AN INTEGRATED APPROACH TO TRANSPUTER-BASED FAULT-TOLERANT SYSTEM DESIGN WITH SPECIAL EMPHASIS ON GLOBAL STATE CONSISTENCY

THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY (ENGINEERING) OF JADAVPUR UNIVERSITY

BY
ANUPAM SINHA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JADAVPUR UNIVERSITY
CALCUTTA-700 032
INDIA
1997
AN INTEGRATED APPROACH TO TRANSPUTER-BASED FAULT-TOLERANT SYSTEM DESIGN WITH SPECIAL EMPHASIS ON GLOBAL STATE CONSISTENCY

SYNOPSIS

THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY (ENGINEERING) OF JADAVPUR UNIVERSITY

BY ANUPAM SINHA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING JADAVPUR UNIVERSITY CALCUTTA-700 032 INDIA 1997
1 Introduction

The design of fault-tolerant systems is a challenging problem. It requires close attention to many issues, e.g., the fault-model, the connectivity of the system topology, the concurrent programming paradigm used (say, for instance, synchronous message passing), the message transfer delays of the communication network, the support for scheduling real-time processes, etc. A vast amount of work has already been done in this field. However, there is a noticeable lack of a unified theory of design. In this scenario, we strongly feel the necessity of a design methodology that integrates different techniques to build a system that has satisfactory behaviour. This is the motivation behind the present work.

In our work, we have employed the Transputer and its native language Occam. However, this is by no means a limitation of this work. The methodology presented herein can be adapted to any other message-passing architecture. We also observe that Occam is a convenient means of expressing concurrency, irrespective of any specific implementation, e.g., the Transputer.

The present work is suitable for critical real-time applications like launch vehicles, digital flight control systems, and nuclear power plants. It can also be used in process control systems. The work as presented in the dissertation has been largely supported by two sponsored projects from the Indian Space Research Organization.

2 Objectives

Our main objective is to present a System Design based on sound theory. By doing this, and by avoiding ad hoc techniques, we can produce a robust design.

The translation of theory into practice involves many hurdles. For example, the design of replicated state machines requires us to implement signed messages, a process architecture that avoids deadlocks and starvation, Atomic Broadcast, and Voting. Our journal paper illustrates how this can be achieved.
We also assert the necessity of integrating the fault-detection [SCD91] and fault-masking paradigms. This is required, for instance, to reintegrate a replica that has lost its internal state due to a transient fault.

Continuing with this approach of integration, we seek a unifying principle that is applicable in all paradigms of fault-tolerance. We assert that global state consistency is one such principle. Its importance in Checkpointing and Recovery and replica determinism is well-established. If processors have synchronized clocks, the question of global state consistency can be settled elegantly.

### 3 Approach adopted

The System Design problem requires us to overcome some hurdles. Some of these are platform-specific: the language Occam has certain inadequacies as far as fault-tolerance is concerned. For instance, Occam models nondeterminism by the ALT construct but the Transputer implementation of ALT is neither fault-tolerant nor “fair”.

Other problems are of a more general nature. For example, the establishment of a global timebase requires us to find upper bounds on message transit times between processors. With a microcoded scheduler, we have practically no control over scheduling at the level of software. Hence, determining these upper bounds is a difficult problem. Theoretical analysis of timing involves the detailed study of the source code and the memory layout of the code and data. Our approach to the design involves an integration of theoretical tools of analysis, e.g., Integer Programming, and technology-specific details, like the cycle times of Transputer instructions. Hence this work illustrates practice based on sound theory.

For a given fault model, we select a topology having high reliability. The design of the process architecture in a processor is our next concern. Processes should be free from deadlock and starvation. The prime goal of the fault-tolerant software is to ensure global state consistency. This can be achieved, for instance, by Atomic Broadcast in replicated processing or by taking checkpoints in the fault detection and recovery
approach. The functional correctness of the software is closely coupled with timing correctness. For this reason, we perform a theoretical analysis of timing properties. The results of this analysis are used for setting values of parameters, e.g., time-out intervals in the software.

The global timebase provided by Software Clock Synchronization serves as a basis for integrating different techniques. It is required for implementing Atomic Broadcast, for taking checkpoints, and for timestamping the results of any distributed agreement.

To sum up, we have demonstrated how a proper integration of fault-tolerant message transmission, Agreement protocols (e.g., Interactive Consistency), authenticated messages, and a global timebase enables us to design the software for a fault-tolerant real-time system. We have demonstrated our approach by providing the structure of the source code, experimental results, and theoretical analysis of timing properties.

4 What have been achieved

The main achievement of this work is the development of a methodology for Fault-Tolerant System Design using Transputers and Occam. However, the techniques presented here can be applied to other technologies with minor modifications. Experimental results have been provided in sufficient detail. Theoretical analysis, both for functionality and timing, have been provided for the relatively complex segments of the software design. In the following sub-sections, we present the major achievements of the present work.

4.1 Process Architecture

This is a very important aspect of Occam software. The processes in a Transputer must be designed in such a manner that messages can be received from and sent to other Transputers with minimum delay. Furthermore, processes should never be deadlocked. We have prevented deadlocks by using client-server protocols whenever two processes have two-way communications; in addition, we have ensured that there are no client-server cycles in the process graph. When time-critical processes are involved as, for
instance, a time-server, we ensure that they are designed to be pure servers so that they never block on communications. Processes related to Clock Synchronization are run in the high-priority (non-preemptive) mode [Tpr89]. These principles are illustrated in Chapters 10 and 11 of the dissertation. Process starvation is prevented by implementing a “fair” ALT. Whenever a process receives messages over links in an ALT construct, we use a fault-tolerant ALT. In the actual implementation, the “fairness” and “fault-tolerant” features are merged into a fair and fault-tolerant ALT (FFTALT).

4.2 Use of byte-stream messages
Although Occam2 provides channel protocols, we have decided not to use it. Instead, we have implemented every message as a byte array. This approach will be useful if we want to port the design to any other technology. Numerical data are marshalled and unmarshalled by RETYPing [Occ88] conversions. A unique identifier (uid) [Sch90] of a message is generated from the source, destination, and sequence number fields of the message. This is discussed in Chapter 10 of the dissertation.

4.3 A new failure mode
In this work, we have provided resilience to a difficult failure mode. This corresponds to the crash of a process in the midst of sending a message. If the receiving process uses an ALT construct to receive the message, then it will hang forever. We have provided a solution to this problem (Chapter 7).

4.4 Rollback Recovery
A scheme for Rollback Recovery on the Transputer has been presented. The technique is applicable to block-structured high-level languages in general. The saved state consists of local and global variables of the activation record in the run-time stack. Data type information is also saved for the purpose of relocation in case an item is an address rather than a data. When rollback is required, the run-time stack is reconstructed on a spare processor, address relocations are performed and the process is resumed. The resumption of the process requires the use of assembly language since
*Occam* cannot perform the dynamic creation of processes. The design is presented in Chapter 5 of the dissertation.

### 4.5 A high performance fault-tolerant pipeline

A fault-tolerant pipeline using two processors per pipeline stage has been presented [SDC92]. It is superior in performance and reliability in comparison to a traditional design that uses only spare links and graceful degradation. Every message transfer is implemented as an *Atomic Action*, i.e., an input message is first saved in a spare processor and then acknowledged to the sender. As a result, no *rollback recovery* is required. This approach can be helpful if the time required for *rollback recovery* is prohibitive. This topic is presented in Chapter 6 of the dissertation.

### 4.6 The fair and fault-tolerant ALT

The ALT construct of *Occam* follows the nondeterministic guarded input of Hoare's CSP. This is a fundamental concept for concurrent computation in the message passing paradigm and should not be viewed as technology-specific. However, when it comes to fault-tolerant programming, the Transputer implementation of the *Occam* ALT has a very serious loophole. First, if the sender stops while sending, i.e., it crashes before sending the complete message, the receiver executing in an ALT can hang forever. The root cause of this behaviour is built deep into the implementation of ALT. Second, the Transputer implementation of ALT is not “fair”, i.e., it is possible for the low-priority guards of the ALT to starve if the high-priority guards fire too often. We emphasize that even the ordinary ALT, as opposed to the PRI ALT, imposes a priority for the guards. This priority is the lexical order of the guards in the program text. We have provided solution (Chapter 7) to both of these problems by coding a construct called FFTALT (fair and fault-tolerant ALT) in assembly language.

### 4.7 Signed messages and authentication

In extending the fault model from permanent (crash) to *Byzantine faults*, we are faced with the problem of low connectivity of Transputer networks. This is caused by the low number (4) of links of the Transputer. The solution to this problem is to use
signed messages. This dissertation has adopted the well-known RSA cryptosystem. The theoretical basis for the method is well-established. However, its implementation involves many issues that have to be sorted out by careful judgement. A message is divided into blocks and each block has an associated signature. The selection of the block size and the signature size involves a trade-off between competing factors. If the size is too small, the secrecy of the cryptosystem is jeopardized. If the size is too large, computation time becomes prohibitive. After carefully weighing in these factors, we have come up with a design (Chapter 9) that should be found to be useful [SDB96]. The method is technology-independent and can be applied to processors other than the Transputer.

4.8 A comprehensive design of replicated servers

We have presented a comprehensive design of fault masking by replication [SDB96]. The client-server model of computation has been used and the servers are replicated. The fundamental problem is to ensure replica determinism, i.e., replicas should maintain identical states at about the same instant of time. The inspiration behind this technique is Schneider's state machine model [Sch190]. If the replicas process the same set of messages in the same order then replica determinism is achieved. This, in turn, is ensured by Atomic Broadcast of input messages and voting of output messages. The process architecture presented in this work handles this job elegantly. Experimental results have been provided in sufficient detail. Since the design is involved, we have presented the skeleton of the software. This topic is covered in Chapter 10 of the dissertation.

4.9 Clock Synchronization

This dissertation presents (Chapter 11) an implementation of Clock Synchronization [SDB97]. Again, the design of the process architecture is one of our contributions. Furthermore, we have provided experimental results on message transit times. This is important since the accuracy of synchronization depends upon $t_{del}$, the maximum time required to broadcast a message.
Another important aspect of the design is the execution of processes pertaining to Clock Synchronization in the high-priority mode of the Transputer. This is necessary in order to prevent unnecessary delay in processing messages. The processes pertaining to Clock Synchronization are time-critical. If they are scheduled late inspite of being ready, the accuracy of synchronization suffers.

In distributed programming, we frequently require an alarm clock service. In our work, this service is provided and integrated with the Clock Synchronization algorithm.

The skeleton of the source code has been provided. Apart from illustrating the design, the source code is necessary for obtaining theoretical estimates of message transfer times which is so important for determining the accuracy of synchronization.

4.10 Timing Analysis

The issue of timing analysis of a program is usually sidetracked since it is complicated. Reachability-based methods like Timed/Time Petri Nets are often advocated. However, any reachability analysis suffers from the State Explosion problem. Our approach [SDB97] towards timing analysis is based on Integer Linear Programming in which the constraints are (in)equalities pertaining to the automata models of the constituent processes. Since no state space enumeration is involved, the state explosion problem is not encountered. The power of this method lies in its ability to handle realistic systems that can have a large number of states. This topic is covered in Chapter 12 of the dissertation.

An immediate benefit of this analysis is the determination of timeout periods for fault-tolerant communications. Experimental estimates of timeout periods are not reliable.

Another benefit of timing analysis is that we can obtain an estimate of the message transfer delays in the Clock Synchronization algorithm. We have already observed that these delays determine the accuracy of synchronization.
Timing analysis is a painstaking affair. It requires us to determine the execution time of *Occam* statements. This can be done from a disassembly of the source code. However, the memory map of the object program is also important. A memory location can be accessed faster when it resides in the internal (on-chip) RAM of the Transputer.

In order to obtain upper bounds on execution time that are close to actual, we need to introduce several constraints that are implementation-specific, that is, they are determined by close inspection of the automata models of the processes. This dissertation has shown how additional constraints can be defined to obtain realistic upper bounds in the Clock Synchronization problem (Chapter 13).

The technique can be extended to handle logical constraints by translating them into nonlinear algebraic constraints.

### 4.11 A basis for Software Fault Tolerance

An implementation of Software Fault Tolerance based on *Recovery Blocks* can succeed only if a *variant* terminates. If a *variant* does not terminate or requires too long a time for termination, then it has to be killed. Unfortunately, *Occam* does not permit us to kill a process unless it terminates by itself. This dissertation presents (Chapter 14) a method of achieving the goal by using the assembly language [Tpr88] of the Transputer.

### References


