

An Integration of the Primary-Shadow TMO Replication Scheme with a Supervisor-based Network Surveillance Scheme and its Recovery Time Bound Analysis

K. H. (Kane) Kim * and Chittur Subbaraman

Dept. of Electrical & Computer Engineering
University of California
Irvine, CA, 92697, U.S.A.
kane@ece.uci.edu (*: contact author)

Abstract: The *time-triggered message-triggered object* (TMO) scheme was formulated a few years ago as a major extension of the conventional object structuring schemes with the idealistic goal of facilitating general-form design and timeliness-guaranteed design of complex real-time application systems. Recently, as a new scheme for realizing TMO-structured distributed and parallel computer systems capable of both hardware and software fault tolerance, we have formulated and demonstrated the *primary-shadow TMO replication* (PSTR) scheme. An important new extensions of the PSTR scheme discussed in this paper is an integration of the PSTR scheme and a network surveillance (NS) scheme. This extension results in a significant improvement in the fault coverage and recovery time bound achieved. The NS scheme adopted is a recently developed scheme effective in a wide range of point-to-point networks and it is called the *supervisor-based NS* (SNS) scheme. The integration of the PSTR scheme and the SNS scheme is called the *PSTR/SNS* scheme. The recovery time bound of the PSTR/SNS scheme is analyzed on the basis of an implementation model that can be easily adapted to various commercial operating system kernels.

Keywords: Time-triggered Message-triggered Object, TMO, Primary-shadow TMO replication, PSTR, network surveillance, NS, SNS, point-to-point networks, real-time systems, recovery time bound.

1. Introduction

Large-scale computer-based systems in safety-critical applications must meet hard deadlines for certain output actions. The failure to meet the deadlines could have disastrous consequences. This dictates that the run-time behavior of the computer systems must be highly predictable and the rigorously established real-time fault tolerance technologies should be used in constructing such systems. Moreover, such systems should be structured in a modular and systematic manner in order to make their maintenance and expansion to be manageable [Bas96, Hec91, Mos96].

A few years ago, we established the *time-triggered message-triggered object* (TMO) structuring scheme [Kim94b, Kim97a], a major extension of conventional

object structuring schemes, with the idealistic goal of facilitating general-form design and timeliness-guaranteed design of complex real-time application systems. Subsequently a scheme for realizing TMO-structured distributed and parallel computer systems capable of both hardware and software fault tolerance was developed. It is called the *primary-shadow TMO replication* (PSTR) scheme and facilitates structuring systems as networks of both non-redundant TMO's and active TMO replicas. It is a result of incorporating the basic underlying principle of the well established DRB/PSP technique [Kim94a], called the *primary-shadow active replication principle*, into the TMO structuring scheme [Kim97a]. The potential of the PSTR scheme has been demonstrated through incorporation of the scheme into realistic hard real-time application system prototypes.

Among the most important extensions that have not been developed fully are the integrations of the PSTR scheme and *network surveillance* (NS) schemes. NS schemes facilitate the fast learning by each interested fault-free node in the system of the faults or repair completion events occurring in other parts of the system and also facilitate fast reconfiguration [Kim94a]. We recently developed a semi-centralized real-time NS scheme effective in a wide range of point-to-point networks and it is called the *supervisor-based network surveillance* (SNS) scheme [Kim97b].

In this paper, we present an integration of the PSTR scheme with the SNS scheme, called the *PSTR/SNS* scheme. This scheme is a significant improvement over the previous versions of the PSTR scheme in terms of the fault coverage and recovery time bound achieved in the systems based on point-to-point networks. Moreover, this scheme is one of the few software approaches for time-bounded tolerance of both hardware and software faults that can be used for point-to-point network architectures. We also present a concrete modular implementation model of the PSTR/SNS scheme. As a validation experiment, the execution support for the scheme was incorporated into a prototype implementation of the DREAM kernel which is a model of a timeliness-guaranteed operating system kernel developed at the University of California, Irvine [Kim97a].

We also analyze the performance of the PSTR/SNS scheme based on the presented implementation model for various cases of faults to obtain some tight *recovery time bounds*. In safety-critical real-time distributed computing applications, the recovery time bound is a measure of critical importance, but yet such analyses had been done scarcely until a few years ago.

The paper starts in Section 2 with a brief review of the essence of the TMO scheme and the basic design and operating principles of the PSTR scheme. Section 3 presents the point-to-point network based system architecture model adopted and also the fault source model along with the assumed failure frequency bounds. The SNS scheme developed earlier is reviewed in Section 4. Thus, Sections 2 and 4 give an overview of the basic concepts and were included in order to make this paper self-contained to some extent. Section 5 then presents the basic principles and a modular implementation model of the PSTR/SNS scheme. A recovery time bound analysis of the PSTR/SNS scheme is given in Section 6 and the paper concludes in Section 7.

2. The TMO and the PSTR structuring schemes

2.1 The TMO structuring scheme

The concrete syntactic structure and the associated precise execution semantics of the TMO, formerly called the RTO.k object, was developed in early 90's [Kim94b, Kim97a]. The basic structure of a TMO is depicted in Figure 1. It is an extension of the conventional basic object model(s) and most important and unique extensions can be summarized as follows:

(a) *Time-triggered (TT-) methods*, also called the *spontaneous methods* (SpM's), and *clearly* separated from the conventional *service methods* (SvM's) triggered by messages from clients:

SpM executions are triggered as the real-time clock reaches some values specified at design time. Actions to be taken at real times *which can be determined at the design time* can appear only in SpM's. Therefore, SpM's represent an absolute time domain whereas SvM's represent a relative time domain. Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section.

(b) *Basic concurrency constraint* (BCC):

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of TMOs, activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place. To be exact, when a

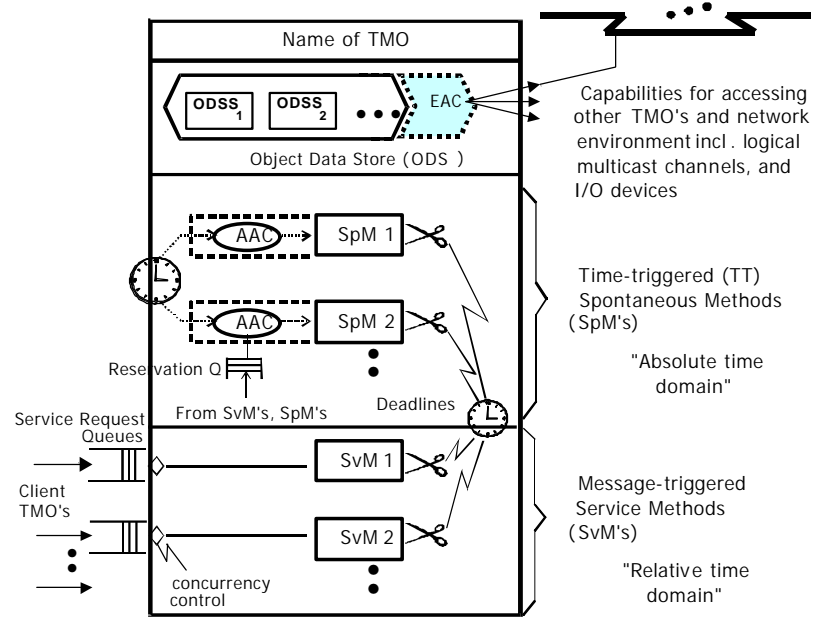


Figure 1. Structure of the TMO (adapted from [Kim97a])

message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data store (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM).

(c) (option) *Ordered isolation* (OI) rule:

We recently incorporated the following rule that also helps in easing the design-time guaranteeing of timely services. This rule called the *ordered isolation rule* is Unlike the above two features, this is an optional feature of the TMO structuring scheme. The term *initiation timestamp* or *I-timestamp* is used with the following meaning. In the case of an SvM execution, the *I-timestamp* is defined as the record of the time instant at which the execution engine initiated the SvM execution after receiving the client request for the SvM execution and ensuring that the SvM execution can be initiated without violating the BCC and other execution rules. In the case of an SpM execution, the *I-timestamp* is defined as the record of the time instant at which the SpM execution was initiated according to the AAC specification of the SpM. Also, a A segment of the object data store (ODS), called an ODS segment (ODSS), is a basic unit of data storage which can be reserved for exclusive access by a method of a TMO. The OI rule has two parts:
(OI-1) A method execution with an older initiation timestamp must not be waiting for the release of an ODSS held by a method execution with a younger initiation timestamp.
(OI-2) A method execution may never be rolled back due to an ODSS conflict.

2.2 The PSTR scheme

As mentioned in Section 1, the PSTR scheme is a result of incorporating the basic underlying principle of the DRB/PSP scheme [Kim94a], called the *primary-shadow active replication principle*, into the TMO structuring scheme. Figure 2 shows the basic operational rules of the PSTR scheme. The figure shows a fault-free execution cycle of an SvM in a primary TMO and the corresponding execution in the shadow partner TMO where the two partner TMO's are hosted on two different nodes connected to a local area network (LAN). Both the execution of the SvM in the primary TMO, for short, the *primary SvM execution*, and the *shadow SvM execution* use the same version P_2 of the computational procedure.

Node A hosts the initial *primary* TMO and node B hosts the initial *shadow* TMO. Both nodes receive the same client request from the network in their appropriate service request queues (SRQ's). After the client request arrives, the *initiation condition check* (ICC) mechanism of the execution engine in the primary node periodically checks whether the required primary SvM execution can be initiated without violating either the BCC or the ordered isolation rule. Upon finding such an appropriate

time, the ICC initiates the primary SvM execution. As soon as the primary SvM execution is initiated, it involves saving the client request message into some *log cache*, the storage which can survive at least as long as the node hosting the TMO does. The reason for saving this message will be evident later in this section. The next step in the primary SvM execution is to inform the shadow TMO of the *ID of the client request* that the former is going to process. When the ICC in the shadow node receives the client request ID from the primary SvM execution, it starts periodically checking whether the shadow SvM execution can be initiated without violating either the BCC or the ordered isolation rule. Upon finding such an appropriate time, the shadow SvM is initiated by the ICC to process the client request corresponding to the received ID. The shadow SvM execution then starts with saving the client request message into its log cache.

The primary and shadow SvM executions process the client request and invoke their *self-checking* independently and concurrently by using the same acceptance test (AT) routine. This routine can be implemented entirely in software. Some errors may be detected by the mechanisms in the execution engine before execution of the AT but such error detection can be treated in the same manner as the failure in the AT is. Since the primary and the shadow SvM executions both pass the test, they independently save all the method computational results into their log cache. Upon saving the results the primary SvM execution informs the shadow SvM execution of the success of the AT. The former then proceeds to perform all of its output actions (which may include *external output actions* as well as *internal output actions* such as the release of ODSS locks after updates). The primary SvM may seek the help of a *network surveillance manager* to confirm the successful execution of its external output actions. Once the primary SvM execution confirms the successful delivery of its output, it sends an *output success notice* to the shadow SvM execution. On the side of the shadow SvM execution, after saving all of its computational results into the log cache, it proceeds to perform its internal output action, i.e., updating the variables of any pertinent ODSS's and releasing the locks on the ODSS's, but then upon receiving the AT success notice, it skips the external output step and waits for the output success notice from the primary partner.

The shadow TMO or SvM execution may learn about the failure of the primary SvM execution in one of the following ways:

- Absence of a client request ID from the primary within a specific deadline which results in the ICC in the shadow node triggering the initiation of the local SvM as the primary SvM execution,
- Absence of an AT result notice from the primary within a specific deadline,
- An explicit AT failure notice from the primary, or

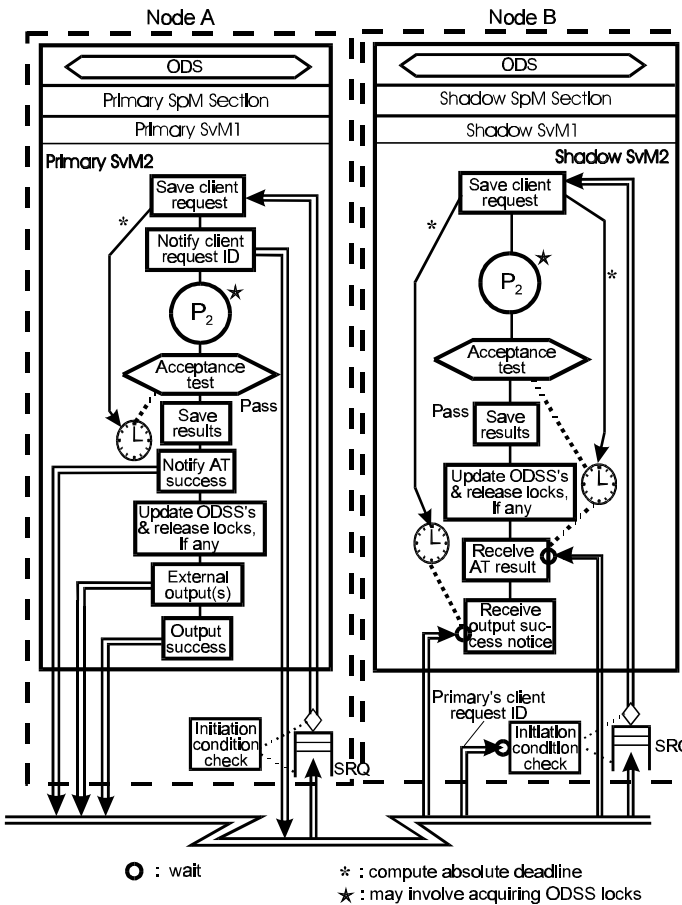


Figure 2. A fault-free method execution cycle of the SvM's in a primary and shadow TMO (adapted from [Kim97a])

- (d) Absence of an output success notice from the primary within a specific deadline.

In Figure 2, both the primary and shadow SvM executions use the same computational procedure. This makes them vulnerable to faults in the computational procedure design since in such a case the same design faults will be present in both the primary and shadow SvM's. To ameliorate this problem, we may construct each SvM using the recovery block language construct [Ran 95, Kim97a]. A recovery block may contain of multiple versions of a method procedure, called *try blocks*, besides an AT to judge the reasonableness of the results produced by each version. Besides the operational rules, basic rules for structuring the primary and shadow TMO methods have been formalized [Kim97a].

3. System architecture model, fault types, and fault frequency bounds

As a system architecture model that can represent a variety of point-to-point network architectures with different specializations, we chose the following architecture. The system consists of m different local area networks (LAN's), each LAN consisting of a maximum of n different local nodes and typically multiple gateway nodes. Within a LAN, the local nodes may be connected with one another either via a broadcast bus or via a point-to-point network. Two nodes in different LAN's communicate with each other through the gateways attached to each LAN. The nodes in a LAN connected by a point-to-point network execute the SNS scheme. Nodes in LAN's which have efficient broadcast facilities, some alternate NS scheme such as the periodic reception history broadcast (PRHB) or TTP [Kop93, Kim94a] which takes advantage of the cheap broadcast facility may be chosen. From now on, we concentrate on the nodes in the former type of LAN's for application of the PSTR/SNS scheme.

Every message sent from a source node to a non-neighbor destination node within R is stored for a while in each intermediate node while the node makes a routing decision. Such a routing scheme is called a store-and-forward routing scheme. Each LAN typically has two or more gateways which function in a redundant fashion. Thus, even if one of the gateways becomes faulty, each node in a LAN can still communicate with a node in a different LAN through another gateway.

In this section, we first consider various possible sources of faults in the system. Then we discuss the types and the frequency of fault occurrences that can be tolerated by the PSTR/SNS scheme.

3.1 Fault sources

Since the number of components in a typical point-to-point network architecture is large, a clear and yet accurate representation of all possible fault sources is a challenging issue. The most practical approach here is to

group various potential fault sources into a manageable set of categories.

In the model chosen, possible sources of faults in a node are represented by a *TMO method process*, an *incoming communication handling channel* (I-channel), and an *outgoing communication handling channel* (O-channel). Any observed fault of a node could be an instance of a combination of faults in the fault sources mentioned above.

The remaining fault source is one or more links of the point-to-point interconnection network. In the remaining discussion, we refer to each of the four fault sources described in the fault source model above as a *fault source component*. We assume that since the routing scheme is of the store-and-forward type, a permanent failure of any one of the fault source components in a node disables not only the node's processing capabilities but also the node's routing capabilities.

3.2 Fault types covered and fault frequencies assumed

The PSTR/SNS scheme has been designed to tolerate various types of faults that occur in various fault sources mentioned in the preceding section subject to the fault frequency assumptions stated below. Of the assumptions, A1-A4 have been made for the SNS scheme [Kim97b].

(A1) The processors in the system are of the fail-silent type and the task processes do not export erroneous messages.

(A2) The clocks in the nodes are kept synchronized sufficiently closely for practical purposes, i.e., for the given applications.

(A3) Each of the nodes performing store-and-forward functions (as well as the source node) transmits each stored message twice continuously. It is assumed that this makes the probability of transient fault occurrences in the components of the two neighbor nodes and those in the link between the two neighbor nodes causing message losses to be negligible.

(A4) Let S_N be the set of fault source components belonging to a node N , and $LL(P, Q)$ be {link connecting two neighbor nodes P and Q }. The time interval from the time of the occurrence of a permanent hardware fault F in S_N or in $LL(P, Q)$ to the time every healthy node in the system learns of the fault occurrence under the SNS scheme is the *detection period* of the SNS scheme for the fault F . The SNS scheme assumes that during that detection period, no other hardware fault occurs in node M , where $M \neq N$, or in $LL(I, J)$, where $(I, J) \neq (P, Q)$. The *worst-case detection latency of the SNS scheme* for all possible hardware faults in the system is denoted by WDL_{SNS} . Therefore, it is assumed that no second hardware fault occurs in the system within WDL_{SNS} time units after the occurrence of the first permanent hardware fault F .

Let S be the set of the models of the nodes in a PSTR station: $S = \{N_{pri}, N_{sha}\}$, where N_{pri} represents the

set of three fault source components (TMO method, I-channel, O-channel) in the primary node, and N_{sha} represents the set of three fault source components in the shadow node. Let SS denote the set of all models of the nodes in the application system, i.e., the models of the nodes in all PSTR stations that execute the fault-tolerant application. Obviously, $S \subset SS$. Let RT denote the maximum interval of time between the occurrence of a permanent fault in a PSTR station and the completion of the repairing of the faulty situation which may involve relocating a member node of the station as will be discussed in Section 4. It is safe to say that RT is much larger than WDL_{SNS} .

(A5) In a time interval of RT , the permanent failures of two or more nodes in SS at least one of which is the primary node of a PSTR station and another of which is the shadow node of the same PSTR station, are assumed to have a negligible probability of occurrence.

(A6) In a time interval of RT , the transient failures of both partner nodes in a PSTR station are assumed to have a negligible probability of occurrence.

(A7) The topology of the point-to-point network used, the nature of the application, and the task relocation arrangement are such that the node and link failures occurring within the bounds of A4, A5, and A6 cannot lead to permanent partitioning of the application system into inoperable disconnected subsystems during the lifetime of the application mission.

4. Overview of the SNS scheme

There are two types of nodes that execute the SNS scheme, the *worker* nodes and a *supervisor* node. The worker nodes are mainly responsible for judging their own health status, the health status of their neighbor nodes, and the health status of the links attached to themselves. The supervisor node performs all the duties that a worker normally does. In addition, it is responsible for collecting *fault suspicion reports* from worker nodes, using the collected information to judge whether a fault has indeed occurred, and then sending the *fault occurrence notice* to all the healthy worker nodes in the system.

The SNS scheme is executed by a set of nodes S_{NSR} that consists of n worker nodes and a supervisor node. Under the normal mode of operation, each node in S_{NSR} will have at least two healthy neighboring nodes. If the number of neighbors of a node N in S_{NSR} drops down to one, then we treat N as a component of its neighbor node Y . Thereafter, Y and its neighbors are responsible for detecting the faults in N . A subset of S_{NSR} is involved in actively executing some distributed real-time application tasks. Under the SNS scheme, *two copies* of every message are sent from a source node to a destination node, the first copy along one path P_1 and the second copy along an alternate path P_2 . P_1 and P_2 may partially overlap.

(1) Basic duties of worker nodes

All the healthy worker nodes in the system perform the following basic duties:

(1a) Every healthy worker node in the system periodically exchanges *heartbeat* signal messages with each of its healthy neighbors at the interval of π .

(1b) Absence of a heartbeat signal from a neighbor node within a specific deadline will raise the suspicion of the worker node. The suspecting node then proceeds to find out the location of the fault at the level of the fault source components modeled in Section 3. Here a healthy node does not distinguish between a permanent fault in the O-channel in its neighbor node from a permanent crash of the processor in the latter node.

(1c) Once the worker node locates the fault source component to the extent it can, it sends a *fault suspicion report* to the supervisor.

(2) Duties of the supervisor

The supervisor node performs the following additional duties:

(2a) Once the supervisor receives a fault suspicion report from a worker node, it judges whether the suspicion is indeed true. For this, the supervisor may use the fact that it has received a certain type of fault suspicion reports from $k > 1$ worker nodes.

(2b) After the confirmation of the fault, the supervisor proceeds to inform the relevant nodes of the detected faults.

(3) Special duties of the supervisor's neighbor nodes

The supervisor's neighbor nodes also perform the following additional duties:

(3a) Make a group decision about the health of the supervisor,

(3b) In case the current supervisor is judged to be faulty, participate in a new supervisor election in which each of the current supervisor's healthy neighbors takes part, and

(3c) Once one among the old supervisor's neighbors is elected as a new supervisor, the newly elected supervisor informs all the healthy worker nodes about the fault in the old supervisor as well as the assumption of its new supervisor role.

5. The PSTR/SNS scheme

Integration of the PSTR scheme with NS schemes tend to significantly improve the fault coverage and recovery time bounds over other versions of the scheme in which no NS scheme is incorporated.

5.1 Rules of the PSTR/SNS scheme

The operating rules of the PSTR/SNS scheme are a superset of those of PSTR discussed in Section 2 and so here we will only discuss the additional rules introduced in the PSTR/SNS scheme.

(1) Both the primary and the shadow PSTR nodes execute the SNS scheme.

(2) In the case in which a shadow method execution detects the absence of a data ID from the primary method within a certain deadline, it will change its role to that of the primary, pick up a new input data item to process, and execute its try block and acceptance test. Upon passing the AT and before updating its object data store segment (ODSS) and/or sending the output message out, the new primary method execution will seek the help of the SNS scheme executing in the host node to find out if any permanent I-channel fault has occurred in the host node and caused it to miss the data ID message from the partner method execution and switch its role to change from that of the shadow to that of the primary. For this, the new primary checks whether WDL_{SNS} time units have gone past since the time the data ID message would have arrived at the host node under normal fault-free circumstances. As mentioned in Section 3, WDL_{SNS} is the worst-case fault detection latency of the SNS scheme. This step is necessary to prevent the two partner nodes from entering dangerous inconsistent states. Only after the new primary method execution confirms that the host node has been fault-free, it will produce the output. Otherwise, it will initiate a spontaneous shutdown of the node. Thus, a faulty node does not produce any external outputs.

The steps taken by the supervisor node are as follows:

- (1) The steps for executing the SNS scheme: The supervisor performs all the requisite actions for the SNS scheme outlined in Section 4.
- (2) The step for checking and acting on anomalies detected in worker PSTR stations: The supervisor node checks whether any anomalous conditions have occurred in worker PSTR stations. Such conditions include the playing of the primary role by both partner nodes in a worker PSTR station, the lack of response from a primary partner to the shadow partner's request for a specific input data item, etc. In such cases, the supervisor will take an action to rectify the anomaly.
- (3) The step for notifying database updates to each of the healthy neighbors: Whenever the supervisor node updates its database on the supervisory information, it sends an update notice to its neighbors so that the neighbors may keep their own copies of the supervisor's database up-to-date. These actions are necessary in order that a newly elected supervisor (which will be one among the current supervisor's neighbors) can take over the supervisory functions in case the current supervisor node fails.

The steps taken by a healthy neighbor node Y of the supervisor can be formally described as follows:

- (1) The steps for executing the SNS scheme: The neighbor node Y performs all the requisite actions for the SNS scheme outlined in Section 4.
- (2) The step for saving a copy of any message bound for the supervisor node: Whenever node Y forwards a message bound for the supervisor node, it keeps a copy until it receives a new database update notice from the supervisor node.

(3) The step for updating a copy of the supervisory database: Upon receiving a database update notice from the supervisor node, node Y updates its copy of the supervisory database.

5.2 A modular implementation model for the PSTR/SNS scheme and a prototype implementation

A modular implementation model for the primary SpM execution support consists of the *Primary SpM Process* and four co-resident kernel-threads called the *Watchdog Timer and TMO Management Thread* (WTMT), the *Incoming Communication Thread* (ICT), the *Outgoing Communication Thread* (OCT), and the *Network Surveillance & Partnership Support Thread* (NPT). The implementation model for the primary SvM execution support is similar and will not be discussed.

All the four kernel-threads are structured as *time-triggered* (TT-) threads. They are periodically acting threads whose execution is driven by the real-time clock. This structuring eases the determination of the worst-case service times of these threads and hence eases the analysis of the recovery time bound of the PSTR/SNS scheme. The functions of the primary SpM process and the four kernel-threads are as follows:

- (1) **Primary SpM Process:** This is the combination of the primary logic of the SpM and additional features to support the PSTR/SNS scheme.
- (2) **WTMT:** This thread is mainly responsible for supporting the execution of the TMO methods. Specifically, this thread triggers the execution of SpM's at instants of real-time determined at the design time and also triggers the execution of SvM's upon the arrival of each client request message. Before triggering the execution of SvM's, this thread checks whether both the BCC and ordered isolation rules are satisfied. In addition, this kernel-thread checks whether the method execution violates the try block execution deadlines, and in case a deadline violation is detected, it orders the primary method to turn to a shadow through a message deposited in $Q_{timeout}$.
- (3) **NPT:** This is responsible for almost everything in the SNS scheme described in Section 4 except the generation of the heartbeat signals. For example, some or all of the functions such as periodic analysis of the received heartbeat signals, reporting fault suspicions, election of a supervisor, and supervisory function, are handled by this kernel-thread, depending on the role of the host node (a worker node, supervisor node or a neighbor of the supervisor node). This thread also notifies a TMO method process of any detected faults in the host node (by depositing an order to shut down in Q_{order} checked by the TMO method process). In addition, this thread honors any requests from the partner or supervisor.
- (4) **ICT:** This thread distributes the messages received in Q_{EXT} from other nodes to appropriate destination queues. In particular, it periodically forwards the messages that should be processed by NPT into Q_{NPT} and the client request messages into Q_{SR} checked by an SvM. In

addition, it also deposits any external input data items into Q_{EI} checked by a TMO method.

(5) **OCT**: This thread sends to the point-to-point network the messages found in Q_{OUT} such as output data items deposited into Q_{OUT} by the TMO method and the messages related to the SNS scheme deposited into Q_{OUT} by the NPT. In addition, this generates and sends heartbeat signals to each of the node's healthy neighbors as a part of the SNS scheme.

The modular implementation model for the shadow SpM execution support is similar to that of the primary SpM and will not be discussed due to space limit. If any fault occurs in the primary method execution, the shadow method execution will learn of the fault by an explicit notice from the primary (as in the case of an AT failure in the primary), in one of the ways outlined in Section 2.2, or by a notice from the NPT in the shadow node that the primary node has failed.

6. Recovery time bound analysis of the PSTR/SNS scheme

The analysis of a tight recovery time bound for any real-time fault tolerance scheme is an important task but it had been a scarce practice until a few years ago. These kinds of analyses provide useful guides to engineers who aspire to produce high-assurance fault-tolerant real-time computer systems. The analysis here is based on the prototype implementation discussed in Section 5.

Notations

(N1) **Time $t(a)$** : Time $t(a)$ is defined as the instant at which an event a occurs.

(N2) **Time interval $d(a,b)$** : Let a and b denote two types of ordered events that take place in the order (a, b) . Then $d(a,b)$ is defined as the time interval from the start of a to the end of b .

6.1 Definitions

(1) **Processing time L of a PSTR station**: L is defined as the time interval from the time of arrival of either a particular client request message at the service request queue Q_{SR} (in the case of an SvM) or a data item at an ODSS (in the case of an SpM) of the primary node to the time the final output action is completed.

(2) **Worst-case processing time L_{ff} of a PSTR station in the absence of faults**: L_{ff} is defined as the maximum value of L when a PSTR station operates under fault-free conditions.

(3) **Worst-case processing time L_{max} of a PSTR station in the presence of faults**: L_{max} is defined as the maximum value of L when faults occur in the PSTR station within the bounds established in Section 3.

(4) **Recovery time bound RTB of a PSTR station**: RTB is defined as the difference between L_{max} and L_{ff} .

(5) **Maximum message transmission time $TRANS$** : Maximum time interval that elapses from the time a message leaves the Q_{OUT} checked by the ICT of a node to

the time the message reaches the Q_{EXT} of any other node in the system. This time includes the time taken for redundant transmissions of the same message over two different paths as explained in Section 3.

(6) **Maximum thread turnaround time**: Maximum interval of time that elapses from the time an item arrives at any of the input queues of the thread to the time the corresponding output is produced. Maximum ICT turnaround time is denoted as MIT and maximum OCT turnaround time is denoted as MOT .

(7) **Worst-case fault detection latency WDL_{SNS} of the SNS scheme**: Maximum amount of time between the occurrence of a fault $F \in \{\text{node fault, link fault}\}$ and the learning of the fault occurrence by all the healthy nodes in the system.

(8) **Lag limit LGL of the shadow**: This is defined as the maximum interval of time a shadow method execution can lag behind the corresponding primary method execution without inducing the risk of missing the deadline for taking the final output action(s). If the shadow method execution exceeds this lag limit, it should enter a *node recovery mode* and try to catch up with the primary.

6.2 Recovery time bound of the PSTR/SNS scheme

Figure 3 shows the timing chart of the primary and shadow SvM executions under fault-free conditions. Various events that occur in the primary and the shadow nodes of a PSTR station under fault-free conditions are shown. The three parameters of interest in Figure 3 are MIT , MOT , and $TRANS$. According to the definitions given in Section 6.1, the worst-case time interval between the arrival of a client request message in Q_{EXT} of the primary node (event $I1_p-a$) and the time at which the ICT finishes moving this message to Q_{SR} (event $I2_p-a$) in Figure 3 is shown as MIT time units. Similarly, the worst-case time interval between the arrival of a message in Q_{OUT} (event $O1-i$) and the time at which the OCT finishes sending this message to the network (event $O1-s$) is shown as MOT time units. Finally, the worst-case time interval between the event $O1-s$ of the primary node and $I2-a$ of the shadow node is shown as $TRANS$ time units. For simplicity in the analysis of the recovery time bounds, we assume that the difference between $t(I1_p-a)$ and $t(I1_s-a)$ is negligible. Note that L_{ff} includes the time period during which the client request message waits in Q_{SR} to be picked up by the primary SvM execution, which includes the time period for satisfying the basic concurrency constraint and ordered isolation rules. The bound for this waiting time should also be known at the design time.

This timing chart facilitates the calculation of various deadlines used to detect failures. The deadline in the shadow node for the checking of the client request ID from the primary for a particular client request message X is calculated as

$$\begin{aligned}
 & t(I1_p-a) + d(I1_p-a, O1-i) + d(O1-i, O1-s) + \\
 & d(O1-s, I2-a) + d(I2-a, I2-p) + LGL \\
 \text{£ } & t(I1_s-a) + d(I1_p-a, O1-i) + MOT + TRANS + \\
 & MIT + LGL = DL_{ID} \quad (1)
 \end{aligned}$$

Here LGL , the lag limit, is the tolerable delay that can be incurred by the shadow SvM execution in checking the arrival of the client request ID from the primary partner. $t(I_{s-a})$ is the time at which X arrived at the shadow node, and $d(I_{p-a}, O_{1-i})$ is the sum of the queuing delay of X at the primary node and the time taken by the primary SvM process to execute the step for sending the ID of X to the shadow. The worst-case value for $d(I_{p-a}, O_{1-i})$, WQD (worst-case queuing delay), can be quite accurately estimated by the shadow node by using the times at which the OSN arrived at the shadow node in the immediately preceding cycle, the time at which X arrived at the shadow node and the subsequent time duration to satisfy the basic concurrency constraint and ordered isolation rules. Note that a copy of X is assumed to have arrived at both the primary and the shadow nodes roughly at the same time.

Thus, once a client request message arrives at the shadow node, the shadow SvM execution should check for the arrival of the ID of the client request message from the partner within $WQD + MOT + TRANS + MIT + LGL$ time units. This deadline DL_{ID} can also be viewed as $MOT + TRANS + MIT + LGL$ time units after the event occurrence $O_{1(i)}$ in the primary node. Suppose this "intermediate" deadline is violated. The shadow SvM

process will change its role to that of the primary, pick up a new client request message, deposit its ID in Q_{OUT} , and start the processing of a new request message. After the new primary passes the AT, it first checks whether the time interval of WDL_{SNS} units has gone past since the time at which the client request ID would have arrived at the shadow SvM execution's client request ID queue. Note that the time at which the shadow SvM execution checks this condition will be in the worst-case equal to $DL_{ID} + P_{exec}$, where P_{exec} is the worst-case time for executing the step of picking the new client request message, depositing the client request ID in Q_{OUT} , and finally executing both the primary try block and the AT after sending the ID (Figure 3). The time at which the client request ID would have arrived at the ID queue will be equal to $(DL_{ID} - LGL)$ and the shadow SvM execution checks for the presence of the client request ID LGL time units later in the worst-case. This means that the shadow SvM execution waits for $Z((DL_{ID} - LGL + WDL_{SNS}) - (DL_{ID} + P_{exec})) = Z(WDL_{SNS} - (LGL + P_{exec}))$ time units after executing its try block and AT, where $Z(x) = x$ if $x > 0$ and $Z(x) = 0$ if $x \leq 0$.

Also, the worst-case processing time of the SvM in the normal fault-free case is

$$L_{ff} = WQD + P_{exec} + DATR_OM + MOT \quad (2)$$

where $DATR_OM$ is the time taken to execute the steps of depositing the AT result and the output message in Q_{OUT} .

The shadow SvM process can learn of the faults occurred in the primary node in four major ways besides relying on a notice from the primary itself. The recovery time bound of the PSTR/SNS scheme can be determined by considering these cases.

Case 1: WTMT in the shadow node detects the absence of a client request ID from the primary by DL_{ID}

As soon as the shadow SvM execution reads the notice from the WTMT about the absence of the client request ID, it will change its role to that of the primary and send the client request ID out. Note that in this case the new primary will execute the step of depositing the ID into Q_{OUT} , execute the primary version of the application logic and then the AT, by taking up to P_{exec} time-units. After passing the AT, the new primary waits (for $Z(WDL_{SNS} - LGL - P_{exec})$ time-units) for confirmation from the NPT as to the fact that the host node has been fault-free. Once the NPT confirms the fault-free status, the new primary will send out the AT result, the output, and finally the OSN. The time interval between the time of arrival of the input data item and the time of sending the output is thus (using (1), (2), and the fact that $t(I_{p-a}) = t(I_{s-a})$)

$$\begin{aligned} & (DL_{ID} - t(I_{s-a})) + P_{exec} + Z(WDL_{SNS} - LGL - P_{exec}) \\ & + DATR_OM + MOT \\ & = (WQD + MOT + TRANS + MIT + LGL) + P_{exec} \\ & + Z(WDL_{SNS} - LGL - P_{exec}) + DATR_OM + MOT \\ & = L_{ff} + Z(WDL_{SNS} - LGL - P_{exec}) + MOT + TRANS \\ & + MIT + LGL \quad (3) \end{aligned}$$

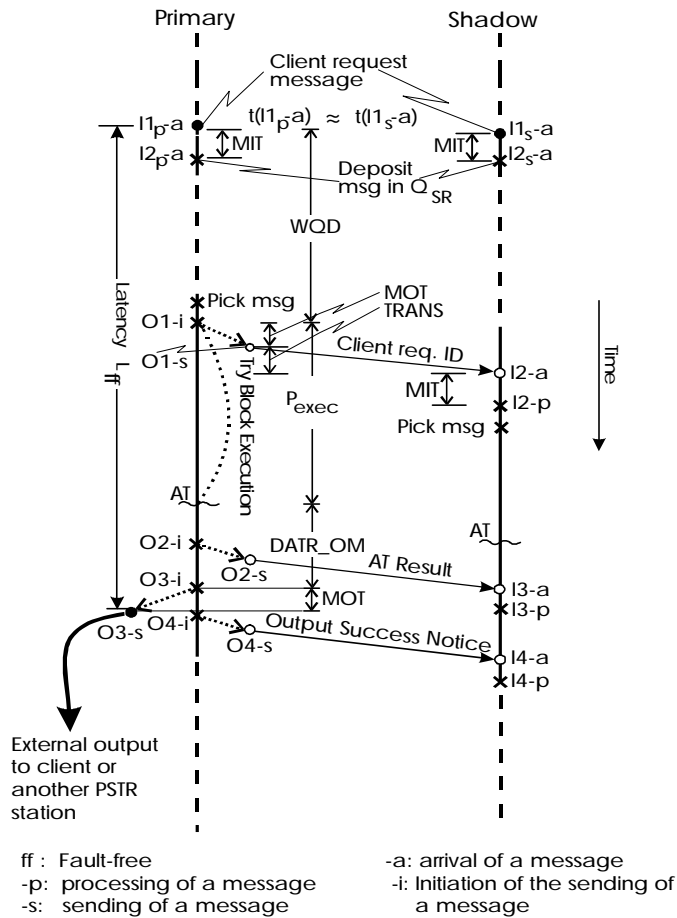


Figure 3. Timing chart for a primary SvM execution

The recovery time of the PSTR station is thus $MOT + TRANS + MIT + LGL + Z(WDL_{SNS} - LGL - P_{exec})$ time units.

Case 2: WTMT in the shadow node detects the absence of AT result from the primary by DL_{AT}

Based on an analysis similar to that was used in Case 1, it is easy to see that the recovery time is $MOT + TRANS + MIT + LGL$ time units.

Case 3: WTMT in the shadow node detects the absence of OSN from the primary within DL_{OSN}

This case is similar to case 2, and the recovery time here is again $MOT + TRANS + MIT + LGL$ time units.

Case 4: NPT in the shadow node detects that the primary node has failed.

Even though the NPT may notify the shadow SvM process at any time during the execution, the shadow SvM process will check for such a notice only when it is activated or when it waits for a message from the primary. Therefore, the recovery time cannot be greater than that in the three preceding cases (1, 2, and 3).

The recovery time bound analysis for an SpM execution is quite similar to the procedure described above and we will not illustrate it here. In the case of an SpM execution, WQD (worst-case queuing delay) is the time interval between the time a data item Y is deposited in an ODSS and the time the SpM reads the data item.

The results of the above analysis can be summarized as follows.

Proposition P1: If a fault occurs in a fault source component of a PSTR station operating under the PSTR/SNS scheme, then the recovery time for an SpM or SvM execution in the station is bounded by $RTB = MOT + TRANS + MIT + LGL + Z(WDL_{SNS} - LG - P_{exec})$, where all the parameters have the meanings given in Sections 6.1 and 6.2.

Numerical example: In the DREAM kernel prototype, $MIT = MOT = 8$ msec. Also, in [Kim97b] WDL_{SNS} for the DREAM kernel was calculated as 143 msec. If we choose $TRANS = 10$ msec, and $LGL = 1$ msec, $P_{exec} = 200$ msec, then $RTB = 27$ msec.

7. Conclusion

The TMO structuring scheme is one specific approach aimed for facilitating the general-form design style and the design-time guaranteeing of timely service capabilities of objects. The PSTR scheme formulated on the basis of the TMO structuring scheme is an approach for designing real-time fault tolerance capabilities into the TMO-structured application systems. In this paper, we have presented an integration of the PSTR with the SNS scheme which improves the fault coverage and recovery time bounds of the PSTR scheme. An analysis of the recovery time bounds of the PSTR/SNS scheme has also been presented. This analysis was based on a modular

implementation model. Efficient integration of the PSTR scheme with other NS schemes is one of many meaningful topics for future research.

Acknowledgments: The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR and in part by the University of California's MICRO Program under Grant 96-169.

References

- [Bas96] Bastani, F. et.al, "Toward Dependable Safety-Critical Software", In *Proc. IEEE CS 2nd Workshop on Object-oriented Real-Time Dependable Systems (WORDS '96)*, Laguna Beach, CA, Feb., 1996, pp. 86-92.
- [Hec91] Hecht, M. et.al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", *Proc. IEEE CS 21st Int'l Symp. on Fault-Tolerant Computing*, June 1991, Montreal, pp.462-469.
- [Kim94a] Kim, K.H., "Action-level Fault Tolerance", Ch. 17 in Sang H. Son ed., '*Advances in Real-Time Systems*', Prentice Hall, 1994, pp. 415-434.
- [Kim94b] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. IEEE CS 1st WS on Object-oriented Real-time Dependable Systems (WORDS '94)*, Oct. 1994, Dana Point, pp.36-45.
- [Kim97a] Kim, K.H., and Subbaraman, C., "Fault-Tolerant Real-Time Objects", *Communications of the ACM*, January 1997, pp. 75-82.
- [Kim97b] Kim, K.H., and Subbaraman, C., "A Supervisor-Based Semi-Centralized Network Surveillance Scheme and the Fault Detection Latency Bound", *Proc. IEEE CS 16th Symp. on Reliable Distributed Systems (SRDS '97)*, Durham, NC, Oct. 1997, pp.146-155.
- [Kop93] Kopetz, H. and Grunsteidl, G., "TTP-A: Time Triggered Protocol for Fault Tolerant Real-Time Systems", *Proc. IEEE CS FTCS-23*, Toulouse, France, June 1993, pp.524-533.
- [Mos96] Moser, L.E., Narasimhan, P., and Melliar-Smith, P.M., "Commentary: Object-oriented Programming of Complex Fault-Tolerant Real-Time Systems", *Proc. IEEE CS 2nd WS on Real-Time Dependable Systems (WORDS '96)*, Laguna Beach, CA, Feb. 1996, pp. 120-124.
- [Ran95] Randell, B., and Xu, J., "The Evolution of the Recovery Block Concept", Ch. 2 in Michael R. Lyu, ed., '*Software Fault Tolerance*', 1995, pp. 1-21.