dublin, Ireland). Talk presented by Tangney.

Brendan Tangney discussed some observations on fault tolerance support in distributed systems which have emerged from work being developed within the context of the Comandos ESPRIT project. Tangney emphasized the notion that fault-tolerance should be transparently available to all applications and, in particular, application level code should require no changes in order to be made fault-tolerant.

The discussion at the end of this Session was initiated by Jim Gray who observed that replicated services, such as a time service, can be difficult to implement; the basic problem is non determinism, as the application program model is complex. David Powell replied that dealing with non determinism requires that passive rather than active replication be implemented. Gray asked if all applications are non deterministic. Powell answered that that is not the case. Gray observed further that, in order to deal with non deterministic applications, one must run the replicas in the same environment. Powell replied that consensus must be provided when non determinism occurs. Santosh Shrivastava pointed out that arbitrary programs cannot be replicated; rather, they have to be structured as message passing systems.

Fred Schneider asked both Powell and Becker whether a “flaky” network may cause the occurrence of two primaries in their systems, as both those systems implement (different) replication policies for fault tolerance purposes. Becker answered that, in his system, this is not a problem. Powell, instead, replied that, in Delta-4, error detection is based on the abstraction of bounded delay, and that the issues raised by a flaky network are assumed to be dealt with at low system architecture levels. Schneider did not understand this answer from Powell; Flaviu Cristian rephrased Schneider’s question in terms of “Do your failure assumptions include partitions?”; Powell answered “No”.

Edoardo Marelli was intrigued by the notion of fail-silent Network Attachment Controller (NAC), used in Delta-4. In particular, Marelli asked whether commercial hardware and software were used to implement NACs, whether modifications were introduced to these hardware and software, and what performance had been measured. Powell answered that the Delta-4 project developed its own software (outside the executive) and hardware; the latter incorporates extensive self checking. Performance testing will be carried out next year; currently, a NULL RPC takes about 40 ms; the project hopes to reduce this figure to 20 ms.

Hermann Kopetz asked Powell whether the use of event-triggered protocols within a system implementing replicated services, such as Delta-4, may kill event determinism. Powell answered that, in Delta-4, bounded time drift is assumed in order to cope with this possibility. Kopetz observed that, in order to define accurately that time drift boundary, the choice between the use of an either average or upper bound is not obvious; in particular, in a system that makes use of protocols based on acknowledgements and retransmissions, an upper bound derived from the maximum time drift can be much larger than one derived from the average time drift. Powell replied that there is no general solution to this problem, and a possible strategy can be based on the use of small timer values.

Flaviu Cristian asked Becker about the implementation of bounded failure detection in a context characterized by unbounded delays, such as that of the distributed ISDN control system described by Becker. Becker replied that they have assumed that "most messages arrive soon". Cristian pointed out that, using this assumption, they have traded "safety for timeliness".

Finally, Cristian asked Powell how the Delta-4 project has solved the "tension between open systems and bounded delays". Powell answered that in open systems bounds can be overestimated; if a bound in some such a system is exceeded then a timing failure occurs. Cristian observed that these timing failures can be a source of partitioning.

## Session 10: Closing Session

**Chair and Rapporteur:** Fred B. Schneider

The final session was divided into two parts. First, Edorado Marelli from the European Space Agency (ESA) made a brief presentation about some of the problems ESA is up against in implementing fault-tolerant, real-time computing systems for use in outer space. A brief question/answer period followed that presentation.

Then, the workshop participants engaged in a rather broad discussion that ranged over topics that were discussed in earlier sessions as well as those that were not (but should have been) discussed during the workshop.

Discussion following Marelli's presentation:

Bacon: Tell us a bit more about the applications you expect to run.

Marelli: I deliberately didn't do this; we want to be as general purpose as possible. But some common elements are: (1) The applications are not complex. (2) They need robust mechanisms to acquire data or drive actuators. (3) Telemetry is often involved. (4) There are real-time demands. (5) Some applications are command and control oriented.

Powell: You really need to specify your applications completely. This is often difficult to do – information such as admissible failure rates and acceptable outage times is hard to come by. But it is needed if the overall problem is not to be overspecified.

Marelli: Yes, applications have different reliability requirements.

Powell: And you must get this information. k-fault-resistant is not a real specification. The value of k must be derived from the acceptable application failure probabilities and the specification of the hardware.

Najjar: Are failure rates specified in terms of mission time?

Marelli: Yes, and missions vary from a few minutes to 30 years. The more complex applications appear on long missions. However, our performance demands are not high. None of these applications have duty cycles greater than 100Hz. What is more more important to us is that the system carries on following failures, i.e. (1) Recovery must work (2) Recovery must bring the system to a known state (Satellites have a few major states)

Najjar: Is a 50 MIPS procesor a weight problem?

Marelli: Yes. Faster chips seem to require more support hardware. Flight applications must be inherently safe.

Kopetz: Do you need new hardware architectures?

Marelli: Ideally, no. it is much easier to validate off-the-shelf hardware. But the reality is that off-the-shelf hardware doesn't fit, and is not space-approved. Maybe we do need a new software architecture.

Warne: You've really outlined a very general problem, and we have no real idea about your actual functional requirements or the scale of your system.

Marelli: My main aim was to trigger a reaction.

Le Lann: Two questions. Firstly, what do you think about existing products (either software or hardware+operating system) that claim real-time? And, how are you going to reconcile the Ada testing model with time-dependent priorities?

Marelli: Second question first. We are using Ada in the space station only. The problems I am discussing here go beyond that. Now to products. Some of them go in the right direction but others don't consider important aspects of our problems. They think that fast context switching is everything. We are evaluating these products, but haven't found anything useful.

Final Session General Discussion:

Schneider: To start out this discussion, it might be instructive to see if we can identify any well-defined issues that have arisen during the last few days. Or, any well-defined issues that have been resolved by our discussions.

Shapiro: I've learned about reliability but not operating systems. Yesterday, we heard the reliability people talking about mechanisms and the O/S people talking about microkernels. There seems to be a gap here. Do we need some sort of modular O/S where processes are forbidden to talk to other processes using different mechanisms? e.g. the processes using optimistic fault-tolerance are walled off from the processes using pessimistic fault-tolerance.

Nehmer: 95% of all contributions presented here were special solutions to problems that we can't specify. We must specify whate kind of fault-tolerance we want to have.

Strigini: (1) We do have real numerical requirements for fault-tolerance. (2) It is hard to validate that we have achieved these. (3) How do we fit computational models into variability models?

Casey: We should construct these applications bottom-up.

Powell: I sympathize, but we have 10 bottom-up solutions. We need requirements to choose between competing solutions.

Cristian: Fault-tolerance means different things to different people: (1) (example Jim Gray) "Nice" failure behavior. (2) Failure masking (either in a hierarchical fashion or with server groups) For each type of fault-tolerance, we use different mechanisms. For example, for (1) we use transactions, communication abstractions, and self-timing components. For (2) we use exception mechansims, membership protocols, broadcast protocols.

Schneider: For me, fault-tolerance is the ability of the system to conform to its specification despite the occurrence of certain events, which we regard as "failures".

Powell: Cristian's two kinds of fault-tolerance are really just different specifications. The main difference between them is the acceptable downtime.

Weihl: How do we define the class of failures to be handled before we know about the implementation?

Marzullo: The failure specification is simply preconditions about the environment.

Schneider: There's a difference between system requirements and system specification. Requirements may be stochastic, specifications are not stochastic. The FAA three-second downtime requirement is a "requirement", not part of the specification. Presumably, it is implied by the specification.

Cristian: I was trying to define fault-tolerance.

Weihl: We can't have mechanisms that prevent mistakes. We need to work on design techniques also.

Coulouris: I'd like to return to Marc Shapiro's suggestion. He want to subdivide the world into parts with internally consistent semantics. We already have a way to do this - computer languages. If fault-tolerance is well understood, we will get a semantics that we can incorporate into a language, for example we could incorporate process groups into a language. Are there other suggestions?

Cristian: Well, ARGUS has done this with transactions. This is for the first type of fault tolerance (nice failure semantics). But, it is harder to do this for failure masking.

Shapiro: We need firewalls between people using different fault-tolerance mechanisms, that's all.

Weihl: We can't have both modularity and transparency.

Gray: Fault-tolerance is an end-to-end property. There is no such thing as a fault-tolerant operating system. It is expensive. It is hard to write fault-tolerant applications. The sort of code ESA and NASA are writing will cost them between $5000 and $10000 a line. Normal vendor code costs about $500 a line. This argues strongly for standard software in order to reduce the cost.

Kopetz: Fault-tolerant operating systems are a mechanism for constructing fault-tolerant applications. We must always start with the application specification. The O/S should have tools needed by the application. For real-time applications we need: (i) clock synchronization, (ii) a group membership service, (iii) fail-fast processors, (iv) predictable performance, (v) and, we must be able to validate all this.

Cristian: Fail-fast is nice, we need mechanisms to support both types of fault-tolerance.

Malek: Here's another point of view on the two type of fault-tolerance. We're really doing redundancy management, either in space or in time. Space-redundancy corresponds to fault masking. Time-redundancy corresponds to the use of recovery mechanisms, pretty close to Cristian's "nice failure semantics."

Schneider: There seem to be two views about software fault tolerance. (1) That it's the same as hardware fault-tolerance; (2) that it's different.

Gray: Hardware fault-tolerance is a solved problem. There are lots of anecdotes about software fault-tolerance. Some important points from the anecdotal evidence: (1) We need to do lots of testing. (2) We can't specify it. (3) Therefore, we can't validate it. Transactions are an ad-hoc technique for dealing with software fault-tolerance. Hardware is not expensive, software is expensive.

Cristian: For software FT1 (Nice failure semantics), we can use match- and-compare techniques. For software FT2 (Failure masking), the current techniques are not effective for software failures. E.g. voting is not reliable.

Strigini: We want to manage fault-tolerance at the application level, so we must use the usual end-to-end arguments.

Johnson: Gray said transactions are useful for software fault-tolerance. I don't agree.

Gray: The cross-checks and assertions are useful. One can build a self-checking system. But these checks must be in the application—so there is no real solution to software fault tolerance yet. This sort of fault-detection is really just defensive programming.