Real-Time Primary-Backup (RTPB) Replication with Temporal Consistency Guarantees

Hengming Zou and Farnam Jahanian

Real-time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109
{*zou, farnam@eecs.umich.edu*}

Abstract

A common approach to building fault-tolerant distributed systems is to replicate servers that fail independently. The objective is to give the clients the illusion of service that is provided by a single server. The main approaches for structuring fault-tolerant servers are passive *and* active *replication. This paper presents a primary-backup (passive) replication scheme for supporting fault-tolerant real-time applications. The proposed scheme, called Real-Time PB (RTPB) replication service, is an elegant and simple approach that provides the benefits of fault-tolerance, real-time access, and temporal consistency guarantees that are not otherwise easily attainable.*

This paper formally defines two types of temporal consistency, namely external temporal consistency *and* inter-object temporal consistency*. By introducing a key concept called* phase variance*, we are able to build our temporal consistency models and derive necessary and sufficient conditions that can be used as the basis for update and transmission scheduling that achieve temporal consistency guarantees. Furthermore, we prove that the term* phase variance *used in the models can be bounded under various scheduling algorithms, namely EDF, Rate Monotonic [20], and Distance-Constrained Scheduling [9].*

The paper also presents an implementation of the real-time primary-backup replication scheme with the aforementioned temporal consistency models. This implementation was developed within the x-kernel architecture on the MK 7.2 microkernel from the Open Group. The results of a detailed performance evaluation of this implementation is also discussed.

1 Introduction

With ever-increasing reliance on digital computers in *embedded real-time systems* for diverse applications such as avionics, automated manufacturing and process control, air-traffic control, and patient life-support monotoring, the need for dependable systems that deliver services in a timely manner has become crucial. Embedded real-time systems are in essence *responsive:* they interact with the environment by "reacting to stimuli of external events and producing results, within specified timing constraints" [14]. To guarantee this responsiveness, a system must be able to tolerate failures. Thus, a fundamental requirement of fault-tolerant real-time systems is that they provide the expected service even in the presence of failures.

Most real-time computer systems are distributed and consist of a set of nodes interconnected by a real-time communication subsystem. Conceptually, a real-time computer system provides a set of well-defined services to the environment. These services must be made fault-tolerant to meet the availability and reliability requirements on the entire system. Therefore, some sort of redundancy must be employed for failure detection and recovery. This redundancy can take many forms: it may be a set of replicated hardware or software components that can mask the failure of a component (*space redundancy*), or it may be a backward error recovery scheme that allows a computation to be restarted from an earlier consistent state after an error is detected (*time redundancy*).

Two common approaches for space redundancy are *active* (state-machine) and *passive* (primary-backup) replication. In active replication, a collection of identical servers maintain copies of the system state. Client write operations are applied atomically to all of the replicas so that after detecting a server failure, the remaining servers can continue the service. Passive replication, on the other hand, distinguishes one replica as the primary server, which handles all client requests. A write operation at the primary server invokes the transmission of an update message to the backup servers. If the primary fails, a failover occurs and one of the backups becomes the new primary.

Although maintaining redundant components adds overhead to the system, this overhead can be reduced by exploring weak consistency semantics of applications. The resulted relaxization of consistency constraints could dramatically reduce system cost. This is true not only in the field of fault tolerance, but in other areas too. For example, many researchers in the database field have long recognized that strict serializability is too costly and often unnecessary. Instead, relaxed correctness criteria are used in scheduling database transactions which consequently permits a higher degree of concurrency. Similarly, imprecise computations exploit the fact that some computations can be completed successfully even if the input data is not totally up to date [21].

Different consistency semantics exist and are used in diverse applications depending on the objective and environment of the tasks. One category of consistency semantics that is particularly relevant to the primary-backup replication in a real-time environment is the *temporal consistency*, which is the consistency view seen from the perspective of the time continuum. Two common types of temporal consistency are the *external temporal consistency* which deals with the relationship between an object of the external world and its image on the servers, and the *inter-object temporal consistency* which is concerned with the relationship between different objects or events.

External temporal consistency at the primary server is needed because the primary provides services to outside clients. In order to provide meaningful and correct service, the state of the primary must closely reflect that of the actual world, or in other words, be consistent with the outside world. This consistency is also needed at the backup server because the backup is supposed to replace the primary in case of primary failure. The closeness of the state of the backup to that of the actual world is vital for a successful failover. Usually the restriction of the closeness placed on the backup is not as tight as that on the primary but must be within a tolerable range for the applications. Inter-object temporal consistency comes into play when two objects or two events are related in a temporal sense. For example, when an airplane takes off, there is a time bound between accelerating the plane and the lifting of the plane into air because the runway is of limited length and the airplane can not keep accelerating on the runway indefinitely without lifting off.

This paper presents the design and implementation of a real-time primary-backup replication scheme that combines fault-tolerant protocols, real-time scheduling, temporal consistency guarantees, and flexible x-kernel architecture to accommodate various system requirements. This work builds on the *Window Consistent Replication Service* by Mehra et al. [22] but distinguishes itself from that work in the following areas:

- The temporal consistency model is more general.
- Inter-object temporal consistency is proposed.
- Implementation is built within x-kernel architecture.

The most important contributions of this paper are the introduction of the concept of *inter-object temporal consistency*, the definition of the term *phase variance*, the theoretical models of all kinds of temporal consistency semantics, and the necessary and sufficient conditions that can be used as a basis for update and transmission scheduling such that (both external and inter-object) temporal consistency at both the primary and backup are preserved. The implementation is built within the x-kernel architecture on MK 7.2 microkernel from the Open Group[1]. Performance data are collected to evaluate the implementation.

The rest of the paper is organized as follows: section 2 introduces the concept of external temporal

[1] Open Group is formerly known as the Open Software Foundation (OSF)

consistency and develops an abstraction model for it. Section 3 presents the concept of inter-object temporal consistency. Section 4 addresses implementation issues in our experience of building a real-time primary-backup replication system followed by performance analysis of our system in Section 5. Section 6 summarizes the related work, and finally, Section 7 draws some conclusions about the RTPB system and discusses possible future extensions.

2 External temporal consistency

External temporal consistency is concerned with the relationship between an object of the real world and its image on a server. The problem of enforcing a temporal bound between an object/image pair is essentially equivalent to the problem of enforcing such a bound between any two successive updates of the object on the server. If the maximum time gap between a version of an object on the server and the version of the object in the real world must be bounded by some δ, then the time gap between two successive updates of the object on the server must be bounded by δ too, and vice versa. Hence, the problem of guaranteeing an object's external temporal consistency is transformed to the problem of guaranteeing that the time gap between two successive updates of the object on the server is bounded by some δ. If we define timestamp $T_i^P(t), T_i^B(t)$ at time instant t to be the finish time of the last update of object i before or on time instant t at the primary and backup respectively, then the temporal consistency requirement stipulates that inequalities $t - T_i^P(t) \leq \delta_i^P$ and $t - T_i^B(t) \leq \delta_i^B$ hold at all t. This concept of external temporal consistency is illustrated in Figure 1.

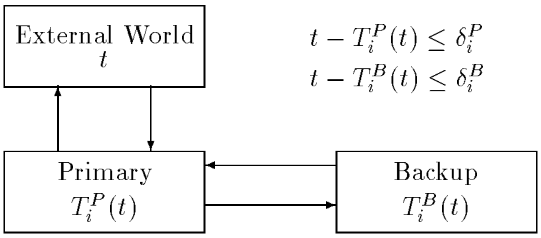

Figure 1: concept of external temporal consistency

2.1 Consistency at primary

Here we want to find out the condition that can guarantee an object's external temporal consistency constraint at the primary. Let:

O_i^P denote object i at the primary.
$T_i^P(t)$ denote timestamp of O_i^P at time instant t.
p_i denote the period of the task that updates O_i^P.
e_i denote the execution time of the task updating O_i^P.
δ_i^P denote the external consistency constraint for O_i^P.

A sufficient condition to guarantee the external temporal consistency is given below:

Lemma 1: External temporal consistency for object O_i^P at the primary is satisfied if $p_i \leq (\delta_i^P + e_i)/2$.

The correctness of the lemma is intuitive because any two consecutive invocations is bounded by $\delta_i^P + e_i$ and hence the temporal distance of any two time instants is within δ_i^P. A proof is supplied in the appendix.

The above condition can be significantly relaxed with the introduction of the concept of phase variance.

Definition 1: The k-*th phase variance* v_i^k, $k = 1, 2, \ldots, \infty$ of a task is the absolute difference between the time gap of its k-*th* and (k-1)-*th* invocations, and its period p_i. Let I_k, I_{k-1} represent the k-*th*, (k-1)-*th* invocations of the task, respectively, Then $v_i^k = |(I_k - I_{k-1}) - p_i|, \quad k = 1, 2, \ldots, \infty$

Figure 2 and 3 illustrate this concept.

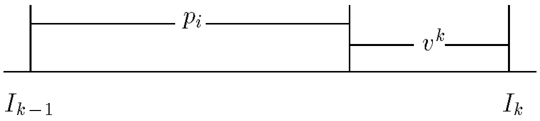

Figure 2: k-th phase variance, $I_k - I_{k-1} \geq p_i$

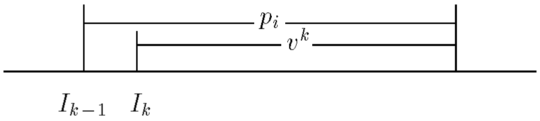

Figure 3: k-th phase variance, $I_k - I_{k-1} < p_i$

Definition 2: The *phase variance* v_i of a task is the maximum of its k-*th phase variance.* i.e.: $v_i = \max(v_i^k), k = 1, 2, \ldots, \infty$.

Since any two consecutive invocations of a periodic task is bounded between e_i and $2p_i - e_i$, it follows immediately from the definition of phase variance that:

$$v_i = \max(|(I_k - I_{k-1}) - p_i|) \leq p_i - e_i \tag{2.1}$$

Now, we can derive a necessary and sufficient condition that guarantees external temporal consistency for an object at the primary.

Theorem 1: External temporal consistency for O_i^P at the primary is satisfied if only if $p_i \leq \delta_i^P - v_i$. here v_i is the phase variance of the task that updates O_i^P.

The proof is supplied in the appendix.

Bound of phase variance

The result of Theorem 1 is of little use if phase variance can not be bounded by a bound that is better than the one given by Inequality 2.1. Fortunately, we are able to derive better bounds on phase variance for various scheduling algorithms.

Theorem 2: Inequalities $v_i \leq xp_i - e_i$ and $v_i \leq (x.p_i)/(n(2^{1/n} - 1)) - e_i$ are satisfiable under EDF and Rate Monotonic Algorithm [20], respectively. Here n is the number of tasks on the particular processor, x is the utilization rate of the task set.

A proof is supplied in the appendix.

From Theorem 1 and 2, we see that the restriction on an object can be significantly relaxed if the utilization rate of a task set is known (this is usually the case). It can also be shown that if the number of objects whose external temporal consistency we want to guarantee is less than the number of tasks in the task set, the bound on phase variance can be further tightened. The formula that takes into account the number of objects for which we want to guarantee temporal consistency is straight forward to derive.

Zero bound of phase variance

Thus far, we have demonstrated that phase variance can indeed be bounded by some known number. In fact, we can ensure that the phase variance is exactly zero in most cases through a direct application of the schedulabilities results from Distant-Constrained Scheduling (DCS) [9].

The DCS algorithm is used to schedule real-time tasks in which consecutive executions of the same task must be bounded, i.e. the finish time for one execution is no more than δ time units apart from the next execution. The solution proposed to distanced-constrained scheduling is largely based on the Pinwheel problem. A set of schedulers including Sa, Sx, and Sr that can achieve the purpose have been discussed [9].

Consider a task set $T = T_1, T_2, \ldots, T_n$. Suppose e_i and c_i denote the execution time and distance constraint of task T_i, respectively. Any two invocations of task T_i is bounded by c_i. Han and Lin [9] have shown that the task set can be feasibly scheduled under scheduler S_r if $\sum_{i=1}^{n} e_i/c_i \leq n(2^{1/n} - 1)$.

If we substitute p_i for c_i, then each invocation of task T_i is executed at exactly the same interval p_i after some iterations (could be 0). Thus, the phase variance of task T_i in this case is $|(I_k - I_{k-1}) - p_i| = |p_i - p_i| = 0$. Therefore, we have the following theorem.

Theorem 3: $v_i = 0$ is satisfiable by scheduler S_r if

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq n(2^{1/n} - 1) \tag{2.2}$$

Combining with the result of Theorem 1, one can show that under the condition stated in inequality 2.2, the condition to guarantee external temporal consistency for an object is relaxed to that of ensuring that the period of the task updating the object is bounded by the temporal constraint placed on the object.

2.2 Consistency at backup

In a primary-backup replication service system, the temporal consistency at the backup site is maintained by the primary's timely sending of update messages to the backup. Our interest here is to find out at what frequency the primary should schedule update messages to the backup such that the external temporal consistency of an object at the backup is guaranteed. We introduce the following additional notations:

O_i^B denotes object i at the backup.
$T_i^B(t)$ denotes timestamp of O_i^B at time instant t.
r_i denotes the period of the task that updates O_i^B.
e'_i denotes the execution time of the task updating O_i^B.
ℓ is the communication delay from primary to backup.
δ_i^B denotes the external temporal constraint for O_i^B.

A sufficient condition to guarantee the external temproal consistency is given below:

Lemma 2: External temporal consistency for O_i^B at the backup is satisfied if $r_i \leq (\delta_i^B + e_i + e'_i - \ell)/2 - p_i$.

With the introduction of phase variance, a necessary and sufficient condition can be derived:

Theorem 4: External temporal consistency for O_i^B at the backup is satisfied if only if $r_i \leq \delta_i^B - v'_i - p_i - v_i - \ell$, where v_i and v'_i, are the phase variances of the tasks that updates O_i^P and O_i^B, respectively.

A proof is attached in Appendix E.

If we choose p_i to be the largest that satisfies the external temporal constraint of object i at the primary, i.e. $p_i = \delta_i^P - v_i$, we derive:

$$r_i \leq \delta_i^B - v_i' - (\delta_i^P - v_i) - v_i - \ell = (\delta_i^B - \delta_i^P) - v_i' - \ell$$

Moreover, if $v_i' = 0$, then the above formula is simplified to $r_i \leq (\delta_i^B - \delta_i^P) - \ell$. Hence, we have:

Theorem 5: If $v_i' = 0$, then external temporal consistency for object O_i^B at the backup is satisfied if only if $r_i \leq (\delta_i^B - \delta_i^P) - \ell$.

Let δ denote $\delta_i^B - \delta_i^P$, then Theorem 5 states that in order to guarantee the external temporal consistency for object O_i^B at the backup, an update message from the primary to the backup must be sent within the next $\delta - \ell$ time units after the completion of each update on the primary. This is identical to the window-consistent protocol proposed by Mehra et. al. [22], here δ is the window of inconsistency (or window consistent bound) between the primary and backup.

3 Inter-object temporal consistency

The previous section introduced the notion of external temporal consistency which deals with the relationship of an object in the external world and its images on the servers. This section presents the concept of inter-object temporal consistency which is concerned with the relationship between different objects or events. If one object is related to another object in a temporal sense, then a temporal constraint between the two objects must be maintained. For example, when an airplane takes off, there is a time bound between accelerating the plane and the lifting of the plane into air because the runway length is limited and the airplane can not keep accelerating on the runway indefinitely without lifting off. The concept of inter-object temporal consistency is illustrated in Figure 4:

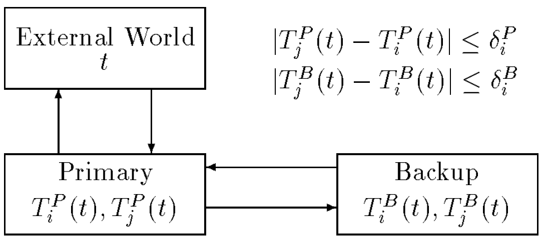

Figure 4: inter-object temporal consistency

In the following, we establish the conditions that must be met by an object pair such that their relative temporal bound is guaranteed. Let δ_{ij} denote the inter-object temporal consistency constraint between object i and j, the inter-object temporal consistency requires that inequality $|T_j^P(t) - T_i^P(t)| \leq \delta_{ij}$ to hold at both the primary and backup at all t. A sufficient condition to guarantee this is:

Lemma 3: Inter-object temporal consistency between object O_i^P and O_j^P is satisfied at the primary if $p_i \leq (\delta_{ij} + e_i)/2$, $p_j \leq (\delta_{ij} + e_j)/2$, and at the backup if $r_i \leq (\delta_{ij} + e_i')/2$, $r_j \leq (\delta_{ij} + e_j')/2$.

The correctness of the lemma is intuitive since any two consecutive invocations of one task that meets the condition is bound by the temporal constraint, any two neighboring invocations of two tasks that meet the same condition are bound by the temporal constraint too. Notice that the dealing with inter-object temporal consistency makes the update scheduling for the backup independent of that at the primary. Hence, we do not need to consider the update frequence at the primary when considering inter-object temporal consistency at the backup.

With the introduction of phase variance, a necessary and sufficient condition can be expressed as:

Theorem 6: Inter-object temporal consistency between object O_i^P and O_j^P is satisfied at the primary if only if $p_i \le \delta_{ij} - v_i$, $p_j \le \delta_{ij} - v_j$, and at the backup if only if $r_i \le \delta_{ij} - v_i'$, $r_j \le \delta_{ij} - v_j'$.

The correctness of the theorem is agian intuitive since any two consecutive invocations of one task that meets the condition is bound by the temporal constraint, any two neighboring invocations of two tasks that meet the same condition are also bound by the temporal constraint. A formal proof for this theorem is attached in Appendix F.

If the phase variacnes of the tasks that update object O_i^B and O_j^B are made zero (i.e. $v_i = v_j = v_i' = v_j' = 0$), then the conditions stated in Theorem 6 are simplified to the following:

$$p_i \le \delta_{ij}, \text{ and } p_j \le \delta_{ij} \text{ for the primary}$$
$$r_i \le \delta_{ij}, \text{ and } r_j \le \delta_{ij} \text{ for the backup}$$

which means that the inter-object temporal constraint between object i and j at both the primary and backup can be maintained by scheduling the two updates (for object i and j, respectively) within a bound of δ_{ij} time units.

4 Implementation

We have developed a prototype implementation of a real-time primary-backup (RTPB) replication service based on the temporal consistency models described in previous sections. The remainder of Section 4 describes the key features of this implementation with its performance evaluations presented in Section 5.

4.1 Implementation environment and system configuration

Our system is implemented as a user-level x-kernel based server on the MK 7.2 microkernel from the Open Group. The x-kernel is a protocol development environment which explicitly implements the protocol graph [12]. The protocol objects communicate with each other through a set of x-kernel uniform protocol interfaces. A given instance of the x-kernel can be configured by specifying a protocol graph in the configuration file. A protocol graph declares the protocol objects to be included in a given instance of the x-kernel and their relationships.

We chose the x-kernel as the implementation environment because of its several unique features. First, it provides an architecture for building and composing network protocols. Its object-oriented framework promotes reuse by allowing construction of a large network software system from smaller building blocks. Secondly, it has the capability of dynamically configuring the network software, which allows application programmers to configure the right combination of protocols for their applications. Thirdly, its dynamic architecture can adapt more quickly to changes in the underlying network technology [24].

Our system includes a primary server and a backup server. A client application resides on the same machine as the primary. The client continuously senses the environment and periodically sends updates to the primary. The client accesses the server using Mach IPC-based interface (cross-domain remote procedure call). The primary is responsible for backing up the data on the backup site and limiting the inconsistency of the data between the two sites within some specified window. The following assumptions are made in the implementation:

- Link failures are handled using physical redundancy such that network partitions are avoided.
- An upper bound exists on the communication delay between the primary and backup. Missed message

deadlines are treated as performance failures.

- Servers are assumed to suffer crash failures only.
- The underlying operating system is assumed to support priority-based scheduling.

Figure 5 shows the RTPB system architecture and x-kernel protocol stack. At the top level is the RTPB application programming interface which is used to connect the outside clients to the Mach server on one end and Mach server to the x-kernel on the other end. Our real-time primary-backup (RTPB) protocol sits right below the RTPB API layer. It serves as an anchor protocol in the x-kernel protocol stack. From above, it provides an interface between the x-kernel and the outside host operating system, the OSF Mach in our case. From below, it connects with the rest of the protocol stack through the x-kernel uniform protocol interface. The underlying transport protocol is UDP. Since UDP does not provide reliable delivery of messages, we need to use explicit acknowledgments when necessary. The primary host interacts with the backup host through the

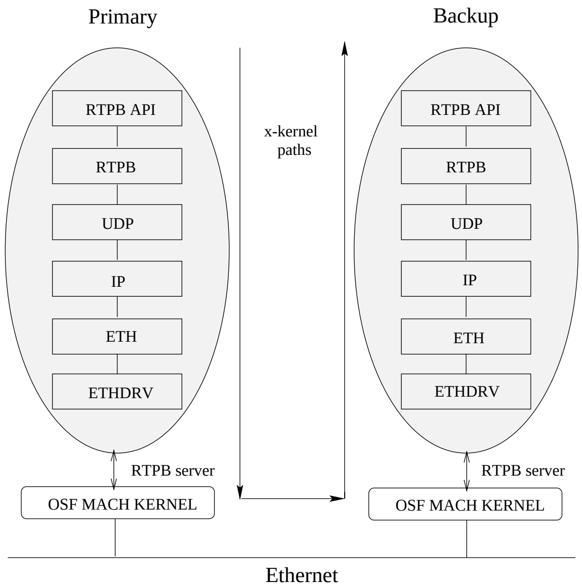

Figure 5: system architecture and protocol stack

underlying RTPB protocol that is implemented inside the x-kernel protocol stack (on top of UDP as shown in Figure 5). There are two identical versions of the client application residing on the primary and backup hosts respectively. Normally, only the primary client application is running. But when the backup takes over in case of primary failure, it also activates the backup client application and brings it up to the most recent state. The client application interacts with the RTPB system through the Mach API interface we developed for the system.

4.2 Admission control

Before a client starts to send updates of a data object to the primary periodically, it first registers the object with the service so that the primary can perform admission control to decide whether to admit the object into the service. During registration, the client reserves the necessary space for the object on the primary server and on the backup server. In addition, the client specifies the period it will update the object p_i as well as the temporal consistency allowed for the object on both the primary site and the backup site, where the temporal consistency specified by the client is relative to the external world. The consistency window between the real data object i and its copy on the primary is δ_i^P. Because the copy of object i on the primary changes only when the client sends a new update, the inconsistency between the real data and its image on the primary is dependent on the frequency of client updates. Hence, it is the responsibility of the client to send updates frequently enough to make sure the primary has a fresh copy of the data.

The primary server compares the value of δ_i^P and p_i. If $p_i \leq \delta_i^P$, then the inconsistency between the real data and the primary copy will always fall into the specified consistency window (as shown in Section 2). If the condition does not hold, the primary will not admit the object. The primary can provide feedback so that the client can negotiate for an alternative quality of service for the object.

Given the temporal consistency constraint for object i on the primary δ_i^P and the constraint on the backup δ_i^B, the consistency window for object i between the primary and the backup δ_i can be calculated as $\delta_i = \delta_i^B - \delta_i^P$. There is an upper bound, ℓ, on the communication delay between the primary and the backup. If the consistency window δ_i is less than the bound ℓ, it is impossible to maintain consistency between the primary and backup servers. Therefore, during the registration of object i, the primary will check if relation $\delta_i > \ell$ holds for object i. If the relation does not hold, the primary rejects the object.

After testing that the temporal consistency constraints hold for object i, the primary needs to further check if it can schedule a periodic update event (to the backup) for object i that will meet the consistency constraint of the object on the backup without violating the consistency constraints of all existing objects. For example, the primary will perform a schedulability test based on the rate-monotonic scheduling algorithm [20]. If all existing update tasks as well as the newly added update task for object i are schedulable, the object is admitted into the system.

Each inter-object temporal constraint is converted into two external temporal constraints according to the results derived in Section 3. Specifically, given objects i adn j and their inter-object temporal constraint δ_{ij}, their inter-object temporal bound can be met at the primary if $p_i \leq \delta_{ij}$, $p_j \leq \delta_{ij}$, and the schedulability test is successful. This same constraint can be met at the backup as long as the constraint δ_{ij} is sufficiently large such that the primary can schedule two new update tasks that periodically send update messages to the backup without violating the temporal constraints of any existing object that is registered with the replication service.

4.3 Update scheduling

In our model, client updates are decoupled from the updates to the backup. The primary needs to send updates to the backup periodically for all objects admitted in the service. It is important to schedule sufficient update transmissions to the backup for each object in order to control the inconsistency between the primary and the backup. In the absence of link failures, there is an upper bound, ℓ, on the communication delay between the primary and the backup.

For external temporal constraint, if a client modifies an object i, the primary must send an update for the object to the backup within the next $\delta_i - \ell$ time units; otherwise the object on the backup may fall out of the consistency window. In order to bound the temporal inconsistency, it is sufficient that the primary send an update for object i to the backup at least once every $\delta_i - \ell$ time units. Because UDP, the underlying transport protocol we use, does not provide reliability of message delivery, we build enough slack such that the primary can retransmit updates to the backup to compensate for potential message loss. For example, we set the period for the update task of object i as $(\delta_i - \ell)/2$ in our experiments.

For inter-object temporal constraint, the primary need not send update to the backup within the next $\delta_i - \ell$ time units after the primary is updated. But rather, the primary schedules the two updates for object i and j within δ_{ij} time units (see Section 3).

Another question is whether the backup should acknowledge each update message. We chose not to send acknowledgement messages for two reasons. First, acknowledging each update for each object introduces considerable communication overhead. Secondly, the local area network is fairly reliable under normal load. Most of the message losses occur when the network is overloaded, in which case acknowledgments and retransmissions can only make the situation worse. Retransmission is triggered by a request from the backup.

4.4 Failure detection and recovery

Failure detection and recovery is a key component of the replication service. It determines the availability of the service. The approach is that all replication servers exchange periodic messages. These messages serves as the heartbeats among those servers. In our system, both the primary and the backup have a "ping" thread which sends periodic messages to the other server. Each server acknowledges the "ping" message from the other one. If a server receives no acknowledgment over some time, it will timeout and resend a "ping" message. If there is no response beyond a certain amount of time, the server will declare the other end dead. If the backup is dead, the primary cancels the "ping" messages as well as update events for each registered object. If the primary crashes, the backup takes over as the new primary. The new primary changes the address in the name file to its own internet address, invokes a backup version of the client application at the local machine, feeds the new client with information stored in its memory by an up call, starts listening to all client requests, and then waits to recruit a new backup. The new client replaces the client at the crashed machine to perform the sensing task. Our implementation supports the integration of a new backup after a failure is detected.

5 System performance

This section summarizes the results of a detailed performance evaluation of the RTPB replication service introduced in this paper. The prototype evaluation considers several performability metrics:

- Response time with/without admission control.
- Average maximum primary-backup distance.
- Average duration of backup inconsistency

These metrics are influenced by several parameters, including client write rate, object size, number of objects being accepted, window size, communication failure, and scheduling compression.

All graphs in this section illustrate both the external temporal consistency and inter-object temporal consistency. Each inter-object temporal constraint is converted into two external temporal constraints with the external temporal constraint being replaced by the inter-object temporal constraint.

5.1 Client response time within RTPB

To show that our admission control process is functioning correctly, we measured the system response time to client requests under two different conditions with and without admission control. Figure 6 shows the

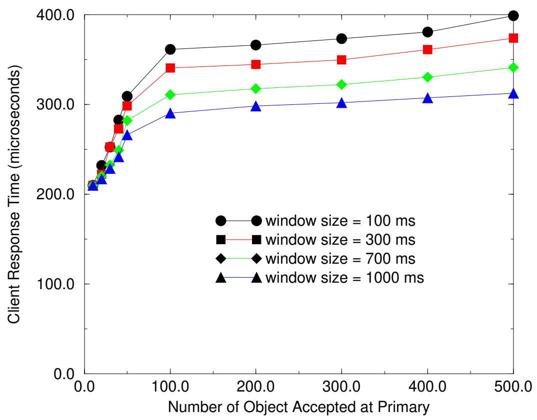

Figure 6: client response time with admission control

client response time as a function of the number of objects being accepted by the admission control process running at primary. As shown in the graph, the number of objects has little impact on the response time of the system. This is due to the fact that the admission control in the primary acts as a gate keeper that prevents too many objects from being admitted. The decoupling of client updates from backup updates is also a major contributing factor. Figure 7 shows the same metric without admission control. As we can see from the graph,

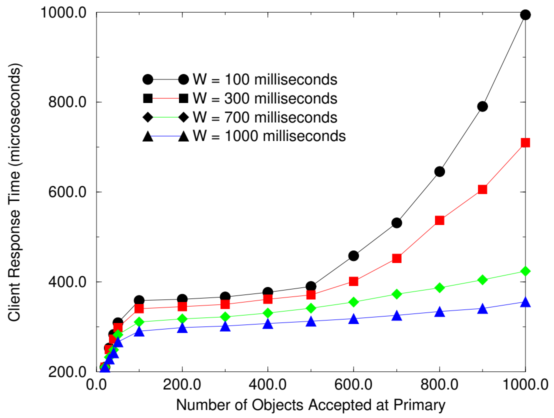

Figure 7: response time without admission control

the number of objects has little impact on the response time of the system when it is within the allowable limit of the window size. But the response time increases dramatically if the number of objects exceeds the maximum allowable number of objects under a given window size. By disabling admission control, the number of objects accepted into the system can go out of control which consequently degrade system performance.

For the same number of objects in Figure 6 and 7, larger window sizes ensure a better response time. The larger the window size, the more leeway the primary has (due to the decoupling in RTPB) in scheduling update messages to the backup. Hence, the primary is able to schedule local updates and client responses while delaying transmission of updates to the backup.

5.2 Primary-backup distance

Since the relaxation of consistency constraint between the primary and the backup can potentially introduce data inconsistency, it is important to measure the average maximum distance between the primary and the backup. Figure 8 shows the average maximum distance as a function of the probability of message loss from the primary to the backup. The graph illustrates average maximum distance for various client update rates.

From the figure, we see that the average maximum distance between the primary and the backup is close to zero when there is no message loss. However, as message loss rate goes up, the distance also increases. For example, when the message loss is 10 percent, the average maximum distance between the primary and the backup approaches 700 milliseconds. In general, the distance increases as message loss rate or client write rate increases.

It can be inferred that the average maximum distance can be reduced by compensating for message loss. It is desirable to build our real-time primary-backup protocol on top of a transport protocol which provides reliable and timely message delivery. In our experiments, the primary sends updates twice as often as necessary to compensate for potential message loss.

Figure 9 and 10 measure the average maximum distance between the primary and backup as a function of the number of objects being accepted at primary. As shown in Figure 9, when admission control is used,

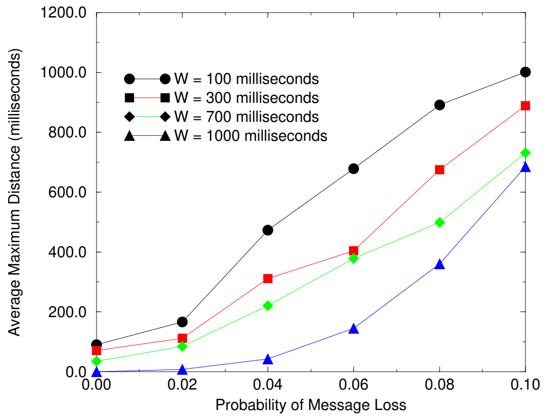

Figure 8: average maximum primary/backup distance

the number of objects has little impact on the average maximum distance between the primary and the backup. The admission control in the primary acts as a gate keeper that prevents too many objects from being admitted. Hence, it guarantees that the average maximum distance for admitted objects is minimized.

However, as shown in Figure 10, if the admission control process is disabled, then the number of objects being admitted into the RTPB system can exceed the maximum allowable number of objects for a particular window size. This results in an increase in the average maximum distance between the primary and the backup. The comparison of Figure 9 and 10 demonstrates the need for an admission control policy.

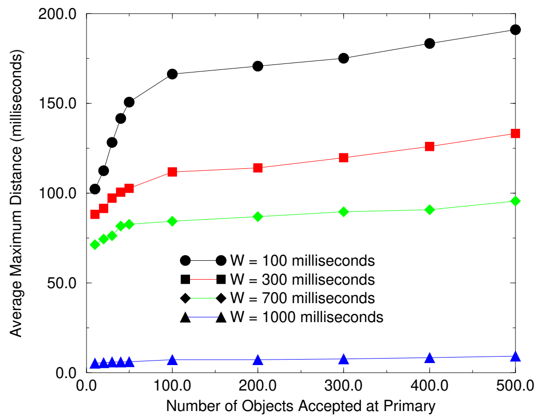

Figure 9: average maximum primary/backup distance with admission control

5.3 Duration of backup inconsistency

Another interesting metrics measures the duration of inconsistency at the backup. Figure 11 and 12 show the duration of backup inconsistency as a function of the probability of message loss between the primary

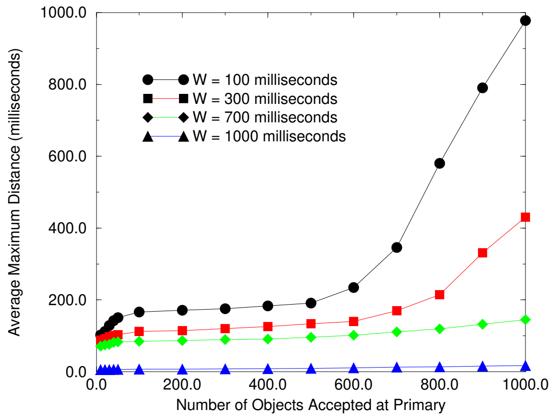

Figure 10: average maximum primary/backup distance without admission control

and the backup. The difference between these two graphs is that Figure 11 shows the result under normal scheduling while Figure 12 shows the result under compressed scheduling (primary schedules as many updates to backup as the resources allow [22]). The figures show that under both normal and compressed scheduling,

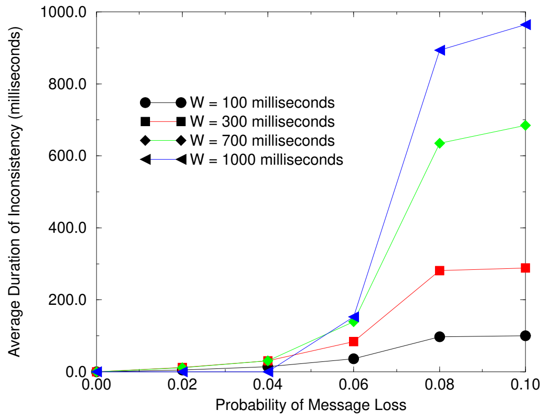

Figure 11: duration of backup inconsistency

the larger the probability of message loss, the longer the backup stays in an inconsistent state from the primary. However, for the same window size, the results for compressed scheduling are different from normal scheduling. For normal scheduling, the larger the window size, the longer the backup stays in an inconsistent state. But for compressed scheduling, the effect of window size on the duration of backup inconsistency is just the opposite. If an update message is lost, the backup would stay inconsistent until the next update message comes. Since the frequency of update message is determined by window size under normal scheduling, a larger window size would mean longer duration of backup inconsistency. However, under compressed scheduling, the frequency of update messages is not determined by window size but by the capacity of the CPU resource at the primary. Therefore, larger window size would mean shorter duration of backup inconsistency because the

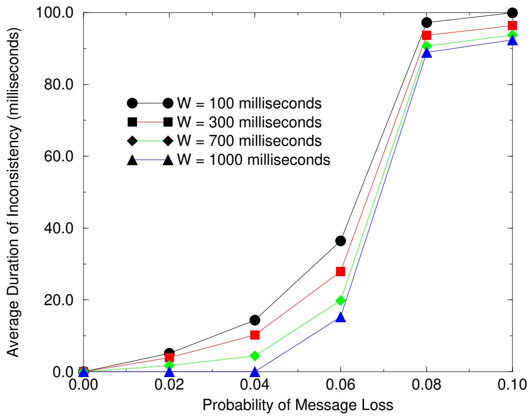

Figure 12: duration of backup inconsistency with compressed scheduling

update frequency at the backup is much higher than the update frequency at primary.

6 Related work

6.1 Replication models

One can structure a distributed system as a collection of servers that provide services to the outside clients or other servers within the system. A common approach to building fault-tolerant distributed systems is to replicate servers that fail independently. The objective is to give the clients the illusion of service that is provided by a single server. The main approaches for structuring fault-tolerant servers are *passive* and *active* replication. In passive replication schemes [3, 4, 23], the system state is maintained by a primary and one or more backup servers. The primary communicates its local state to the backups so that a backup can take over when a failure of the primary is detected. This architecture is commonly called the *primary-backup* approach and has been widely used in building commercial fault-tolerant systems. In active replication schemes [5–7], also known as the *state-machine approach*, a collection of identical servers maintain replicated copies of the system state. Updates are applied atomically to all the replicas so that after detecting the failure of a server, the remaining servers continue the service. Schemes based on passive replication tend to require longer recovery time since a backup must execute an explicit recovery algorithm to take over the role of the primary. Schemes based on active replication, however, tend to have more overhead in responding to client requests since an agreement protocol must be performed to ensure atomic ordered delivery of messages to all replicas.

Past work on synchronous and asynchronous replication protocols has focused, in most cases, on applications for which timing predictability is not a key requirement. Real-time applications, however, operate under strict timing and dependability constraints that require the system to ensure timely delivery of services and to meet certain consistency constraints. Hence, the problem of server replication posses additional challenges in a real-time environment. In recent years, several experimental projects have begun to address the problem of replication in distributed hard real-time systems. For example, MARS [11] is a time-triggered distributed real-time system: its architecture is based on the assumption that the worst-case load is determined *apriori* at design time, and the system response to external events is cyclic at predetermined time-intervals. The MARS architecture provides fault-tolerance by implementing active redundancy. A Fault-Tolerant Unit (FTU) in a MARS system consists of a collection of replicated components operating in active redundancy. A component, consisting of a node and its application software, relies on a number of hardware and software mechanisms for error detection to ensure a fail-silent behavior.

RTCAST [1] is a lightweight fault-tolerant multicast and membership service for real-time process groups which exchange periodic and aperiodic messages. The service supports bounded-time message transport, atomicity, and order for multicasts within a group of communicating processes in the presence of processor crashes and communication failures. It guarantees agreement on membership among the communicating processors, and ensures that membership changes resulting from processor joins or departures are atomic and ordered with respect to multicast messages. Both MARS and RTCAST are based on active replication whereas RTPB is a passive scheme.

Rajkumar [8, 26] present a publisher/subscriber model for distributed real-time systems. It provides a simple user interface for publishing messages on a logical "channel", and for subscribing to selected channels as needed by each application. In the absence of faults each message sent by a publisher on a channel should be received by all subscribers. The abstraction hides a portable, analyzable, scalable and efficient mechanism for group communication. It does not, however, attempt to guarantee atomicity and order in the presence of failures, which may compromise consistency.

6.2 Consistency semantics

The approach proposed in this paper bounds the overhead by relaxing the requirements on the consistency of the replicated data. For a large class of real-time applications, the system can recover from a server failure even though the servers may not have maintained identical copies of the replicated state. This facilitates alternative approaches that trade atomic or causal consistency amongst the replicas for less expensive replication protocols. Enforcing a weaker correctness criterion has been studied extensively for different purposes and application domains. In particular, a number of researchers have observed that serializability is too strict as a correctness criterion for real-time databases. Relaxed correctness criteria facilitate higher concurrency by permitting a limited amount of inconsistency in how a transaction views the database state [10, 13, 15–19, 25, 27].

For example, a recent work [16] [15] proposed a class of real-time data access protocols called SSP (Similarity Stack Protocol) applicable to distributed real-time systems. The correctness of the SSP protocol is justified by the concept of *similarity* which allows different but sufficiently timely data to be used in a computation without adversely affecting the outcome. Data items that are similar would produce the same result if used as input. SSP schedules are deadlock-free, subject to limited blocking and do not use locks. Furthermore, a schedulability bound can be given for the SSP scheduler. Simulation results show that SSP is especially useful for scheduling real-time data access on multiprocessor systems.

Similarly, the notion of imprecise computation [21] explores weaker application semantics and guarantees timely completion of tasks by relaxing the accuracy requirements of the computation. This is particularly useful in applications that use discrete samples of continuous time variables, since these values can be approximated when there is not sufficient time to compute an exact value. Weak consistency can also improve performance in non-real-time applications. For instance, the quasi-copy model permits some inconsistency between the central data and its cached copies at remote sites [2].This gives the scheduler more flexibility in propagating updates to the cached copies. In the same spirit, the RTPB replication service allows computation that may otherwise be disallowed by existing active or passive protocols that support atomic updates to a collection of replicas.

7 Conclusions and future work

The conclusions can be draw about RTPB are:

- Simple elegant approach for fault tolerance.
- Fast response to clients (due to consistency relaxation and decoupling of client writes/backup updates).
- Temporal consistent with regard to both the external world and the inter-object relationships.
- Controlled inconsistency between primary/backup.

The principle of our RTPB model is simple and easy to understand. The interaction between the primary and clients as well as between the primary and backup are well defined. Our RTPB replication service can provide fast response to client demands due to relaxation of the temporal constraint placed between the primary and the backup, and the decoupling of client updates from backup updates. Yet the system is still able to maintain a controlled inconsistency (window consistency) between the primary and backup.

Avenues for future study include: optimization of scheduling update messages from the primary to the backup; support for multiple backups; application of external and inter-object temporal consistency to active replication; and hybrid active/passive replication schemes for real-time systems.

Acknowledgement

The authors of this paper wish to thank Zhiqun Wang for her contribution to the prototype development of RTPB.

References

[1] Tarek Abdelzaher, Anees Shaikh, Scott Johnson, Farnam Jahanian, and Kang Shin. Rtcast: Lightweight multicast for real-time process groups. In *IEEE Real-Time Technology and Applications Symposium*, pages 250–259, June 1996.

[2] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transaction on Database Systems*, 15(3):359–384, September 1990.

[3] P. Alsberg and J. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the IEEE, Int'l Conf. on Software Engineering*, 1976.

[4] J.F. Bartlett. Tandem: A non-stop kernel. In *ACM Operating System Review*, volume 15, 1991.

[5] K.P. Birman. The process group approach to reliable distributed computing. Technical Report TR 91-1216, Cornell University, July 1991.

[6] F.B.Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Transaction on Computing Surveys*, 22(4):299–319, December 1990.

[7] F.Cristian, B.Dancy, and J.Dehn. Fault-tolerance in the advanced automation system. In *Proceedings Int'l Symp. on Fault-Tolerant Computing*, pages 160–170, June 1990.

[8] M. Gagliardi, R.Rajkumar, and L. Sha. Designing for evolvability: Building blocks for evolvable real-time systems. In *Proc. Real-Time Technology and Applications Symposium*, pages 100–109, june 1996.

[9] Ching-Chih Han and Kwei-Jay Lin. Scheduling distance-constrained real-time tasks. In *Proceedings Real-Time Systems Symposium*, December 1992.

[10] H.F.Korth, N.Soparkar, and A. Silberschatz. Triggered real-time databases with consistency constraints. In *Proc. Int'l Conf. on Very Large Data Bases*, August 1990.

[11] H.Kopetz, A.Damm, C.Koza, M. Mulazzani, W.Schwabl, C.Senft, and R.Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. In *Proceedings of the IEEE Micro*, pages 25–40, February 1989.

[12] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[13] B. Kao and H. Garcia-Molina. An overview of real-time database systems. In S.H. Son, editor, *Advances in Real-Time systems*, pages 463–486. Prentice Hall, 1995.

[14] H. Kopetz and P. Verissimo. Real time and dependability concepts. In Sape Mullender, editor, *Distributed Systems*, chapter 16, pages 411–446. Addison-Wesley, 2 edition, 1993.

[15] Tei-Wei Kuo, D. Locke, and F. Wang. Error propagation analysis of real-time data intensive application. In *IEEE Real-Time Technology and Applications Symposium*, June 1997.

[16] Tei-Wei Kuo and Aloysius K. Mok. Ssp: a semantics-based protocol for real-time data access. In *Proceedings of IEEE 14th Real-Time Systems Symposium*, December 1993.

[17] Tei-Wei Kuo and Aloysius K. Mok. Real-time database - similarity semantics and resource scheduling. In *ACM SIGMOD Record*, March 1997.

[18] K-J Lin. Consistency issues in real-time database systems. In *Proc. 22nd Hawaii International Conference on System Sciences*, pages 654–661, January 1989.

[19] Kwei-Jay Lin and Farnam Jahanian. Issues and applications. In Sang Son, editor, *Real-time Database Systems*. Kluwer Academic Publishers, 1997.

[20] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[21] J.W.S. Liu, W.-K. Shih, and K.-J. Lin. Imprecise computation. In *Proceedings of IEEE*, volume 82, pages 83–94, Jan 1994.

[22] Ashish Mehra, Jennifer Rexford, and Farnam Jahanian. Design and evaluation of a window-consistent replication service. In *IEEE Transaction on Computer*, 1997.

[23] N.Budhiraja, K.Marzullo, F.B.Schneider, and S.Toueg. Primay-backup protocols: Lower bounds and optimal implementations. In *Proceedings of IFIP Working Conference on Dependable Computing*, pages 187–198, 1992.

[24] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[25] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *Proc. ACM SIGMOD*, pages 377–386, May 1991.

[26] R. Rajkumar, M. Gagliardi, and L. Sha. The real-time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation. In *Proc. Real-Time Technology and Applications Symposium*, pages 66–75, May 1995.

[27] S.B.Davidson and A. Watters. Partial computation in real-time database systems. In *Proc. Workshop on Real-Time Operating Systems and Software*, pages 117–121, May 1988.

Appendix

A Proof of Lemma 1

Proof: In the worst case, a version of object O_i^P at the primary could stay current for as long as $2p_i - e_i$ time units before the completion of the next update, hence, at any time instant t, we have:

$$t - T_i^P(t) \leq 2p_i - e_i \leq 2 \cdot (\delta_i^P + e_i)/2 - e_i = \delta_i^P \quad \Box$$

B Proof of Theorem 1

Proof: Without loss of generality, assume $v_i = v_i^k = |I_k - I_{k-1} - p_i|$.

Sufficient: There are two cases:

(1) $I_k - I_{k-1} < p_i$, then:
$t - T_i^P(t) \leq I_k - I_{k-1} < p_i \leq \delta_i^P - v_i \leq \delta_i^P$

(2) $I_k - I_{k-1} \geq p_i$, then:
$t - T_i^P(t) \leq I_k - I_{k-1} = v_i + p_i \leq v_i + (\delta_i^P - v_i) = \delta_i^P$

Thus, the condition is sufficient.

Necessary: We proceed by contradiction. Suppose $p_i > \delta_i^P - v_i$. Then for $I_k - I_{k-1} \geq p_i$, we have:

$$I_k - I_{k-1} = v_i + p_i > v_i + (\delta_i^P - v_i) = \delta_i^P$$

Hence, the temporal constraint could be violated. Thus, the condition is also necessary. □

C Proof of Theorem 2

Proof: We only prove the formula under EDF. The formula under RM can be proved similarly.

when the utilization rate is x, we have:

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} + \cdots + \frac{e_n}{p_n} = x$$

Divide the equation by x, we get:

$$\frac{e_1}{xp_1} + \frac{e_2}{xp_2} + \cdots + \frac{e_n}{xp_n} = 1$$

Hence if we shrink the period of each task by a factor of x, the resulting task set is still schedulable under EDF. According to Inequality 2.1, we get: $v_i \leq xp_i - e_i$. □

D Proof of Lemma 2

Proof: In the worst case, a version of object O_i could stay at the primary for a maximum of $2p_i - e_i$ time units before it is sent to the backup; the message takes ℓ time units to reach the backup; and the update occurs at the backup after the message arrival; and can stay current at the backup for a maximum of $2r_i - e'_i$ time units. Thus, at any time instant t, we have:

$t - T_i^B(t) \leq 2p_i - e_i + \ell + 2r_i - e'_i \leq 2p_i - e_i + \ell + 2(\frac{\delta_i^B + e_i + e'_i - \ell}{2} - p_i) - e'_i = \delta_i^B$ □

E Proof of Theorem 4

Sufficient: In the worst case, a version of object O_i could stay at the primary for a maximum of $p_i + v_i$ time units before it is sent to the backup; the message takes ℓ time units to reach the backup; and the update occurs at the backup after the message's arrival; and can stay current at the backup for a maximum of $r_i + v'_i$, time units. Thus, at any time instant t, we have:

$t - T_i^B(t) \leq r_i + v'_i + p_i + v_i + \ell$

Apply the given condition in Theorem 4 to the above formula, we have:

$t - T_i^B(t) \leq \delta_i^B - v'_i - p_i - v_i - \ell + v'_i + p_i + v_i + \ell = \delta_i^B$

Therefore, our condition in Theorem 4 is sufficient.

Necessary: We proceed by contradiction. Suppose $r_i > \delta_i^B - v'_i - p_i - v_i - \ell$. Then we only need to construct a situation under which the external temporal consistency for object O_i^B at the backup is broken.

We construct our case as follows: first, let $v_i = v'_i = 0$. Then we have $r_i > \delta_i^B - p_i - \ell$ from the assumption. Next, we find some maximum positive values Δ_1 and Δ_2 such that: $0 < \Delta_1 < e_i, 0 < \Delta_2 < e_j$

Now, let's consider the worst case scenario under the above assumptions. An update on the primary could stay for a maximum of $p_i + d_1$ time units (remember $v_i = 0$) before it is sent to the backup; the update message takes ℓ time units to reach the backup; the update happens at the backup after the message's arrival; and can stay current at the backup for a maximum of $r_i + \Delta_2$ time units (remember $v'_i = 0$). Therefore, we have:

$$t - T_i^B(t) = p_i + \Delta_1 + \ell + r_i + \Delta_2 > p_i + \Delta_1 + \ell + (\delta_i^B - p_i - \ell) + \Delta_2 = \delta_i^B + \Delta_1 + \Delta_2 > \delta_i^B$$

Thus, the external temporal consistency for object O_i^B at the backup is broken. Therefore, our condition in Theorem 4 is also necessary. $\square$

F Proof of Theorem 6

Here we only prove the case involving the primary. The case for the backup can be similarly derived.

Sufficient: Use the result of Theorem 1's proof:
$0 \le t - T_i^P(t) \le p_i + v_i$
$0 \le t - T_j^P(t) \le p_j + v_j$

With simple operations, we can get:
$-(p_i + v_i) \le T_i^P(t) - T_j^P(t) \le p_j + v_j$
$-(p_j + v_j) \le T_j^P(t) - T_i^P(t) \le p_i + v_i$

Combine the above results, we have:
$|T_j^P(t) - T_i^P(t)| \le \max(p_i + v_i, p_j + v_j)$

Apply the given condition in Theorem 6, we have:

$$|T_j^P(t) - T_i^P(t)| \le \max(\delta_{ij} - v_i + v_i, \delta_{ij} - v_j + v_j) = \delta_{ij}$$

Therefore, our condition in Theorem 6 is sufficient.

Necessary: We proceed by contradiction. Suppose $p_i > \delta_{ij} - v_i$. Then we need to show a situation under which the inter-object temporal consistency between object O_i^P and O_j^P is violated.

We construct the scenario as follows: first, let $v_i = 0$, then we have: $p_i > \delta_{ij}$ from the assumption. Then there exists a value Δ such that: $p_i > \delta_{ij} + \Delta$ and $\Delta > 0$. Now, there are two cases to be considered:

Case 1: $p_i \le p_j$. We do the following:

let the initial phase of the task that updates O_j^P be 0, the phase of the first invocation of the task that updates O_i^P that extends pass time instant p_j be $p_j + e_j - e_i - \Delta$.

Then we pick time instant $t = p_j + e_j - \Delta$. We have: $T_i^P(t) = p_j + e_j - \Delta$ (because the latest invocation of the task that updates O_i^P was started at $p_j + e_j - e_i - \Delta$, and thus must be finished on $p_j + e_j - e_i - \Delta + e_i = p_j + e_j - \Delta$). We also have $T_j^P(t) = e_j$ (because the first invocation of the task that updates O_j^P was started at 0, and therefore must be finished by e_j, and its second invocation can NOT be finished by $p_j + e_j - \Delta < p_j + e_j$). Therefore, we have:

$$T_j^P(t) - T_i^P(t) = p_j + e_j - \Delta - e_j = p_j - \Delta \ge p_i - \Delta > \delta_{ij} + \Delta - \Delta = \delta_{ij}$$

Thus, the inter-object temporal consistency between object O_i^P, O_j^P at primary is broken.

Case 2: $p_i > p_j$. We do the following:

let the initial phase of the task that updates O_i^P be 0, and the phase of the first invocation of the task that updates O_j^P that extends pass time instant p_i be $p_i + e_i - e_j - \Delta$.

Then we pick time instant $t = p_i + e_i - \Delta$. We have: $T_j^P(t) = p_i + e_i - \Delta$ (because the latest invocation of the task that updates O_j^P was started at $p_i + e_i - e_j - \Delta$, and thus must be finished on $p_i + e_i - e_j - \Delta + e_j = p_i + e_i - \Delta$). We also have $T_i^P(t) = e_i$ (because the first invocation of the task that updates O_i^P was started at 0, and therefore must be finished by e_i, and its second invocation can NOT be finished by $p_i + e_i - \Delta < p_i + e_i$)

Therefore, we have:

$$T_j^P(t) - T_i^P(t) = p_i + e_i - \Delta - e_i = p_i - \Delta > \delta_{ij} + \Delta - \Delta = \delta_{ij}$$

Again, the inter-object temporal consistency between object O_i^P and O_j^P at the primary is broken.

We can show similar results under assumption $p_j > \delta_{ij} - v_j$ or $p_i > \delta_{ij} - v_i$ and $p_j > \delta_{ij} - v_j$ Hence, our condition in Theorem 6 is also necessary. $\square$