Computer Architecture Technical Committee

NETWORKS ARCHITECTURE SYSTEMS

NEWSLETTER

JUNE 1985

TABLE OF CONTENTS

Guest Editorial on New Trends in Dependable Computer Architecture in Europe . . . 1
L. Simoncini

The "Saturne Project": A Fault- and Intrusion-Tolerant Distributed System . . . 4
Y. Deswarte, J.C. Fabre, J. Fraga, J.C. Laprie and D. Powell

Reliable Computing in a UNIX United Environment . . . 23
K.H. Bennett, L.F. Marshall and B. Randell

New Trends in Dependable Computer Architecture in West Germany . . . 39

The Design of Fault Tolerance in the UPPER System . . . 40
P.M. Behr and W.K. Giloi

Attempto: A Testable Experimental Multiprocessor System with Fault-Tolerance . . . 46
M. Dal Cin, R. Brause, E. Dilger, J. Lutz and T. Risse

Fault-Tolerant Dirmu Multiprocessor Configurations . . . 51
E. Maehle

Future: A Fault-Tolerant Multimicro-Computersystem with Taskspecific Redundancy . . . 57
F. Demmelmeier

RDC: A Fault-Tolerant Real Time Computer System with Distributed Microprocessor Stations . . . 64
G. Bonn and F. Saenger

MARS: A Maintainable Real Time System . . . 70
H. Kopetz and W. Merker

Support for Transaction and Recovery in CNET Applications . . . 87
P. Ancilotti and M. Fusani

The MuTeam Distributed Multiprocessor Architecture . . . 97
P. Corsini, L. Simoncini and L. Strigini

The Tomp Multiprocessor Project . . . 105
G. Conte and D. DelCorso

MODIAC: A Distributed System for Industrial Automation . . . 111
G. Bruno and L. Ciminiera

IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY

ATTEMPTO - A testable experimental multiprocessor system with fault-tolerance

M. Dal Cin, R. Brause, E. Dilger, J. Lutz, Th. Risse
Institute for Information Sciences
University of Tuebingen
D - 7400 Tuebingen, FRG

Abstract: This paper describes the overall hardware and software structure of a fully decentralized, fault-tolerant system for a single-user multi-tasking computing environment. Currently, the system is intended for use as a test-bed for fault-tolerant computing.

1. Introduction

With the rapid decline in the cost of computer hardware it is now feasible to dedicate even a multi-processor system to a single user, contrary to a few years ago. Consequently, we felt that it makes sense to exploit the advantages of multiple resources provided by a multimicroprocessor system and to develop a single-user fault-tolerant computing environment. This would allow us to substitute such resource-consuming techniques as user time-sharing and inter-user protection by techniques implementing fault-tolerance. The system has been called ATTEMPTO /1/ (A TesTable Experimental MultiProcessor with fault-TOlerance) and is intended to serve to the developing team as a test-bed for fault-tolerance mechanisms.

The prevailing design goals are:
- The user himself should be able to decide for each application job - according to its criticality - to what extent it should run in a fault-tolerant environment.
- The mechanisms implementing fault-tolerance should be transparent to the user who observes the system as a multi-tasking monoprocessor system.
- All fault-tolerance mechanisms - such as fault-diagnosis, voting or reconfiguration - should be fully decentralized and reconfigurable in order that the system can survive the breakdown of single components and in order to be able to perform experiments on fault-tolerance by adding or exchanging software and hardware components.
- The system is to be built from conventional hardware parts; most of its fault-tolerance is to be implemented in software.

These requirements led primarily to the development of a modular, hierarchically structured operating system layer /2/ which provides the fault-tolerance services of ATTEMPTO.

After a brief review of the currently used hardware (Sec. 2) and our concept of fault-treatment (Sec. 3), we present the overall structure of the operating system layer (Sec. 4).

2. Hardware Structure

As processing nodes single-board computers with dual-port RAM are chosen (Intel-Products). Communication between these nodes is provided by the Multibus. To ensure that none of the processing units damages the user's input data, the user input is directly available to all units (by connecting the user terminal to the serial i/o-port of each board). Via the dual-port RAM's a unique logical communication link is established between each pair of processing nodes. For each processing unit there is one interrupt line on the communication bus. These interrupts are used for a special synchronization protocol /3/ which allows an asynchroneous, atomic transmission of messages. (The temporal order in which incoming messages are accepted is the same for all processing units. However, it may be different from the temporal order of their individual arrival).

3. Fault Treatment

With regard to fault-treatment we adhere to an end-to-end strategy /4/ as follows. Copies of an application job are executed asynchroneously in parallel by several processing nodes. Failures of individual nodes are ignored as long as they do not affect the user. They are, however, diagnosed and masked before they may have an effect on the environment (wrong or no output of user job). Fault-diagnosis is based on the so-called job-result comparison approach /5/. Several distributed diagnosis protocols for job-result comparison have been investigated and verified by Time Petri Net analysis /6/. Idle nodes perform self-test routines. This approach is conceptually simple and independent of the hardware structure and of failure types.

Nodes executing copies of an application job are referred to as colleagues. They form a single virtual processing node. The size of a virtual node can be specified by the user at program start. For instance, a virtual node with two processing units provides fault-detection only; virtual nodes with 3 (4, 5, or 7) colleagues can tolerate failures of 1 (2, 3 or 5) colleagues.

The binding of application jobs to processing units is transparent to the user. It is based on the principle of job-attraction /7/ (implemented in software) in order to avoid the need for centralized scheduling and dispatching. Each node maintains its own system tables, which it updates upon receiving messages from other nodes. The above mentioned synchronisation protocol guarantees consistent (decentralized) system tables.

4. System Software Structure

ATOS is the node operating system of ATTEMPTO. It is comprised of two parts: The OS-Kernel and the Fault-Tolerance Layer (FTL) which is responsible for implementing the fault-tolerance. This layer is transparent to the user and is programmed in Modula-2 /8/. The FTL itself is divided into several sub-layers /2/, viz:
- the Fault-Tolerance Instance FTI,
- the communication support layer,
- the service layer and
- the system layer.

The first two sublayers consist of collections of specific Modula-2 modules. Each module comprises a data structure (e.g. Job Control Buffer, Data Input Buffer, Signature Array Buffer, etc.) and an active unit that maintains the data structure. Active units are referred to as Module-clerks and are Modula-2 processes. Clerks communicate by exchanging messages. The second pair of sub-layers is composed out of a set of information concealing modules with strictly procedural interfaces.

Hence, the architecture of the higher part of FTL is based on the message oriented model of Lauer and Needham /9/ and that of the lower part is based on the procedure oriented model. From the viewpoint of the OS-kernel, the FTL is just another user process (with high priority) that shares its processor time among several Modula-2 processes. Its objective is to provide fault-tolerance if required by the user. The FTL also provides complete internal observability to the experimenter (but not to the ordinary user). Moreover, its modular and hierarchical structure and the fact that it is exclusively programmed in a higher level language allow us to substitute single modules by modules implementing different strategies for fault-treatment.

We now characterize very briefly the function of each sublayer:

The FTI provides high level fault-tolerance services. Its core is formed by the modules DIB, DOB, SAB and FTD. The DIB-Clerk manages the data typed in by the user (input buffer) and prepares the user-job output data for fault-diagnosis. The DOB-Clerk manages the user job data output buffer and forwards only the data which are diagnosed as being correct. The DOB-Clerk is authorized to do so by the SAB-Clerk. The SAB-Clerk handles all diagnosis tasks. To this end it maintains a so called signature-array buffer. In order to be more efficient in the use of the communication bus the computation results of a user job are first compressed to a feasible normed length by the SAB-Clerk and then compared with the corresponding results delivered by its colleagues via the communication bus. Data compression is done by computing a signature of 16 bits (i.e. of the bus width). The FTD-Clerk (Fault-Tolerance Dispatcher) manages a Job Control Buffer and implements the principle of job attraction.

The communication support layer (Communication Instance, CI) is responsible for correct communication between an application job and its FTI as well as between the FTI's of different processing nodes. CI contains a module called Post-Office (PO) which constitutes the interface to the OS-kernel. In order to send a message to colleagues a clerk sends this message to its Post-Office. The PO-Clerk completes the message with additional information (e.g. the node-id) and delivers it via the OS-Kernel to the communication-port handler. The PO-Clerk forwards also all incoming messages to the receiver clerks of the FTI.

The service layer provides services necessary for Modula-2-process management, buffer management, resource management, etc. The system level provides services for storage management, context switching and system calls.

It is worth mentioning that, although our technique is not specific to any particular implementation of the OS-kernel (it is only essential that the kernel is able to distinguish fault-tolerance requests and local system calls), our prototype is intended to run under local UNIX-kernels. Roughly speaking, system calls are diverted to the FTL if fault-tolerance requires this. The decision is made by the kernel routine CIkernel which gains control again as soon as the fault-tolerance service has been delivered by the FTL. This technique offers several advantages:

- Every runnable code can be executed fault-tolerantly without modifications in response to the user's wishes;
- Changes of the kernel that become necessary remain local and controllable since there is only one entry point into the kernel;
- The entry to the FTL is protected just as entries to the OS-kernel are;
- The kernel routine, CIkernel, can easily be attached to any operating system kernel (pseudo device);
- The method is more or less machine independent.

During development we emulated the system on a minicomputer. Currently the emulation is being upgraded in order to serve as a testbed for other fault-tolerance purposes.

5. Conclusion

Designing ATTEMPTO we confined ourselves to considering only those fault-tolerant concepts which we felt to be fundamental and which did not require extensive hardware modifications. We are, however, convinced that the proposed combination of asynchronous fault-masking and distributed fault-diagnosis compare favorably with techniques /10/ such as checkpointing and roll-back.

The work has been supported by the Deutsche Forschungsgemeinschaft under Contract Da 141.

References

/1/ Ammann E., Brause R., Dal Cin M., Dilger E., Lutz J., Risse Th., ATTEMPTO, a fault-tolerant multiprocessor work station: design and concepts, Proc. FTCS-13, Milano, pp 10-13 (1983)

/2/ Risse Th., Brause R., Dal Cin M., Dilger E., Lutz J., Entwurf und Struktur einer Betriebssystemschicht zur Implementierung von Fehlertoleranz, Proc. Fehlertolerierende Rechensysteme, Großpietsch K.-E., Dal Cin M. (Eds.), Informatik-Fachberichte 84, Springer-Verlag, pp 66-76 (1984)

/3/ Brause R., Ammann E., Dal Cin M., Dilger E., Lutz J., Softwarekonzepte des fehlertoleranten Arbeitsplatzrechners ATTEMPTO, Symp. German Chapter of ACM Microcomputing II, W. Remmele, H. Schecker (Eds.), Teubner Verlag, pp 328-341 (1983)

/4/ Saltzer J.H., et al., End-to-end arguments in system design, Int. Conf. Distributed Computing Systems, Paris pp 509-512 (1981)

/5/ Ammann E., Dal Cin M., Efficient algorithms for comparison-based self-diagnosis, Proc. Self-diagnosis and Fault-Tolerance, Dal Cin M., Dilger E. (Eds.), Attempto-Verlag Tübingen, pp 1-18 (1981)

/6/ Dal Cin M., Florian F.-H., Analyses of a fault-tolerant distributed diagnosis algorithm, preprint Tübingen 1984

/7/ Katsuki D. et al, PLURIBUS - an operational fault-tolerant multiprocessor, Proceedings IEEE Vol. 66, pp 1146-1159 (1978)

/8/ Wirth N., Programming in Modula-2, Springer-Verlag 1982

/9/ Lauer H.C., Needham R.M., On the duality of operating system structures, Operating Systems Review 13, pp 3-19 (1979)

/10/ Randell B., System structure for software fault-tolerance IEEE Trans. on Software Eng. SE-1, pp 220-232 (1975)