

# 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

*This paper presents a real-time primary-backup (RTPB) replication scheme for supporting fault-tolerant real-time applications. It 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 [13], and Distance-Constrained Scheduling [9]. A RTPB implementation was developed within the x-kernel architecture on the MK 7.2 microkernel and the results of a detailed performance evaluation is also discussed.*

## 1 Introduction

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 *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 relaxation 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 [14].

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 *temporal consistency*, which is the consistency view seen from the perspective of the time continuum. Two common types of temporal consistency are *external temporal consistency* which deals with the relationship between an object of the external world and its image on the servers, and *inter-object temporal consistency* which is concerned with the relationship between different objects or events.

External temporal consistency at the primary

---

The work reported in this paper was supported in part by the Advanced Research Projects Agency, monitored by the US Air Force Rome Laboratory under Grant F30602-95-1-0044

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. [15] 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<sup>1</sup>. 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 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. Since the problem of enforcing a temporal bound on an object/image pair is essentially equivalent to that of enforcing such a bound between any two successive updates of the object on the server, the guaranteeing of an object's external temporal consistency is transformed to the guaranteeing of a temporal bound between any two successive updates of this object on the server. 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  $t$ . Hence, our task becomes to find the condition that make these two inequalities hold. Let:

$O_i^P, O_i^B$  denote object  $i$  at the primary and backup.  
 $T_i^P(t), T_i^B(t)$  denote timestamp of  $O_i^P, O_i^B$  at time  $t$ .  
 $p_i$  denote the period of the task that updates  $O_i^P$ .  
 $I_i^k$  denote the  $k$ -th invocation of the task updating  $O_i^P$ .  
 $e_i$  denote the execution time of the task updating  $O_i^P$ .  
 $r_i$  denotes the period of the task that updates  $O_i^B$ .  
 $e'_i$  denotes the execution time of the task updating  $O_i^B$ .  
 $\ell$  is the communication delay from primary to backup.  
 $\delta_i^P, \delta_i^B$  denote temporal constraint for  $O_i^P, O_i^B$ .

Then we have:

**Lemma 1:**  $t - T_i^P(t) \leq \delta_i^P$  holds if  $p_i \leq (\delta_i^P + e_i)/2$ .  
 $t - T_i^B(t) \leq \delta_i^B$  holds if  $r_i \leq (\delta_i^B + e_i + e'_i - \ell)/2 - p_i$ .

The first condition is correct because any two consecutive invocations are bounded by  $\delta_i^P + e_i$ , the difference between any time instant  $t$  and the last update before  $t$  is within  $\delta_i^P$ . The correctness of the second condition can be argued similarly. Due to space limitation, we omit the proof in the paper. Refer to the extended version [19] for all the proofs of the lemmas and theorems in this paper.

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

**Definition 1:** The  $k$ -th phase variance  $v_i^k$  of a task is the absolute difference between the time gap of its

<sup>1</sup>formerly known as the Open Software Foundation (OSF)

$k$ -th and  $(k-1)$ -th invocations, and its period  $p_i$ , i.e.  $v_i^k = |(I_i^k - I_i^{k-1}) - p_i|$ ,  $k = 1, 2, \dots, \infty$ .

Figure 1 shows the concept.

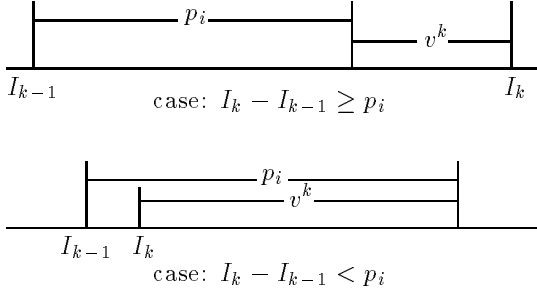


Figure 1: two cases of the  $k$ -th phase variance

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

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:

$$v_i = \max(|(I_i^k - I_i^{k-1}) - p_i|) \leq p_i - e_i \quad (2.1)$$

Similarly, we can define the phase variance on the backup  $v_i'$ . Now, we can derive a necessary and sufficient condition that guarantees external temporal consistency for an object at the primary and backup. Recall that the guaranteeing of temporal consistency is equivalent to that of ensuring inequalities  $t - T_i^P(t) \leq \delta_i^P$  and  $t - T_i^B(t) \leq \delta_i^B$  hold, we have:

**Theorem 1:**  $t - T_i^P(t) \leq \delta_i^P$  holds if only if  $p_i \leq \delta_i^P - v_i$ . And  $t - T_i^B(t) \leq \delta_i^B$  holds if only if  $r_i \leq \delta_i^B - v_i' - p_i - v_i - \ell$

### 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  $v_i \leq (x \cdot p_i)/(n(2^{1/n} - 1)) - e_i$  are satisfiable under EDF and Rate Monotonic Algorithm [13], respectively. Here  $n$  is the number of tasks,  $x$  is the utilization rate.

From Theorem 1 and 2, we see that the restriction on an object can be 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 for this case is straightforward 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].

Consider a task set  $T = T_1, T_2, \dots, T_n$ . 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 if  $\sum_{i=1}^n e_i/p_i \leq n(2^{1/n} - 1)$

Combining with the result of Theorem 1, one can show that under the condition stated in Theorem 3, 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.

If we choose  $p_i$  to be the largest that satisfies the external temporal constraint of object  $i$  at the primary, i.e.  $p_i = \delta_i^P - v_i$ , then from Theorem 1,  $t - T_i^B(t) \leq \delta_i^B$  holds if only if  $r_i \leq (\delta_i^B - \delta_i^P) - v_i' - \ell$ . Moreover, if  $v_i' = 0$ , the condition is simplified to  $r_i \leq (\delta_i^B - \delta_i^P) - \ell$  which means 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  $\delta_i^B - \delta_i^P - \ell$  time units after the completion of each update on the primary. This is identical to the window-consistent protocol proposed by Mehra et. al. [15], here  $\delta_i^B - \delta_i^P$  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.

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

**Lemma 2:** Inequality  $|T_j^P(t) - T_i^P(t)| \leq \delta_{ij}$  holds 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 lemma is correct because 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. Note that the dealing of inter-object temporal constraint makes the update scheduling for the backup independent of that at the primary. Hence, we can ignore the update frequency 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 4:**  $|T_j^P(t) - T_i^P(t)| \leq \delta_{ij}$  holds at the primary if only if  $p_i \leq \delta_{ij} - v_i$ ,  $p_j \leq \delta_{ij} - v_j$ , and at the backup if only if  $r_i \leq \delta_{ij} - v'_i$ ,  $r_j \leq \delta_{ij} - v'_j$ .

Again, the theorem is correct by the same argument stated above. If the phase variances of all the concerned tasks are made zero (i.e.  $v_i = v_j = v'_i = v'_j = 0$ ), then the conditions stated in Theorem 4 are simplified to the following:

$$p_i \leq \delta_{ij}, \text{ and } p_j \leq \delta_{ij} \text{ for the primary}$$

$$r_i \leq \delta_{ij}, \text{ and } r_j \leq \delta_{ij} \text{ for the backup}$$

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  $\delta_{ij}$  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 [11]. 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.

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 2 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

## 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  $\delta_i^P$ . 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  $\delta_i^P$  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. Each inter-object temporal constraint is converted into two external temporal constraints according to the results derived in Section 3. Specifically, given objects  $i, j$  and their inter-object temporal constraint  $\delta_{ij}$ , 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. The temporal constraint for object  $i$  on the backup is also checked to ensure that it can be met by using the conditions derived in previous sections.

After testing that the temporal consistency constraints hold for object  $i$ , the primary need 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 [13]. 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.

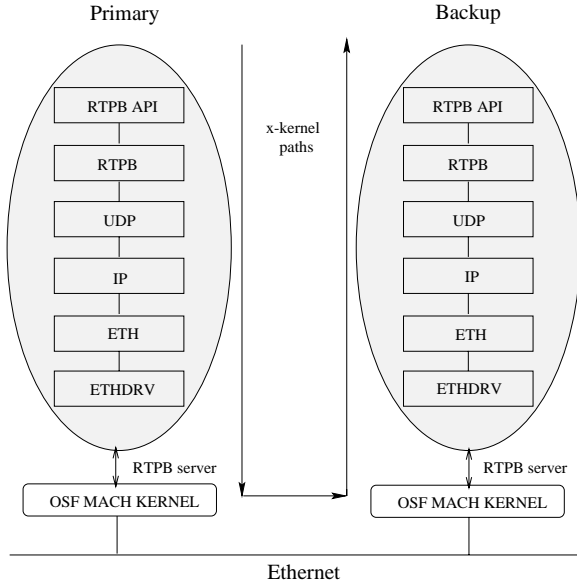


Figure 2: system architecture and protocol stack

backup host through the underlying RTPB protocol that is implemented inside the  $x$ -kernel protocol stack (on top of UDP as shown in Figure 2). 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.

### 4.3 Update scheduling

In our model, client updates are decoupled from the updates to the backup. The primary need 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,  $\ell$ , 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  $\delta_{ij}$  time units (see Section 3).

### 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 serve 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 3(a)

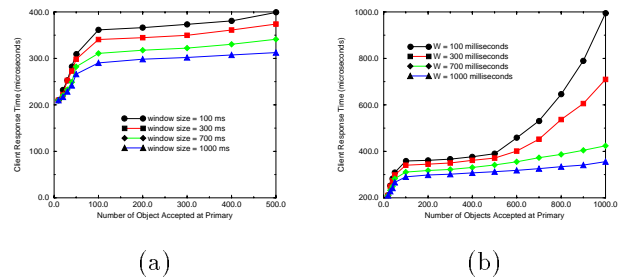


Figure 3: response time with/out admission control

shows the 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 3(b) measured the same metric without admission control and found that the response time increases dramatically when the number of objects being admitted exceeds the allowable limit of the window size.

For the same number of objects, 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 the 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 4(a) shows the average

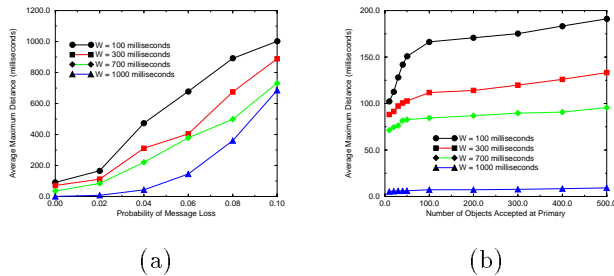


Figure 4: average maximum primary/backup distance

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 4(b) measures the same metrics as a function of the number of objects being accepted at primary with admission control. It can be seen that 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. We also experimented with admission control disabled and found that the average maximum distance increases rapidly when the number of objects being admitted exceeds the maximum allowable limit of a particular window size. This result demonstrates the need for an admission control policy.

## 5.3 Duration of backup inconsistency

Another interesting metric measures the duration of inconsistency at the backup. Figure 5(a) and (b) show the duration of backup inconsistency as a function of the probability of message loss between the primary and the backup. The difference between these two graphs is that Figure 5(a) shows the result under normal scheduling while Figure 5(b) shows the result under compressed scheduling (primary schedules as many updates to backup as the resources allow [15]). The figures show that under both normal

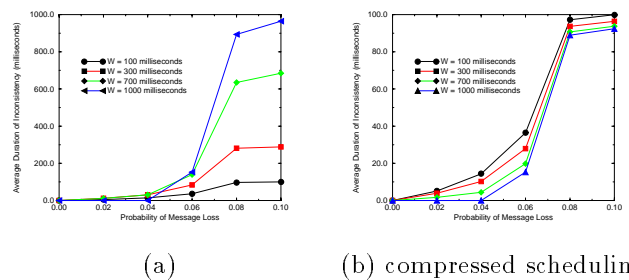


Figure 5: duration of backup inconsistency

and compressed scheduling, 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 update frequency at the backup is much higher than that at the primary.

## 6 Related work

### 6.1 Replication models

One can structure a distributed system as a collection of servers that provide services to 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* [2, 3, 16] and *active* [4, 6, 7] replication.

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 poses 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, TTP [10] is a time-triggered distributed real-time system: its architecture is based on the assumption that the worst-case load is determined *a priori* at design time, and the system response to external events is cyclic at predetermined time-intervals.

RTCAST [18] 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 com-

municating processors, and ensures that membership changes resulting from processor joins or departures are atomic and ordered with respect to multicast messages. Both TTP and RTCAST are based on active replication whereas RTPB is a passive scheme.

Rajkumar [8, 17] 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. A detailed exposition of related work in real-time database is beyond the scope of this paper. Please refer to [5, 12].

Similarly, the notion of imprecise computation [14] 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 [1]. 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 proposed RTPB model provides a simple and elegant approach for supporting fault tolerant real-time systems. It can provide fast response to client demands due to relaxation of the temporal constraints and the decoupling of client updates from backup updates. Yet the system is still able to maintain a controlled inconsistency between the primary and backup.

Avenues for future study include: optimization of RTPB; 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 wish to thank Zhiqun Wang for her contribution to the RTPB prototype development.

## References

- [1] 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.
- [2] P. Alsberg and J. Day. A principle for resilient sharing of distributed resources. In *Proc. IEEE, Int'l Conf. on Software Engineering*, 1976.
- [3] J.F. Bartlett. Tandem: A non-stop kernel. In *ACM Operating System Review*, volume 15, 1991.
- [4] K.P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, December 1993.
- [5] B.Purimetla, R. M.Sivasankaran, K.Ramamritham, and J.A.Stankovic. Real-time databases: Issues and applications. In Sang Hyuk Son, editor, *Advances in Real-Time Systems*, chapter 20. Prentice Hall, 1 edition, 1995.
- [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*, June 1996.
- [9] C-C Han and K-J Lin. Scheduling distance-constrained real-time tasks. In *Proc. Real-Time Systems Symposium*, December 1992.
- [10] H.Kopetz and G. Grunsteidl. Ttp - a protocol for fault-tolerant real-time systems. In *IEEE Computer*, volume 27, pages 14–23, January 1994.
- [11] 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.
- [12] K-J Lin and F. Jahanian. Issues and applications. In Sang Son, editor, *Real-time Database Systems*. Kluwer Academic Publishers, 1997.
- [13] 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.
- [14] J.W.S. Liu, W.-K. Shih, and K.-J. Lin. Imprecise computation. In *Proceedings of IEEE*, volume 82, pages 83–94, Jan 1994.
- [15] A. Mehra, J. Rexford, and F. Jahanian. Design and evaluation of a window-consistent replication service. In *IEEE Transaction on Computer*, 1997.
- [16] N.Budhiraja, K.Marzullo, F.B.Schneider, and S.Toueg. Primary-backup protocols: Lower bounds and optimal implementations. In *Proceedings of IFIP Working Conference on Dependable Computing*, pages 187–198, 1992.
- [17] 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*, 1995.
- [18] T.Abdelzaher, A.Shaikh, S.Johnson, F.Jahanian, and K.G.Shin. Rtcas: Lightweight multicast for real-time process groups. In *IEEE Real-Time Technology and Applications Symposium*, 1996.
- [19] Hengming Zou and Farnam Jahanian. Real-time primary-backup replications with temporal consistency guarantees. Technical Report CSE-TR-356-98, University of Michigan, February 1998.