Primary-backup replication: From a time-free protocol to a time-based implementation

Rui Oliveira*
Universidade do Minho
Portugal
rco@di.uminho.pt

José Pereira*
Universidade do Minho
Portugal
jop@di.uminho.pt

André Schiper
EPF Lausanne
Switzerland
andre.schiper@epfl.ch

Abstract

Fault-tolerant control systems can be built by replicating critical components. However, replication raises the issue of inconsistency. Multiple protocols for ensuring consistency have been described in the literature. PADRE (Protocol for Asymmetric Duplex Redundancy) is such a protocol, and an interesting case study of a complex and sensitive problem: the management of replicated traffic controllers in a railway system [5]. However, the low level at which the protocol has been developed embodies system details, namely timeliness assumptions, that make it difficult to understand and may narrow its applicability. We argue that, when designing a protocol, it is preferable to consider first a general solution that does not include any timeliness assumptions; then, by taking into account additional hypothesis, one can easily design a time-based solution tailored to a specific environment. This paper illustrates the benefit of a top-down protocol design approach, and shows that PADRE can be seen as an instance of a standard Primary-backup replication protocol based on View Synchronous Communication (VSC).

1 Introduction

Fault-tolerant control systems can be built by replicating critical components. Replication masks faults thus increasing availability. Nevertheless, replication raises the issue of inconsistency. In fact, when trying to ensure availability, safe operation of the system might be compromised due to inconsistency among the replicas. As a real world scenario, Essamé *et al.* [5] describe the replication of traffic controllers in a railway system and show how inconsistency can lead to a catastrophic failure. In [5], such scenario is avoided using a replication protocol which has been called the Protocol for Asymmetric Duplex Redundancy (PADRE).

*Partially funded by FCT ESCADA project (POSI/33792/CHS/2000).

PADRE has been conceived at a low level of abstraction and embodies system details, namely timeliness assumptions, that make it difficult to understand and may narrow its applicability. Indeed, from the presentation by Essamé *et al.* it may be difficult to see (1) how the problem compares to other fault-tolerant problems studied and described in the literature, and (2) how the proposed solution relates to common replication techniques.

We argue that, when designing a protocol, it is preferable to consider first a general solution that does not include any timeliness assumptions, and only then to take into account the additional hypothesis of a specific environment. Since PADRE is an interesting case study due to the complexity and sensitivity of its target system we propose to revisit the problem at a higher level of abstraction and to follow a top-down protocol design approach.

The goal and main contribution of this paper is to show that Asymmetric Duplex Redundancy can actually be seen as an instance of the more generic and widely studied primary-backup replication technique, and that the PADRE protocol can be seen as a particular implementation of a generic solution of primary-backup replication [3]. Besides the immediate clarification of the fault tolerance domain that this represents, it also strongly advocates the top-down design approach [9]: a generic solution is developed first, and then instantiated to a specific environment, thereby eliminating the effort required in starting from scratch.

Furthermore, we show that reasoning about the properties of the generic solution do not require "synchronous" assumptions, allowing us to delay the introduction of time constraints to the implementation step. This has the advantage of precisely showing how and where the correctness of the system depends on timing assumptions. In short, by establishing the mapping between the generic solution for primary-backup replication and PADRE, we clarify the role of each mechanism used in the implementation and provide a better understanding of the protocol as a whole.

The rest of the paper is structured as follows. In Section 2 we recall the train control system and the context for Asymmetric Duplex Redundancy. In Section 3 we describe the primary-backup replication technique and show how it can be useful in managing redundant traffic controllers. In Section 4 we succinctly describe the PADRE protocol, and in Section 5 we present the mapping between the generic solution and the PADRE protocol. Section 6 concludes the paper arguing for a top-down approach of protocol design and discusses the role of time in the PADRE protocol.

2 Train control system: specification

We recall in this section the specification of the train control system. A full description is given in [5].

A railway system is composed of tracks on which trains move (Figure 1). In order to control the circulation of trains and avoid collisions, tracks are divided into *sections*, numbered $\#1, \ldots, \#k, \ldots$. Each section is monitored by one *controller* (*i.e.*, section $\#k$ is monitored by controller $\#k$). The responsibility of controller $\#k$ is to prevent collisions of trains in section $\#k$. Each section is further decomposed in *blocks*: in each section, the corresponding controller has to ensure that some train T' cannot proceed to a block already occupied by another train T.

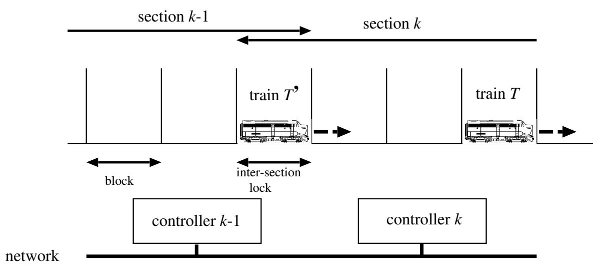

Figure 1. Train control system

One key problem is the hand-over of trains from one controller to the next one. For this purpose, two adjacent sections share a block called *inter-section lock* (Figure 1). Consider the inter-section lock shared by sections $\#k$-1 and $\#k$. Train T' traveling from section $\#k$-1 to section $\#k$ is allowed to leave the inter-section lock only after being registered by the controller $\#k$. If controller $\#k$ crashes, T' will not be able to leave the inter-section lock.

To improve the availability of the system, *i.e.*, to prevent the blocking of a train due to the crash of a controller, controllers can be duplicated (two replicas): a Primary and a Secondary (Figure 2). If controller $\#k$ is duplicated, then train T' should be able to leave the inter-section lock in spite of the crash of one of the two replicas. However, duplication introduces potential inconsistencies. Specifically, the following inconsistency that can lead to a catastrophic failure must be avoided. Consider two trains T and T' traveling from section $\#k$-1 to section $\#k$ and the duplication of the controller of section $\#k$ (Figure 1):

- The Primary controller of section $\#k$ registers train T, allows it to leave the inter-section lock, and crashes immediately after.
- The Secondary controller of section $\#k$ has not registered T (*i.e.*, it is not aware of the existence of T in section $\#k$) and takes control of the section.
- The Secondary registers train T' and allows it to leave the inter-section lock. It is now possible for train T' to bump into train T.

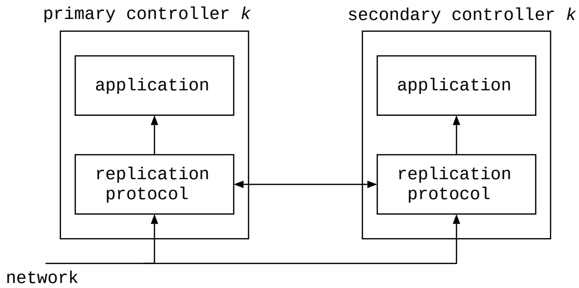

Figure 2. Duplicated controller (controller k)

The above inconsistency can be prevented by a replication protocol (Figure 2) ensuring the following safety condition:

- **Primary/Secondary consistency:** If the Primary controller $\#k$ registers some train T entering section $\#k$, then the Secondary controller $\#k$ can never take control of section $\#k$ without having previously registered train T.

This condition, here stated with regards to the "*registration*" of the trains, can be generalized to any message that changes the state of the controller.

3 The generic primary-backup replication technique

We consider in this section the well-know primary-backup replication technique [3, 1, 6] as a generic and high-level solution to handle the redundancy required by the train control system.

3.1 Principle of primary-backup replication

The primary-backup replication technique consists in having one *primary* and one or more *backups* ready to take over if the primary controller fails. Registration requests are handled by the primary. Once the primary has handled some request req it makes sure that each backup is up-to-date with respect to the new state, should it need to become the primary.

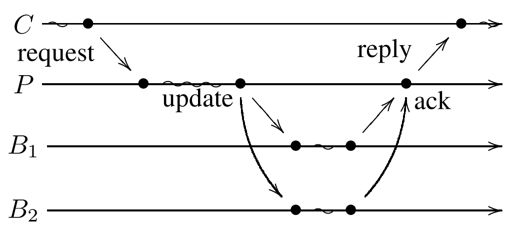

Figure 3. Overview of a primary-backup protocol (P is the primary; B_1, B_2 are the backups)

In the absence of controller failures and with reliable FIFO channels, a simple protocol for primary-backup replication could be as follows (see Figure 3): Upon receiving a request the primary P executes it, broadcasts a state update message to the group of replicas, waits for an acknowledgement (ack) from each backup, and then sends back the reply to the client.

Making the replication protocol correct when the controllers may crash and network messages may be lost is more difficult. However, much of the difficulty can be overcome by devising the replication protocol using the View Synchronous Communication abstraction [1, 2, 10, 11]. Figure 4 shows the architecture of this approach: a straightforward primary-backup protocol (upper box) based on a VSC protocol (lower box). The VSC layer provides a Group Membership Service and VSC communication primitives that we describe next.

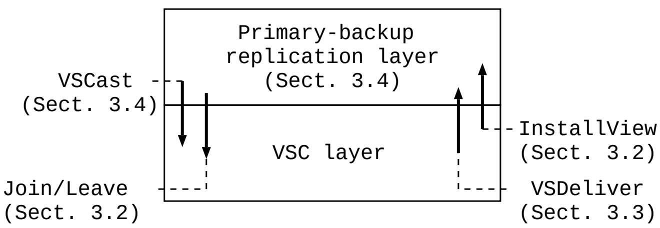

Figure 4. View Synchronous Communication layer

3.2 The Group Membership Service

The group membership service manages the composition of the group of controllers. For primary-backup replication the group consists of the primary and the backups. The successive membership of a group is given by a sequence of *views*, and the event by which a new view is provided to a controller is called the *InstallView* event (Figure 4). A controller may leave the group as a result of an explicit *leave* request, because it failed or because it is expelled by other members of the current view. Similarly, a controller may *join* the group, for example to replace a controller that has left the group. Upon joining, a controller initiates its state through a *state transfer*. One distinguishes two types of group membership services [11]: *primary-partition* and *partitionable*. In this paper we consider only the primary-partition membership service. Let v_i^p denote the i^{th} view installed by controller p. The primary-partition membership service is defined by an agreement property on the view history:

- **Agreement on the view history:** If p installs v_i^p and if q installs v_i^q, then we have $v_i^p = v_i^q$.

The agreement property allows us to denote a view simply by v_i without mentioning the controller superscript. The specification of a group membership service includes additional properties that we intentionally omit here.

In the context of primary-backup replication, if the primary of view v_i crashes, another replica must be elected as primary. With the above agreement condition, electing a new primary is very simple. After the crash of the primary, a new view v_{i+1} is installed: the new primary can be simply the replica in v_{i+1} with the smallest identification. Notice that if failure detection is unreliable, replicas can mistakenly be considered faulty, and excluded from the view.

Replicas that have been excluded from the membership (because of a crash or because they were mistakenly considered faulty) can join again (re-initiating their state through a state transfer). This might lead to two or more views with the same membership. Consider for example the following sequence of views: $v_i = \{p, q, r\}$, $v_{i+1} = \{p, r\}$ (q is mistakenly excluded or has crashed), $v_{i+2} = \{p, q, r\}$ (q has recovered). Views v_i and v_{i+2} are identical, which might pose problems. To ensure that we never have two different views with the same membership, a replica that is excluded from a view comes back with a new identity. This can be implemented using *incarnation numbers*. We use the notation q_x to denote the x^{th} incarnation of q (initially q_0). Using incarnation numbers, the above sequence becomes: $v_i = \{p_0, q_0, r_0\}$, $v_{i+1} = \{p_0, r_0\}$, $v_{i+2} = \{p_0, q_1, r_0\}$. Incarnation numbers will be omitted when there is no ambiguity.

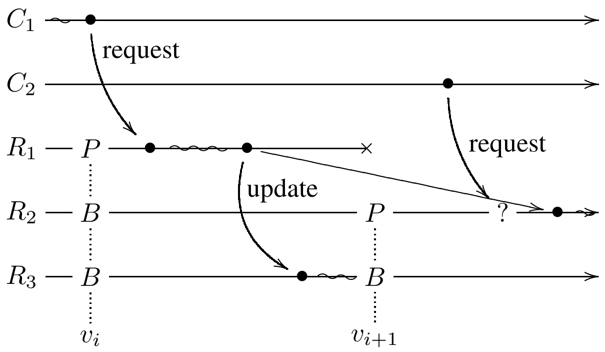

Figure 5. The need for view synchrony ($v_i = \{R_1, R_2, R_3\}$; $v_{i+1} = \{R_2, R_3\}$)

3.3 View Synchronous Communication (VSCast)

View synchronous communication is used by the primary, through primitives *VSCast* and *VSDeliver* (Figure 4), to broadcast update messages to the backups and ensures 1) that update messages are ordered with respect to view changes, and 2) that updates are delivered to either all or none of the replicas.

The need for (1) is illustrated in Figure 5, where R_1 is the primary – denoted by P – in view v_i, and R_2 is the primary in view v_{i+1}. Safety is compromised here because the new primary R_2 receives the request of client C_2 $before$ the $update$ message from the previous primary R_1. The resulting update message issued by R_2 would be inconsistent with the state of R_3. The execution of Figure 5 can be avoided by the *view synchrony* property [10, 11]:

- **View Synchrony:** If replica p belongs to two consecutive views v_i and v_{i+1}, and $VSDelivers\ m$ before installing v_{i+1}, then every replica q in $v_i \cap v_{i+1}$ $VSDelivers\ m$ before installing v_{i+1}.

View synchrony prevents the run depicted in Figure 5: as replica R_3 belongs to two consecutive views v_i and v_{i+1}, and *VSDelivers* the $update$ messages in view v_i, R_2 cannot install v_{i+1} without having previously *VSDelivered* the $update$ message.

View synchrony is however not sufficient to ensure the safety of the protocol. This is shown in Figure 6, where the update message of R_1 issued in view v_i is *VSDelivered* by R_2 and R_3 in view v_{i+1}. View synchrony is not violated here. Nevertheless the run of Figure 6 might also lead to inconsistencies between R_2 and R_3, e.g., to the same inconsistency as in the run of Figure 5. The following property prevents the run depicted in Figure 6.

- **Sending View Delivery:** A message *VSCast* in view v_i is *VSDelivered* in view v_i.

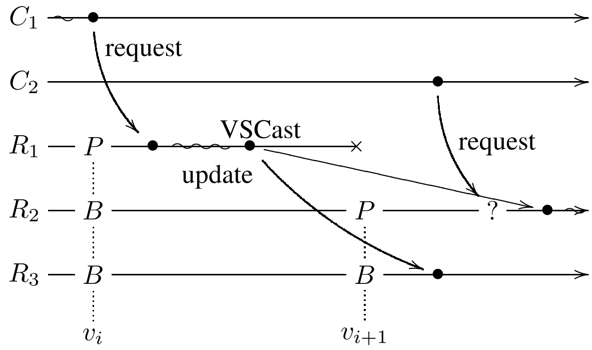

Figure 6. The need for sending view delivery ($v_i = \{R_1, R_2, R_3\}$; $v_{i+1} = \{R_2, R_3\}$)

3.4 Primary-backup replication protocol based on VSC

We detail now the primary-backup replication protocol based on VSC (Figure 4). First consider the registration request (Section 2). It is sent using a reliable broadcast, which ensures that if one correct replica receives the request, all correct replicas receive it [8].[1] The implementation of the reliable broadcast of message m leads each process receiving m to re-send m to all destination processes [8].

As for the rest of the protocol of the replicas, the behavior of the primary is the following (see Figure 3):

1. Each time a *request* is received it is processed by the primary. The processing computes an *update* message that represents the state change induced by the processing of *request*. The *update* message is VSCast to the current view v_i.
2. The primary waits for an acknowledgement (*ack*) from all backups in v_i, or the installation of a new view v_{i+1}:
   (a) If all acknowledgements are received, the primary returns the reply to the request and is ready to handle the next request (e.g., the next registration request in the train control system).
   (b) If a new view is installed and the primary remains the same, if not all acknowledgements have been

[1] Actually, in the standard primary-backup protocol, the request is sent only to the primary. We consider here that the request is sent to all the replicas in order to be closer to PADRE (Section 4), in which *input* messages are sent to all replicas.

received in view v_i, the primary continues waiting for an acknowledgement from all backups in $v_i \cap v_{i+1}$, or the installation of a new view.

The protocol for the backups is as follows:

3. Each backup waits for an *update* message from the primary or the installation of a new view:
   (a) If an *update* is VSDelivered, the backup updates its state accordingly, and sends back an acknowledgement to the primary.
   (b) If a new view v_{i+1} is installed (update messages from view v_i will no longer be received) then the backup assumes the role of primary in v_{i+1} when: i) v_{i+1} does not contain the previous primary, and ii) the backup is the replica with the smallest identification in v_{i+1}.

It should be noted that the protocol remains correct if a primary is mistakenly suspected and expelled from the view. Moreover, the protocol can tolerate the existence of two different primaries at the same global time t, *e.g.*, one primary p in view v_i and a different primary p' in view v_{i+1}. This is because the primary p, even though it can still receive registration requests and process them, will no longer be able to get acknowledgements from all its backups (at least one of them is in view v_{i+1}), and so to send replies. This guarantee is provided by the "sending view delivery" property: even though the primary p can VSCast update messages in view v_i, none of these messages will ever be delivered in view v_{i+1}. So p will wait for acknowledgements until it learns that it has been excluded from the view.

4 The PADRE protocol

We recall in this section the Protocol for Asymmetric Duplex Redundancy (PADRE) of [5], starting by highlighting some of its system assumptions.

4.1 Timeliness assumptions

PADRE has been conceived assuming a system that satisfies several timeliness properties. These properties are those of the Timed Asynchronous model [4]: (1) the drift between process clocks is bounded and the bound known, (2) messages sent over the network have performance/omission failure semantics, (3) there is a known bound for message handling by processes: processes either handle received messages within this interval or halt.

Based on these synchrony assumptions, communication delays can be evaluated and a datagram service with timeliness properties is assumed: every message received is classified as "slow" or "fast". The datagram service ensures validity and non-duplication of messages and establishes the guarantees of message timeliness (slow messages are discarded).

Additionally, PADRE assume the existence of a *hardware bi-stable relay* that in each moment points to one (and only one) of the processes (its use is explained in the next section).

4.2 Protocol overview

Consider the architecture of Figure 7. Input messages arrive from the network to both units, the Primary and the Secondary. If some train T in the inter-section lock needs to register to the controller of the new section, a *register* message is sent (to the Primary and the Secondary). This message is first *received* by the PADRE layer, and then *delivered* to the application (see Figure 7). The train T is only "registered" at the Primary (respt. at the Secondary) upon *delivery* of the register message. A hardware bi-stable relay ensures that only one unit can be the Primary at any time. If the current Primary fails, the relay automatically switches, leading the other unit to become the Primary (if its state permits, see Section 4.2.1).

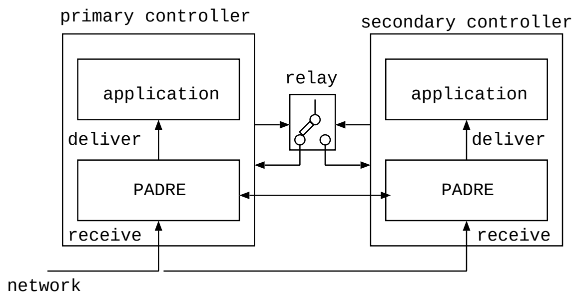

Figure 7. PADRE architecture

PADRE distinguishes four modes of operations for each controller unit: (1) *primary*, (2) *standby*, (3) *quarantine*, and (4) *failed*. A unit in the *primary* mode acts as the Primary. A unit in the *standby* mode acts as the Secondary and has its state consistent with the Primary. A Secondary whose state is not consistent with the Primary is in the *quarantine* mode. So, in order to satisfy the *Primary/Secondary consistency* condition of Section 2, the protocol must ensure that a Secondary switches to the *quarantine* mode if its state cannot be made consistent with the state of the Primary.

Based on these four modes, PADRE distinguishes two configurations: (1) the *nominal* configuration, in which one unit is in the *primary* mode and the other unit in the *standby* mode, and (2) the *safe* configuration, in which one unit is in the *primary* mode and the other is either in the *quarantine* or in the *failed* mode.

4.2.1 The protocol in the nominal and safe configurations

Consider the system in the nominal configuration. We describe the different behaviors of the Primary and the Secondary. The protocol of the Primary is the following (adapted from [5]):

1. The Primary periodically sends a message *"Don't switch to quarantine"* to the Secondary, and sets a *quarantine timeout delay* Q.

2. Each time an input message is received from the network (or from the Secondary), the Primary forwards the message to the Secondary, sets a *wait timeout delay* A, and waits for an acknowledgement:

   (a) If the acknowledgement is received before the timeout A expires, then the Primary delivers the input message to the application.

   (b) If no acknowledgement is received and the timeout A expires, then the Primary (1) stops sending *"Don't switch to quarantine"* messages, (2) stops forwarding input messages to the Secondary, and (3) delivers the input message to the application after expiration of the quarantine timeout Q. At this point the system is expected to be in the safe configuration with the Secondary either in quarantine or failed mode.

3. If the Primary fails, the relay will switch the Secondary, if in *standby* mode, to the *primary* mode.

The protocol for the Secondary is the following:

4. The Secondary waits for the periodic *"Don't switch to quarantine"* message from the Primary. Upon reception of such a message, it sets a *stay alive timeout delay* denoted by I. If no *"Don't switch to quarantine"* message is received before the timer expires, then the Secondary switches to the *quarantine* mode.

5. Each time an input message is received directly from the network, the Secondary forwards it to the Primary.

6. Each time an input message is received from the Primary, the Secondary sends an acknowledgement to the Primary, and delivers the message to the application.

No (automatic) action is taken upon the failure of the Secondary.

4.2.2 Switching from a safe to a nominal configuration

In a safe configuration the Secondary is unable to take control should the Primary fail. To increase the availability, it is worth returning to a nominal configuration as soon as possible. This is possible immediately if the safe configuration is due to quarantine of the Secondary.

Switching from a safe to a nominal configuration requires the Primary to transfer its state to the Secondary. As soon as the state transfer is terminated, the Primary (1) resumes forwarding input messages to the Secondary and waits for acknowledgement from the Secondary before delivering them, and (2) resumes sending *"Don't switch to quarantine"* messages. The Secondary simply waits for the state transfer to terminate, and then switches to the *standby* mode.

4.3 Observation

When comparing the protocol in Section 3.4 with the above PADRE protocol, it is clear that PADRE is more complex. This is because much of the complexity of the protocol in Section 3.4 is hidden in the VSC layer (Figure 4). We didn't discuss the implementation of the VSC layer [7].

5 PADRE as an instantiation of the generic primary-backup protocol

We show now that the VSC primary-backup replication protocol given in the previous section can actually be instantiated into the PADRE protocol. We can split PADRE in two parts: one part that corresponds to the upper box of Figure 4 (Primary-backup replication layer) and one part that corresponds to the lower box of the same figure (VSC layer). Concerning the VSC layer, we show that 1) the bistable relay (Figure 7) and the timing properties of messages allow us to ensure the property of the membership service (agreement on the view history), 2) the view synchronous property of VSC is obtained for free (only one backup), and 3) the fail-aware datagrams ensure the same view delivery property of VSC.

5.1 Primary-backup replication layer

5.1.1 Mapping of the messages

In the VSC primary-backup protocol (Figure 3), the primary (1) processes a client request, and (2) broadcasts an update message (to the members of the current view). If the processing is deterministic, steps (1) and (2) can be done in the reverse order. In this case, the client request is broadcast, and all the replicas process the request. This is done in PADRE. Taking this permutation into account, the mapping of the messages is as simple as this (refer to the primary-backup protocol overview in Figure 3):

- PADRE "input" messages correspond to client requests in the primary-backup protocol.

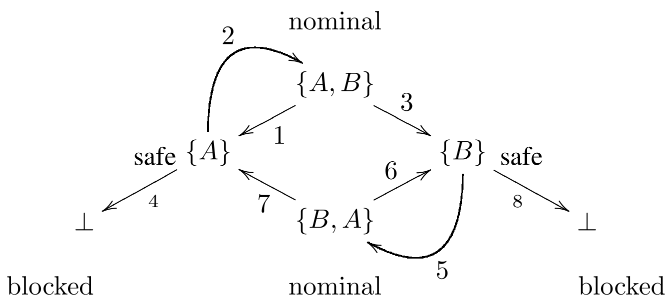

Figure 8. Mapping of PADRE mode changes into view changes

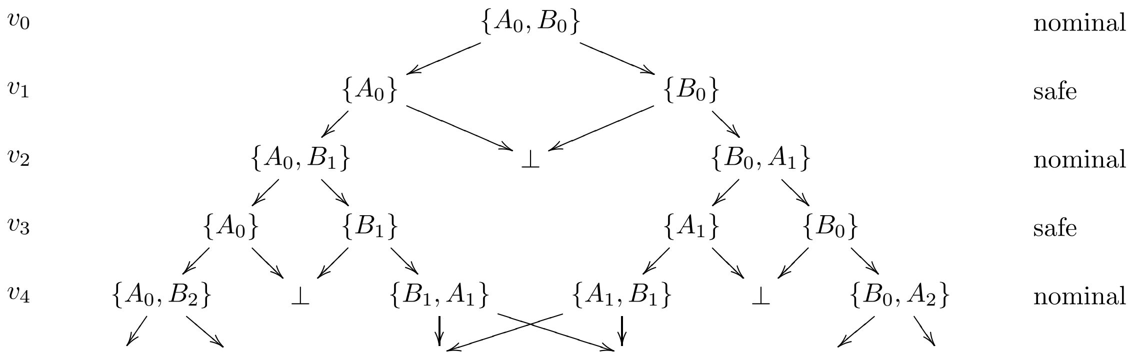

Figure 9. View history tree

- PADRE's forwarding of an "input" message from the primary to the secondary (Section 4.2.1, item 2) corresponds to the update VSCast done by the primary in the primary-backup protocol (Section 3.4, item 1).

- PADRE's acknowledgement messages (Section 4.2.1, item 6) correspond to the acknowledgement messages in the primary-backup protocol (Section 3.4, item 3).

5.1.2 PADRE ensures the reliable broadcast of "input" messages

PADRE ensures the reliable broadcast of input messages by having the Primary as well as the Secondary forward to the other unit each input message received (Section 4.2.1, items 2 and 5). This is similar to the standard implementation of reliable broadcast (Section 3.4, first paragraph).

5.2 VSC layer

5.2.1 PADRE ensures agreement of the view history

We show now that PADRE ensures the agreement on the view history. To do so, we must map the PADRE controllers' units *modes* onto membership *views*. Then we show that this mapping ensures the *view history agreement* property of Section 3.2. Let's denote by A and B the two controller units. The mapping between PADRE modes and views is as follows (the primary is the first unit in a view):

$mode(A)$	$mode(B)$	$View$
P	S	$\{A, B\}$
S	P	$\{B, A\}$
P	F/Q	$\{A\}$
F/Q	P	$\{B\}$

P = Primary; S = Standby; F/Q = Failed or Quarantine.

Moreover, we denote by $\perp$ a state in which the system is blocked (*i.e.*, no new view can be installed). The mapping of PADRE mode changes into view changes follows immediately (see Figure 8). Transition (1) occurs when the Secondary switches to the *quarantine* mode or crashes. Transition (2) corresponds to a switch from a *safe* to a *nominal configuration*. Transition (3) results from the switch of the relay due to the crash of A. Transition (4) takes place when A crashes in a safe configuration. Transition (5) is the symmetric of transition (1), (6) the symmetric of (2), (7) the symmetric of (3), and (8) the symmetric of (4).

Figure 8 does not distinguish the various incarnations of the controller units. Using the notation introduced in Section 3.2, we denote by A_i (respt. B_i) the i^{th} incarnation of A (respt. B), and assume that initial incarnation number is 0 for both units. So the view v_0 on which both units initially agree is $\{A_0, B_0\}$. Figure 9 shows the possible view histories starting from v_0.

The agreement on the view history property of Section 3.2 is equivalent to having the two controller units agree on one path of the possible view history paths depicted in Figure 9. To show that PADRE ensures this agreement, it is sufficient to show that PADRE ensures agreement between two nominal configurations, *i.e.*, from v_{2k} to v_{2k+2} (Figure 10). The agreement on the view history follows directly.

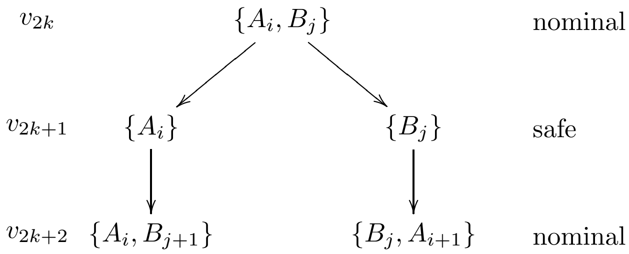

Figure 10. Nominal to nominal configuration

We start by considering the path $\{A_i, B_j\} \rightarrow \{B_j\} \rightarrow \{B_j, A_{i+1}\}$ (the easy case), and then the path $\{A_i, B_j\} \rightarrow \{A_i\} \rightarrow \{A_i, B_{j+1}\}$ (the tricky case).

i) PADRE ensures agreement on the path $\{A_i, B_j\} \rightarrow \{B_j\} \rightarrow \{B_j, A_{i+1}\}$

Let units A and B be both in view $v_{2k} = \{A_i, B_j\}$. If unit B installs view $v_{2k+1} = \{B_j\}$, it is because it has detected the crash of unit A, thanks to the relay. Because the relay is reliable, A indeed has crashed and does not install any view v_k.

If unit B installs view $v_{2k+2} = \{B_j, A_{i+1}\}$, it is because it has detected the recovery of A. Upon recovery, A will necessarily adopt the view v_{2k+2} obtained from B, *i.e.*, both units agree on view v_{2k+2}.

ii) PADRE ensures agreement on the path $\{A_i, B_j\} \rightarrow \{A_i\} \rightarrow \{A_i, B_{j+1}\}$

This case is more tricky. Figure 11 illustrates divergence that must be prevented. This can happen as follows. Initially both units agree on the view $v_{2k} = \{A_i, B_j\}$. Then unit A suspects unit B to have crashed and installs view $v_{2k+1} = \{A\}$. However unit B is not aware that it has been suspected: B is still in view v_{2k}. Later A crashes, the relay leads B do detect it, and to install view $v_{2k+1} = \{B\}$: agreement is violated on view v_{2k+1}.

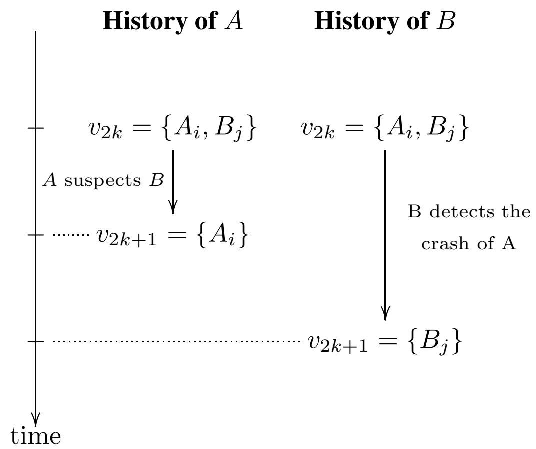

Figure 11. View divergence to avoid

PADRE ensures agreement on the path $\{A_i, B_j\} \rightarrow \{A_i\} \rightarrow \{A_i, B_{j+1}\}$ by preventing the run of Figure 11 from occurring. This is done, thanks to the *"Don't switch to quarantine"* messages, to the timeouts Q, I, and to the parameters Δ, ρ of the Timed Asynchronous model (Section 4.1). If we have $Q \geq \Delta(1 + \rho) + (I + 2\rho)$, then when the timeout Q expires on the Primary, the Secondary is in the *quarantine* mode, or has crashed (see [5] for details). In terms of views this means that when unit A installs view $v_{2k+1} = \{A_i\}$, it knows that B will never install view $v_{2k+1} = \{B_j\}$ (see Figure 12).

5.2.2 PADRE ensures the "view synchrony" and "same view delivery" properties

Finally, we show that PADRE ensures "view synchrony" and "same view delivery". The view synchrony property is trivially ensured by PADRE since for all configuration changes we have $|v_i \cap v_{i+1}| = 1$. Indeed, in this case we have only one process in $v_i \cap v_{i+1}$: violation of view synchrony can only occur with a least two processes in $v_i \cap v_{i+1}$.

The "same view delivery" property is ensured in PADRE by *fail-aware datagrams* [4]. PADRE computes an upper

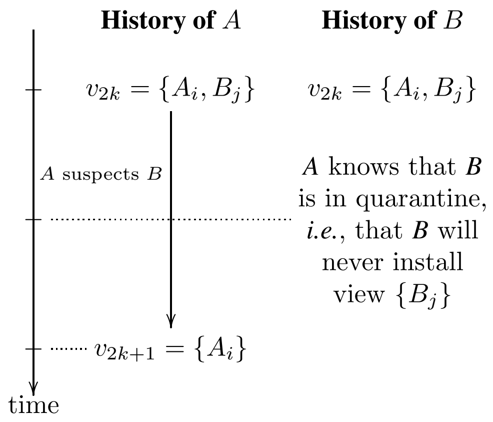

Figure 12. Unit A delays the installation of v_{2k+1} to avoid view divergence

bound on the real transmission delay of each message, and classifies messages as *fast* and *slow*. A fast message is a message that has experienced a "real" transmission delay of at most Δ time units.[2] The same view delivery property is ensured in PADRE by discarding "slow" messages.

6 Discussion

The VSC primary-backup protocol and PADRE can be compared from two perspectives: (i) from the perspective of the system models, and (ii) from the perspective of complexity. From the perspective of the system models, we have on one side a time-free protocol (the VSC primary-backup protocol) and on the other a time-based protocol (PADRE). From the point of view of complexity, because of its multiple timing parameters A, R, I, ω, Q, PADRE is clearly the more complex of the two protocols. Nevertheless, we have shown that PADRE can be seen as an instance of the VSC primary-backup protocol. This might look surprising, but there is no contradiction here. PADRE is complex because it is built on low-level abstractions and handles time explicitly. By comparison, the VSC primary-backup protocol is built on the high-level time-free VSC abstraction. So, much of the complexity is hidden in the implementation of the VSC abstraction. Even though time has to be taken into account in a protocol, it is better to keep it as deep as possible in the architecture. We believe that this is an important design principle that reduces the risk of errors, and allows for a better understanding of the role of time in a protocol.

To illustrate this point, we now show the clarification and improvements that we can get from a careful comparison of PADRE and the VSC primary-backup protocol. We show (1) a place in PADRE where the timing analysis guarantees safety, (2) a place in PADRE where the timing analysis is not related to safety, (3) a place where time could be suppressed, and (4) an omission in PADRE that can lead to a catastrophic failure.

[2] Δ is a parameter of the Timed Asynchronous model [4].

Timeliness to guarantee safety: The timing analysis related to the computation of the timeouts Q (Section 4.2.1, item 1) and I (Section 4.2.1, item 4) is essential to guarantee safety. An inadequate value for these two parameters can lead to a catastrophic failure. The dimensioning of Q and I ensure the agreement on the view history property of group membership (see Figures 11 and 12).

Timeliness not related to safety: While the parameters Q and I are related to safety, this is not the case of the timeout parameter A (Section 4.2.1, item 2), which is of a different nature. Its role is to implement failure suspicions. This is because "input" messages play two roles in PADRE: (1) application messages (subject to the "view synchrony" property in the VSC protocol), and (2) failure detection ("*are you alive*") messages. The failure detection role can be seen in Section 4.2.1, item 2b, where the non reception of the acknowledgement leads to the suspicion of the Secondary. With this explanation, and because PADRE and the VSC primary-backup protocol tolerate incorrect failure suspicions, it becomes clear that the timeout A is not safety critical. A better description of the protocol would lead us to replace item 2 in Section 4.2.1 by:

2. Each time an input message is received from the network (or from the Secondary), the Primary forwards the message to the Secondary and waits for an acknowledgement:
   (a) If the acknowledgement is received, then the Primary delivers the input message to the application.
   (b) If the Secondary is suspected, then the Primary (1) stops sending *"Don't switch to quarantine"* messages, (2) stops forwarding input messages to the Secondary, and (3) delivers the input message to the application after expiration of the timeout delay Q.

Actually, it makes sense from an implementation point of view to have "input" messages play two roles, but this is an optimization.

Where time can be removed: PADRE assumes the fail-aware datagram service to transmit the "*Don't switch to* *quarantine*" messages (denoted DSQ hereafter) and discards slow messages in order to guarantee that the Secondary cannot use an old message to refresh its stay-alive timeout delay ([5], Section 5.3). There is a simpler solution, inspired by the VSC protocol, which does not use time. The solution consists of tagging the DSQ messages with the current view number: while in view $\#i$, the Primary sends $(i,\ DSQ)$ to the other unit. If the other unit is in standby mode (also in view $\#i$), it simply ignores all messages not tagged with the current view number. If the other unit is in quarantine mode, it ignores the DSQ messages. Once the other unit returns to the standby mode, the view number becomes $i + 1$, and all old DSQ messages are ignored.

Omission in PADRE: One critical case has been overlooked in [5]. Consider the beginning of a nominal configuration. The Secondary has not yet switched on its I timer (Section 4.2.1, item 4) and it is awaiting the first DSQ message before doing so. If the first DSQ message arrives late at the Secondary, it can be shown that the situation of Figure 11 can happen, which may lead to a catastrophic failure. To prevent this case, the Secondary must start its A timer without waiting for the first DSQ message.

To conclude, we believe that VSC is a powerful abstraction in which the primary-backup protocol can be expressed in a simple way. Whenever a primary-backup replication protocol has to be designed for a specific environment, a good principle is to consider the implementation of the VSC properties in the specific environment. Optimizations should be considered only afterwards.

References

[1] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, February 1987.

[2] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, August 1991.

[3] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. The Primary-Backup Approach. In Sape Mullender, editor, *Distributed Systems*, pages 199–216. ACM Press, 1993.

[4] F. Cristian and C. Fetzer. The Timed Asynchronous System Model. Technical Report CSE97-519, Department of Computer Science, UCSD, 1997.

[5] D. Essamé, J. Arlat, and D. Powell. PADRE: A Protocol for Asymmetric Duplex REdundancy. In *IFIP 7th Working Conf. on Dependable Computing in Critical Applications (DCCA-7)*, pages 213–232, San Jose, CA, USA, January 1999.

[6] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer*, 30(4):68–74, April 1997.

[7] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, Jan 2001.

[8] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[9] G. Le Lann. On Real-Time and Non Real-Time Distributed Computing. In *9th Intl. Workshop on Distributed Algorithms (WDAG-9)*, pages 51–70. Springer Verlag, LNCS 972, September 1995. Invited paper.

[10] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *IEEE 13th Intl. Conf. Distributed Computing Systems*, pages 561–568, May 1993.

[11] R. Vitenberg, I. Keidar, G.V. Chockler, and D. Dolev. Group Communication Specifications: A Comprehensive Study. Technical report, Dept. of Computer Science, Technion, Israel, September 99.