COMPUTING LABORATORY

Replication within Atomic Actions and Conversations: A Case Study in Fault-Tolerance Duality

L.V. Mancini and S.K. Shrivastava

TECHNICAL REPORT SERIES
No 281 April, 1989
Replication within Atomic Actions and Conversations: A Case Study in Fault-Tolerance Duality

L.V. Mancini and S.K. Shrivastava

Abstract

We have recently proposed a duality mapping for fault-tolerant system structures. Two canonical models of distributed fault-tolerant systems have been constructed and shown to be duals of each other. One model incorporates objects and atomic actions as the entities for program construction while the second model employs communicating processes with conversations. As a consequence of the duality, techniques and mechanisms which have been developed within the domain of just one of the models can be mapped and applied to the other model. The paper illustrates this point by mapping some well-known object replication techniques developed within the context of object and actions model to the communicating process model thereby revealing some interesting process replication techniques.


Series Editor: M.J. Elphick

© 1989 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

MANCINI, Luigi Vincenzo

Replication within atomic actions and conversations: a case study in fault-tolerance duality. [By] L.V. Mancini and S.K. Shrivastava

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1989.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 281)

Added entries

SHRIVASTAVA, Santosh Kumar

UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 281

Abstract

We have recently proposed a duality mapping for fault-tolerant system structures. Two canonical models of distributed fault-tolerant systems have been constructed and shown to be duals of each other. One model incorporates objects and atomic actions as the entities for program construction while the second model employs communicating processes with conversations. As a consequence of the duality, techniques and mechanisms which have been developed within the domain of just one of the models can be mapped and applied to the other model. The paper illustrates this point by mapping some well-known object replication techniques developed within the context of object and actions model to the communicating process model thereby revealing some interesting process replication techniques.

About the authors

Mr. Mancini was at the Computing Laboratory from May 1985 to February 1989 as a Research Associate.

Professor Shrivastava joined the Computing Laboratory in August, 1975, where he is a Professor.

Suggested keywords

ATOMIC ACTIONS
COMMUNICATING PROCESSES
CONVERSATIONS
DISTRIBUTED SYSTEMS
FAULT TOLERANCE
REPLICATION

Suggested classmarks (primary classmark underlined)

Dewey (18th): 001.64404
U.D.C. 519.687
Replication within Atomic Actions and Conversations: A Case Study in Fault-Tolerance Duality

L.V. Mancini
Dipartimento di Informatica
Università di Pisa
Pisa, Italy

and

S.K. Shrivastava
Computing Laboratory
The University
Newcastle upon Tyne, UK.

ABSTRACT

We have recently proposed a duality mapping for fault-tolerant system structures. Two canonical models of distributed fault-tolerant systems have been constructed and shown to be duals of each other. One model incorporates objects and atomic actions as the entities for program construction while the second model employs communicating processes with conversations. As a consequence of the duality, techniques and mechanisms which have been developed within the domain of just one of the models can be mapped and applied to the other model. The paper illustrates this point by mapping some well-known object replication techniques developed within the context of object and actions model to the communicating process model thereby revealing some interesting process replication techniques.

Keywords: fault-tolerance, atomic actions, conversations, replication, distributed systems, communicating processes.

1. Introduction

One widely used technique of introducing fault-tolerance - particularly in distributed systems - is based on the use of atomic actions (atomic transactions) for structuring programs. An atomic action possesses the properties of serializability, failure atomicity and permanence of effect. Atomic actions operate on objects (instances of abstract data types). The class of applications where such an object-action model (OM) has found usage include transaction processing applications in office information, airline reservation and database systems. A number of other applications - typically concerned with process control - are structured as concurrent processes communicating via messages. Some examples are process control, avionics and telephone switching systems. Fault-tolerance in such systems is introduced through a controlled use of checkpoints (conversations) by processes. We will refer to this way of structuring an application as employing the process-conversation model (PM).

In a recent paper [SMR], we have argued that the OM and PM approaches to the provision of fault-tolerance are duals of each other. As a result of this observation, we can state that there is no inherent reason for favouring one approach over the other; rather the choice is largely dictated by the architectural features of the underlying layer. Indeed, we would now claim that the differences between the two approaches are basically a matter of viewpoint and terminology. However, one finds that fault-tolerant systems are constructed and described using the concepts and terminology applicable to just one of the two models, with no apparent realization of the potential relevance of systems and the literature describing them which make use of the other model. By realizing the equivalence between the two approaches to fault-tolerance, it is possible now to map techniques and mechanisms which happen to have been developed within the domain of just one of the models to the other model.

For example, there is a large body of literature on the topic of replicated data management for increasing the availability of data, see [BG,BJRE,ES] for a representative sample. A number of algorithms have been developed and their correctness properties have been stated and proved using rigorous mathematical techniques. However, replication techniques for communicating processes have not been studied and analyzed to the same degree of detail. Consider the following problem in process replication. Given is a set of processes which interact with each other purely through message passing. These processes make use of checkpoints for recovery from failures. It has been decided to replicate a subset of these processes for increasing availability. Design algorithms and explore possible optimizations for replica management such that replicated processes behave consistently (that is, they appear to behave like a single process). At first this problem may seem daunting. However, in this paper we show that interesting techniques for replicated process management can be developed easily from replicated data management techniques and applied to process control systems that have been developed using the PM.

This paper is structured as follows. In Section Two, the two models are briefly described. The duality mapping is then established in Section Three. Section Four then describes some object replication techniques, which are then applied for developing process replication techniques in Section Five. Section Six considers some existing process replication techniques and shows that they are special cases of the process replication techniques obtained through the duality mappings. We also show how our approach enables a clearer understanding of process replication techniques.

The main technical contributions of this paper are twofold: (1) it draws attention of the reader to the potential benefits obtained from the duality mapping by performing a detailed case study in replica management; and (2) as a side-effect of this study, new replicated process management techniques of practical importance have emerged.
2. Fault-tolerant system models

2.1. Object-action Model

Objects are instances of abstract data types. An object encapsulates some data and provides a set of operations for manipulating the data; these operations being the means of object manipulation. In most object-based fault-tolerant systems (see [Ne, LS, Sh, Sv, Sp]) for a representative sample, an operation is performed by invoking an object with a remote procedure call (RPC), which passes value parameters to the object and returns the results of the operation to the caller. Programs which operate on objects are executed as atomic actions with the properties of (i) serializability, (ii) failure atomicity and (iii) permanence of effect [Gr]. The first property ensures that concurrent executions of programs are free from interference (i.e., concurrent execution can be shown to be equivalent to some serial order of execution [BR, Es]). The second property ensures that a computation can either be terminated normally (committed), producing the intended results or be aborted, producing no results. This property is obtained by appropriate use of backward error recovery, which is invoked whenever a failure that cannot be masked occurs. Typical failures causing an action to be aborted are node crashes and communication failures such as persistent loss of messages. It is reasonable to assume that once a computation commits, the results produced are not destroyed by subsequent node crashes. This is the third property - permanence of effect - which ensures that state changes produced are recorded on stable storage which can survive node crashes with high probability. A two-phase commit protocol is required during the termination of an action to ensure that either all the objects updated within the action have their new states recorded on stable storage (committed termination), or no updates get recorded (aborted termination).

A variety of concurrency control techniques for atomic actions to enforce the serializability property have been reported in the literature. A very simple and widely used approach is to regard all operations on objects to be of type read or write, which must follow the well-known locking rule permitting concurrent reads but only exclusive writes. In a classic paper [Es], Eswaran et al. proved that actions must follow a two-phase locking policy. During the first phase, termed the growing phase, a computation can acquire locks on objects but not release them. The tail end of the computation constitutes the shrinking phase, during which time held locks can be released but no locks can be acquired.

Dependencies between actions can be illustrated using action diagrams such as the one depicted in Figure 1 (the notation employed is based on that used by Davies [Da]). According to Figure 1, a directed arc from an action (e.g. \( A_1 \)) to some other action (e.g. \( A_2 \)) indicates that \( A_2 \) uses objects released by \( A_1 \). Optionally, an arc can be labelled naming the objects used by the action. For example, \( A_2 \) uses objects \( x \) and \( y \) released by \( A_1 \), and \( A_4 \) uses object \( y \) which in turn has been released by \( A_2 \).

One of the most important aspects of the OM from our point of view is the fact that objects and actions are the two primary entities from which an application program is constructed. Any implementation of actions and objects may well require a suitable mapping on to processes (clients and servers) for carrying out the required functions, since most present day operating systems support processes as the entities to be manipulated. However, the role played by processes is hidden at the application level. Similarly, there is no explicit use of message passing between objects, since RPCs hide the details of message interactions between clients and servers. For example, in the Argus programming system [LS], the implementation of guardians (objects) requires a number of processes for receiving and executing calls from clients - but processes are not visible entities to be used explicitly by an application program.

![Figure 1: Action diagram.](image)

2.2. Process-conversation Model

In contrast to the OM, where processes and messages play at most a secondary role, the PM has them as the primary entities for structuring programs. An application is structured out of a number of concurrent and interacting processes.

The PM will be assumed to have the following characteristics: (1) processes do not share memory, at least explicitly, and communicate via messages sent over the underlying communication medium; (2) appropriate communication protocols ensure that processes can send messages reliably such that they reach their intended destinations uncorrupted and in the sent order; (3) a process can take a checkpoint to save its current state on some reliable storage medium (stable storage). If a process fails, it is rolled back to its latest checkpoint.

In a system of interacting processes, the recovery of one process to its checkpoint can create an inconsistent global state, unless some other relevant processes are rolled back as well. This leads to the notion of a consistent set of checkpoints or a recovery line (RLT). A set of checkpoints, one from each process, is consistent if the saved states form a consistent global state. Figure 2 illustrates the notions of consistent and inconsistent sets of checkpoints where open square brackets on process axes indicate checkpoints and sloping arrows represent messages. Suppose process \( p \) fails at the point indicated by the vertical arrow and is rolled back to its latest checkpoint. The global state of the system as represented by set of checkpoints on the cut \( C_p \) is inconsistent since the checkpoint of \( r \) has recorded a message not sent by \( p \); the set of checkpoints on recovery line \( C_r \) is however consistent. Thus failure of \( p \) can cause a cascade rollback of all the four processes - this is the domino effect mentioned in [Ra]. The dynamic determination of a recovery line is a surprisingly hard task; the reader should consult [KT] for a clear exposition.

The domino effect can be avoided if processes coordinate the checkpointing of their states. A well-known scheme of coordinated checkpoints is the conversation scheme [Ki, Ra]. The set of processes which participate in a conversation may communicate freely between each other.
3. Duality Mappings

The duality between the OM and PM can be established by considering objects and actions to be the duals of processes and conversations respectively. Further, object invocations can be considered duals of message interactions [LN]. A given conversation diagram (e.g. Figure 4.a), can be translated into an action diagram quite simply (e.g. Figure 4.b) by replacing each conversation C with a corresponding action A, and adding an arrow from A to A' if C and C' have at least one process in common and that process enters C after exiting from C'. An arc from one action to the other is labelled with the objects representing the processes common to the corresponding conversations. A reverse mapping is possible by replacing distinct objects named in the action diagram by processes. An action is replaced by the corresponding conversation, with the set of processes in the conversation determined by the set of objects named in all the incoming and outgoing arcs of the action.

A summary of the various characteristics of the two models for which duality has been established is presented in Table 1. For detailed arguments supporting the duality mapping summarized in Table 1 the reader's attention is drawn to [SMR].

By realizing the duality between the two approaches to fault-tolerance, it is possible now to map techniques developed within the context of one model to the other model. For example, in [SMR] we have shown how optimization techniques for read only actions can be applied to conversations. In the rest of this paper we will develop replicated process management techniques from replicated object management techniques. More precisely, we consider the following problem. Given is a set of concurrent processes interacting via conversations. We assume that a subset of these processes (server processes) provide various services to the remaining processes (client processes) and these services have to be made available despite a bounded number of server node crashes. Thus it is necessary to replicate servers on different nodes. Algorithms for implementing such a system where client processes interact with replicated servers within the framework of conversations have not been studied before. Here we will show how such algorithms can be developed easily by applying the duality mapping to object replication techniques which have been studied extensively.

4. A review of object replication techniques

In a system where nodes never fail, replicated objects can easily be managed. It is sufficient to perform any operation of an object x on all copies of x. Unfortunately, this approach is impractical in systems where nodes can fail and recover. For example, this approach requires that each operation be performed on all copies of x, even if some have failed. Since there will be times when some copies of x are unavailable due to node failures, the system will not always be able to perform the required operation on all copies of x at the time it receives the request. If the system were to adhere to this approach, it would have to delay processing the operation until it could access all copies of x. Such a delay is obviously unsatisfactory. If any copy of x becomes unavailable, then no action that invokes x can execute to completion. The more the copies of x, the higher the probability that one of them is unavailable. In this case, replication actually reduces the availability of objects.
Several techniques have been proposed to manage replicated objects. To be specific we will consider two well-known techniques: the available copies and the primary copy schemes. In the following we will briefly describe these techniques. We assume that nodes fall in a fail-silent manner (that is, a node is either operational or down), and that all operational nodes can communicate with each other. Therefore, each operational node can independently determine which nodes are down, simply by attempting to communicate with them. If a node does not respond to a message within some timeout period, then it is assumed to be down.

4.1. The available copies scheme

As stated earlier, an object provides a set of operations, some of which can modify the state of the object (e.g., push and pop operations of a stack object). Initially, we will assume that a node does not recover after a failure.

The available copies scheme does not require an action to update all copies of each object. An action should send every operation request to all of the copies that it can, but it may ignore any copies that are down. After sending an operation request to all copies of object \( x \), an action may receive rejections from some nodes (if the operation is conflicting with some other action), positive response from others (meaning the operation has been accepted and performed), and no response from others (those that have failed). Operation requests for which no responses are received are called missing. If any rejection is received or if all operation requests to \( x \)'s copies are missing, then the whole operation is rejected and the action must abort. Otherwise, the whole operation is successful. Since a fail-silent behaviour of the nodes is assumed, anyone of the positive responses can be taken as the result of the operation invocation.

4.1.1. Recovery

So far we have assumed that a failed node does not recover. This limitation can be removed by providing a reconfiguration mechanism, in order to support the recovery of a copy from a failure, and in general the creation and removal of copies of an object. To achieve reconfiguration after a node crash, the set of nodes holding the available copies of an object must be dynamically established.

A solution is to employ directories to record for each object \( x \) the set of \( x \)'s copies that are available. Like any other object, a directory may be replicated, that is, it may be implemented as a set of directory copies at different nodes. In the following discussion of a scheme to support dynamic reconfiguration, we assume that there is a fixed set of copies for each directory, known to every node. That is, new directory copies are never created - the method for creating new object copies can be easily extended to create new directory copies.

Directories recording available object copies are manipulated by two special actions, Join for creating new object copies, and Delete for deleting unavailable object copies. When a node \( N \) containing a copy of \( x \), say \( x_N \), recovers from a failure, it runs an action Join\( (x_N) \). Join\( (x_N) \) brings the state of \( x_N \) up-to-date by: (1) finding a directory copy \( d \) listing the set of copies of \( x \); (2) reading \( d \) to find an available copy of \( x \), say \( x_D \); (3) copying \( x_D \)'s state into \( x_N \); (4) declaring \( x_N \) to be available by making an entry for \( x_N \) in each available copy of the directory \( d \).

When a node fails, some client that tries to invoke an object operation at that node observes the failure. Then, it runs a Delete action for each copy stored at the failed node. Delete declares the relevant object copy to be unavailable by removing the entry for this copy from every available copy of the directory.

Because objects are replicated, it is possible to eliminate the need for stable storage for recording their states, since a
rerecovering node can use a Join action for acquiring up-to-date versions of objects.

4.1.2. Read optimization

An optimization is possible with the available copies scheme if one takes advantage of the semantics of the operations. The operations executed by each object may be partitioned into two broad classes: the class write of the operations that modify the state of the object, and the class read which incorporates operations that do not alter the state of the object. The operations of class read then need not be invoked on all available copies of an object but just on one, while the operation requests of class write need to be sent to all available copies of an object. For example, in the case of the stack object, the operation top, which returns the value at the top of the stack without modifying it, can be invoked just on any available copy of the stack. The operations push and pop are of class write, and must be invoked on all available copies of the stack.

A distributed two-phase locking scheme can be employed for concurrency control. The following rule is required: whenever an operation of class read is invoked on an object, the action must first acquire a read lock (if not already acquired) on any available copy of the object; for a write operation, the action must first acquire write locks (if not already acquired) on all the available copies of the object.

When this form of read optimization is used, a validation protocol is required to prevent inconsistencies in the presence of failures (see [BHG] for details). To understand the need for such a protocol, consider the case of two replicated objects x and y, with copies x₁, x₂ and y₁, y₂ respectively. An action, A₁, reads a value from a copy of x, and writes it to all the available copies of y. While another action, A₂, performs a similar update from y to x. Both the actions simultaneously read from objects (A₁ from x₁ and A₂ from y₂) after which two failures occur making x₁ and y₁ unavailable. The actions then update the available copies of the objects. Clearly, this not a serializable behaviour (updates on x₁ and y₁ have been missed). The validation protocol is intended to detect such occurrences. An action's validation protocol starts during commit processing, by which time the action's operations on copies have been acknowledged or timed out. At that time the action knows all the copies it has actually accessed. The validation protocol checks that all copies that were available (unavailable) are still available (unavailable), else the action is aborted. In the example considered above, both A₁ and A₂ will be aborted.

The read optimization with validation protocol scheme suffers from the limitation that in certain situations an action has to be aborted if a failure occurs. As we have discussed previously, without read optimization, the completion of an action can be guaranteed in the presence of a specified number of node failures, by distributing operation requests to all available copies. Distributing information about read requests, and in particular about read locks, may seem unreasonably expensive. However, in [Bjarne], a scheme for lazy propagation of read locks is mentioned which guarantees that read lock information is delivered to a node before any action that requires that information is executed.

The Join and Delete actions discussed previously are still required to support recovery with read optimization. In the subsequent discussions, we will use the term copy scheme to refer to the particular scheme without read optimization.

4.2. The primary copy scheme

With the primary copy scheme, executing actions use a non-replicated view of the system. That is, for each object that the actions access, the operations are carried out on the same copy of the object, called the primary copy. The distribution of the operations to other backup copies is delayed until the action has terminated and is ready to commit. It is necessary therefore to maintain an intentions list of deferred operations. During the termination of an action, the appropriate portion of the intentions list has to be sent to each node that contains backup copies of the relevant objects. Alternatively, the primary copy of an object can also send its new state in place of the intentions list. In any case, the information can be piggybacked on the prepare messages of the first phase of the commit protocol. If the primary copy fails then the executing action is aborted and can be resubmitted to use a different copy that will take over as primary.

In order to support recovery after a failure of a primary copy, it is necessary to elect a backup copy as the new primary. A simple scheme, that does not involve additional communication, is to fix a static ordering of the copies. An alternative is to run a consensus protocol among the backup copies - the election of the new primary copy can be based on the current load on the system.

With the primary copy scheme, it is possible to put all deferred operation requests destined for the same node in a single message. This tends to minimize the number of messages required to execute an action. By contrast, with available copies scheme, the action sends operation requests to replicated copies while it executes. Thus the available copies scheme tends to use more messages than the primary copy scheme. Another advantage of the primary copy scheme is that aborts often cost less compared with the available copies scheme. In the available copies scheme, when an action aborts, it is likely that many of the action's operations have already been distributed to replicated copies. Not only are these operations wasted, but they must also be undone. With primary copy, the distribution of those operations are delayed till termination time making abortion cheaper. Fast aborts in the primary copy scheme are at the expense of commits which can be more time consuming than in the available copies scheme. This is because during the first phase of the commit protocol a node may be asked to process a potentially large number of deferred operations on backups.

With the primary copy scheme, read optimization is possible - the intentions lists of only write operations need be distributed to backup copies when the executing action is ready to commit.

The most important aspects of the object replication techniques relevant to this paper are summarized below. The pure available copies scheme provides k-object-resiliency, meaning that out of k copies of an object, all the k copies have to become unavailable before the action using it is forced to abort. With read optimization, k-object-resiliency is not always reachable; this is the price paid for obtaining higher efficiency. The primary copy scheme does not provide k-object-resiliency in the sense mentioned above; the executing action has to be aborted if the primary fails. The action can be resubmitted once a secondary is elected to be the primary.
5. Process replication techniques

Process replication techniques take advantage of the existence of multiple nodes by replicating critical processes on two or more nodes.

A terminology commonly used for classifying the redundancy employed in PM is to differentiate active replication from passive replication. With an active replication scheme, a given computation is executed simultaneously on a number of processes while with a passive replication scheme, if the process running the computation fails then a designated backup process takes over. Not surprisingly, active replication techniques correspond to the available copy schemes and passive replication techniques to the primary copy schemes.

The duality mapping between object and process replication schemes is shown in Table 2.

5.1. The available processes scheme

The dual of the available copy scheme results in an approach where replicated processes behave like a single process. Interactions with a replicated process implies interactions with all of its replicas. A copy of a request to a replicated process is sent to all replicas, and all replicas execute each request. In case a reply is required, all replicas generate replies; only the first reply received is considered, and the others are discarded (since the replies from all working replicas should be identical under the fail-silent assumption on processor behavior). A reconfiguration strategy for the available processes scheme can be designed by adopting the directory-based scheme. Note that replicated processes need not record their checkpoints on stable storage (see Section 4.1.1).

5.1.1. Read optimization

Read optimization can be achieved in PM using the approach employed in OM. Assume that processes receive message via message ports which are data structures capable of holding messages of a certain class. Message ports can be of class read, capable of receiving messages whose processing does not alter the state of the receiver process, and write, capable of receiving messages whose processing can alter the state of the process. A message intended for a read port of a process need not be sent to all available copies of that process.

Read optimization in PM results in weakening of the conversation rule. Instead of having all the copies of a replicated process participating in only one conversation at a time, a replicated process can take part in more than one conversation, if only requests of the class read are being served.

This optimization may lead to problems of consistency similar to those encountered in OM, as can be appreciated by considering the process diagram of Figure 5. Here a system with non-replicated processes p and q and replicated processes x (copies xA, xB) and y (copies yC, yD) is considered. Suppose that within the conversation C1, p reads the state of x and updates the state of y (denoted by r_x and w_y, respectively) and that, within the conversation C2, q reads the state of y and updates the state of x (denoted by r_y and w_x).

Conversation C1 begins by reading from xA, and conversation C2 begins by reading from yD. After p and q complete their reads, processes xA and yD fail. Then p and q perform their writes. Since yD is down, yC is the only available copy of y. So C1's w_y is invoked only on yC. Similarly, since xA is down, C2's w_x is invoked only on xB.

The execution above does not violate the conversation rule for read optimization (just as in OM, the two-phase locking rule will not be violated). However, this execution is not equivalent to any serial execution. A serial execution of C1 and C2 on a non-replicated system would have either C1 reading the value of x written by C2, or C2 reading the value of y written by C1. In the example neither conversation reads the data written by the other.

The dual of the validation protocol is required at the end of conversations to ensure correctness. The aim of the validation protocol in the PM is to make sure that all processes found available (unavailable) during the execution of a conversation are still available (unavailable).

5.2. The primary process scheme

With the primary process scheme, the same copy of a replicated process, called primary process, takes part in conversations. A replicated process is provided with backup processes on different nodes (in particular, just one backup process might be employed). Request messages are sent to the primary process, which handles the requests. The distribution of requests to other backup processes is delayed until the end of the conversations. At that time, the primary copy sends the list of requests served during the conversation to the other backup processes. Alternatively, the primary process can send a checkpoint of its new state to the backup processes. In the event of a primary process failure, the executing conversation is rolled back to the beginning and restarted with a backup process which takes over and becomes primary. The dual of the election scheme mentioned earlier will be required to select the new primary process.

6. Some existing process replication schemes

Schemes following the available processes approach have been proposed in the literature independent of the data replication techniques [CGR, Sc]. In [Sc], a general approach is proposed for coordinating copies of replicated processes so that each copy executes the same sequence of process interactions. This is achieved by the implementation of the abstractions of agreement and order. A similar approach is adopted in [CGR], where client processes send request messages to all the copies of a replicated server, and a distributed consensus protocol for every request is employed to enable each copy of the server to process requests from different clients in the same order. However, these papers do not describe how aborts are performed and domino effect avoided if a client rolls back, further, neither reconfiguration mechanisms have been proposed, nor any concurrency control techniques described in [CGR].

The available process scheme discussed in Section 5.1 provides a complete solution to these problems. Conversations can be exploited for reducing the frequency of the distributed consensus protocol. In particular, once a replicated process has agreed to participate in a conversation with a client, request messages can be served without further agreement till the end of the conversation.

A technique resembling the primary copy scheme has been described by Borg et al. [BBG]. Whenever anyone sends a message to a process, the same message is forwarded to the backup process. The system ensures that both the processes
<table>
<thead>
<tr>
<th></th>
<th>Active replication</th>
<th></th>
<th>Passive replication</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pure form</td>
<td>recovery</td>
<td>read optimization</td>
<td>pure form</td>
</tr>
<tr>
<td>Object replication</td>
<td>available copy</td>
<td>available copy + join</td>
<td>available copy + validation</td>
<td>primary copy</td>
</tr>
<tr>
<td>Process replication</td>
<td>available process</td>
<td>available process + join</td>
<td>available process + validation</td>
<td>primary process</td>
</tr>
</tbody>
</table>

Table 2: Object and process replication.

Figure 5: Incorrect execution.

cannot continue running until it has been verified that both have correctly received the message. Thus, if one process crashes because of any hardware fault, the other one can continue. Furthermore, the remaining process can then clone itself, making a new backup to maintain the fault-tolerance capability.

One possible disadvantage of [BBG] is that, if processes exchange messages at a high rate, a considerable amount of time may go into keeping the processes synchronized at each exchange. An alternative scheme is discussed by Powell and Presotto in [PP]. The system described in that paper puts little additional load on the processes being backed up. All messages sent on the network are recorded by a special recorder process. From time to time, each process checkpoints itself onto a remote disk. If a process crashes, recovery is done by sending the most recent checkpoint to an idle process and telling it to start running. The recorder process then sends to the newly created process all the messages that the original process received between the checkpoint and the crash. The primary process scheme developed here to work in conjunction with conversations reduces the need for frequent message exchanges by employing the dual of the deferred update technique used in the primary copy scheme.

7. Summary and conclusions

Given that a large class of data replication techniques has been proposed in the database literature, it seems natural to enquire whether these techniques can be adapted to process control systems. Assuming that a set of processes interact via message passing and use checkpoints for recovery and that the servers are to be replicated for increased availability, can we use existing data replication techniques for deriving process replication techniques? In the present paper, we have performed such a transformation by making use of the duality mapping between fault-tolerant structures proposed in [SMR]. The duality mapping was first described and then two data replication techniques were presented. These were then used for developing available process and primary process schemes for processes using conversations. Some existing process replication techniques were also described to show that they are special cases of the schemes developed here.
In this paper we have considered the case of replicated servers only; it is also possible to replicate both clients and servers, e.g. (Co). Such extensions can easily be developed using the methodology presented here. We hope that in addition to the process replication schemes, the paper has also contributed by demonstrating the usefulness of the fault-tolerance duality concept.

Acknowledgements

The duality concept has been developed jointly with Brian Randell. This work has been supported in part by grants from the U.K. Ministry of Defence and the Science and Engineering Research Council.

References


