

Optimization of a Real-Time Primary-Backup Replication Service

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

The primary-backup replication model is one of the commonly adopted approaches to providing fault tolerant data services. Its extension to the real-time environment, however, imposes the additional constraint of timing predictability, which requires a bounded overhead for managing redundancy. This paper discusses the trade-off between reducing system overhead and increasing (temporal) consistency between the primary and backup, and explores ways to optimize such a system to minimize either the inconsistency or the system overhead while maintaining the temporal consistency guarantees of the system. An implementation built on top of the existing RTPB model [20] was developed within the x-kernel architecture on the Mach OSF platform running MK 7.2. Results of an experimental evaluation of the proposed optimization techniques are discussed.

1 Introduction

A common approach to building fault-tolerant distributed systems is to replicate servers that fail independently. 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. The main attraction of the state-machine approach is the transparency in failure recovery. The client does not see, nor does it need to know, that there are faulty servers. However, the design and implementation of the underlying agreement protocol is often tricky and complicated. Passive replication, on the other hand, is simpler in design and involves less redundant processing. It 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 which updates the backups. If the primary fails, a failover occurs and one of the backups becomes the new primary. The major drawback of the passive approach is that the clients need to participate in the recovery process and the recovery takes longer than that of the active scheme.

While each of the two approaches has its advantages and disadvantages, neither directly solves the problem of accessing data in a real-time environment. Many embedded real-time applications, such as computer-aided manufacturing and process control, require timely execution of tasks, and their own processing needs should not be compromised by fault tolerant access to data repositories. Therefore it is important for replication servers to provide timely access to the service. In such a real-time environment, the scheme employed in most of the conventional replication systems may prove insufficient for the needs of applications. Particularly, when time is scarce and the overhead for managing redundancy is too high, an alternative solution is required to provide both timing predictability and fault tolerance.

This paper builds on our *Real-Time Primary-Backup (RTPB) Replication* [20] model that achieves a balance between fault tolerance and timing dependability. However, the proposed scheduling algorithm in [20] is not optimal in the sense that it minimizes neither the temporal inconsistency between the primary and backup nor the system overhead in maintaining a given bound on the inconsistency. The objective of this paper is to extend the design of the RTPB service to support optimization within the system in the above two aspects. The most important contributions of this paper are the formulation of the optimization

problems and the solutions to them. As a by-product, we prove that the problem of minimizing system overhead in maintaining a given bound on the temporal inconsistency between the primary and backup is intractable but the solution in the case when a temporal bound is imposed for each individual object is polynomial. Experimental data that validates the results are presented.

The next section describes the underlying RTPB model and defines the concept of average temporal distance. Section 3 and 4 discuss the optimization problem from the two perspectives we have mentioned. Section 5 extends the results by considering faults. Section 6 is the implementation followed by performance analysis in Section 7. Section 8 discusses the related work, and finally, Section 9 is our conclusion.

2 The RTPB model

The RTPB scheme developed in [20] is an extension of the traditional primary-backup approach to support real-time computation and temporal consistency guarantees. The main features that set RTPB apart from its traditional counterpart are the decoupling of client request processes from backup updates and bounded temporal inconsistency between the primary and backup. This earlier work introduced the concepts of *phase variance*, *external* and *inter-object temporal consistency*, and the consistency model and scheduling algorithm that guarantees such consistencies.

External temporal consistency deals with the relationship between an object in the external world and its image on the servers. For example, a tracking system must update the positional data of an airplane quickly after it changes position in the air. The inter-object temporal consistency concerns the relationship between different objects or events. For example, in an air traffic control system, there is a time bound between the updates of the positions of two airplanes that are waiting to land at the same airport.

The RTPB system consists of a primary and a backup with the primary being responsible for processing client requests and keeping the backup in a temporal consistent state. We assume that the consistency at the primary is maintained by the timely updates of objects by the clients of the system. The external temporal consistency stipulates that the timestamp of any object i at time t is no more than δ_i^P time units apart from t on primary and δ_i^B on backup; the inter-object temporal consistency requires that the difference be-

tween the timestamps of any two related objects i and j is no more than δ_{ij} . Figure 1 illustrates the model.

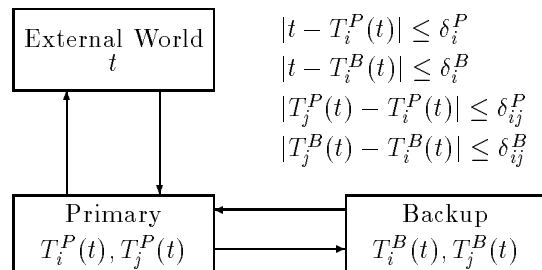


Figure 1: real-time primary-backup replication

The difference between the timestamps of object i 's copies at the primary and backup is a good measurement of the closeness of the two servers with respect to object i , and is conveniently called *temporal distance* of the object in the system. If we know the temporal distance for every object in the system, we can define the overall closeness between the primary and backup to be the arithmetic average of the closeness with respect to each object. We call this arithmetic average *average temporal distance* (ATD) or *temporal inconsistency* between the primary and backup. Let O_i^P and O_i^B denote object i at the primary and backup, respectively, and $T_i^P(t)$ and $T_i^B(t)$ denote the corresponding timestamps of O_i^P and O_i^B at time t :

Definition 1: The temporal distance between the primary and backup with regard to object i at time t is $|T_i^P(t) - T_i^B(t)|$. The average temporal distance between the primary and backup at time t is $\sum_{i=1}^n |T_i^P(t) - T_i^B(t)|/n$, where n is the total number of objects registered in the replication service.

The definitions of temporal distance and average temporal distance also suggest a way to compute their values. In particular, to compute the temporal distance for any single object i at any time t , we need to calculate the absolute differences between the time instants at which object i was last updated on the primary and backup, respectively. To compute ATD in a RTPB system at any given time t , we need to compute the temporal distance for each object and then take their arithmetic average. In the following sections, we discuss ways to minimize the ATD and the system overhead in maintaining a given bound on the ATD.

3 Minimizing the ATD

Observe that the more frequently updates are sent to backup, the smaller the ATD will be. Thus, if the period of the updates sent to backup for each object is minimized, the ATD between the primary and backup is also minimized. Furthermore, the minimization of the update period at the backup for each object results in the minimization of the sum of the update periods for all objects, and vice versa. Hence, we can use the sum of the update periods for all objects as our objective function. Let:

p_i^P denote the period of the task that updates O_i^P .
 e_i^P denote the execution time of the task updating O_i^P .
 p_i^U denote the period of the task that updates O_i^B .
 e_i^U denote the execution time of the task updating O_i^B .
 δ_i^P, δ_i^B denote temporal constraint for O_i^P, O_i^B .

Then our optimization problem can be stated as:
minimize objective function: $\sum p_i^U$ **under constraint:** $\sum_{i=1}^n (e_i^U/p_i^U + e_i^P/p_i^P) \leq 2n(2^{1/2n} - 1)$ and $p_i^P \leq p_i^U \leq \delta_i^B, i = 1, 2, \dots, n$

The constraint $\sum_{i=1}^n (e_i^U/p_i^U + e_i^P/p_i^P) \leq 2n(2^{1/2n} - 1)$ is needed to guarantee task schedulability under both the Rate-Monotonic [14] and Distance-Constrained [4] scheduling algorithms.¹ The inequality $p_i^U \leq \delta_i^B$ ensures that the temporal constraint imposed on object i at the backup is maintained while inequality $p_i^P \leq p_i^U$ guarantees that no unnecessary update is sent to the backup since more frequent updates at backup would not make any difference when there is no message loss between servers.

However, such formulation of the optimization problem is not very useful in practice because it is not in a normalized form and the solutions to it are skewed towards the boundaries. In other words, for most objects, the periods are either very high or very low. For example, if we have four objects in the system with $e_i^P = e_i^U = \{1, 2, 3, 4\}, p_i^P = \{10, 20, 30, 40\}, \delta_i^B = \{50, 50, 60, 70\}$, respectively, then the minimization of $\sum p_i^U$ is achieved if $p_i^U = \{45, 20, 30, 40\}$. All values except one in the p_i^U vector are on the lower limits (skewed solution) in the constraining inequalities.

In fact, we can show that in any optimal solution (if one exists) for the above optimization problem, there is only exactly one object that has its period strictly bounded between the boundaries given in the constraining inequalities. Indeed, we can generalize this observation to the following theorem:

¹An exposition of scheduling algorithms is beyond the scope of this paper, refer to the references for detail.

Theorem 1: The optimal solution in maximizing $\sum_{i=1}^n c_i x_i$ subject to $\sum_{i=1}^n a_i x_i \leq b, l_i \leq x_i \leq u_i$ for $i = 1, \dots, n$ has at most one of the variables bounded strictly between its lower and upper boundaries.²
 \square

Since our optimization problem can be converted to the form stated in Theorem 1, any solution to it is skewed toward boundaries and thus not acceptable in practice. This analysis necessitates a need to find a normalized objective function whose solutions are not skewed toward any extreme.

One suitable alternative is the **maximization of the minimum** $p_i^P/p_i^U, i = 1, 2, \dots, n$. This objective function is simple and its solution can be easily found by a polynomial time algorithm. Furthermore, all the periods are made as small as possible together without skew. Now the optimization problem can be formulated as: **maximize** objective function $\min(p_i^P/p_i^U)$ **under constraint** $\sum_{i=1}^n (e_i^U/p_i^U + e_i^P/p_i^P) \leq 2n(2^{1/2n} - 1)$ and $p_i^P \leq p_i^U \leq \delta_i^B, i = 1, 2, \dots, n$

The proof that the above function indeed minimizes the ATD is beyond the scope of this paper. However, its correctness can be explained intuitively as follows: since the maximization of the minimum of (p_i^P/p_i^U) minimizes the update transmission period p_i^U proportionally to its corresponding update period at the primary, the overall update frequency at the backup is therefore maximized proportionally, which results in the minimization of the ATD.

The above formulation states that to minimize the ATD, one needs to minimize the maximum period of the update tasks for the object set subject to both schedulability and temporal constraint tests. A polynomial time algorithm that achieves this objective is discussed below.

We distinguish between *Static* and *dynamic* allocation. Static allocation deals with the case where the set of objects to be registered are known ahead of time. The goal is to assign the update scheduling period for each object such that the value of the object function is maximized. Dynamic allocation deals with the case where a new object is waiting to be registered in a running system in which a set of objects have already been registered.

Static allocation: Since all objects are known a priori, the algorithm for scheduling updates from primary to

²Due to space limitation, we omit the proof here. Refer to the extended version [19] for all the proofs of the theorems

backup is relatively straightforward:

1. Assign $p_i^U = \delta_i^B$ initially
2. Check schedulability test using rate-monotonic [14] or distance-constrained [4] scheduling. If the test fails, then it is impossible to schedule the task set without violating the given temporal constraint.
3. Otherwise, sort the terms $p_i^P/p_i^U, i = 1, \dots, n$ into ascending order.
4. Shorten the update period (p_i^U) of the first term in the ordered list until either it becomes the second smallest term, or the update period reaches the lower bound (p_i^P), or the utilization rate of the whole task set is saturated.
5. Repeat steps 3-4 until the utilization becomes saturated.

Note that steps 3-4 can be implemented efficiently by using binary insertion which is bounded by $\log n$, and the running time of the algorithm is dominated by the number of iterations of steps 3-4 which is bounded by $(\max(p_i^P/p_i^U) - \min(p_i^P/p_i^U))^2$, the total running time of our algorithm is $O((\max(p_i^P/p_i^U) - \min(p_i^P/p_i^U))^2 * \log n)$.

Dynamic allocation: A new object is being registered with a system in which other objects have already been checked in. We want to find out if the new object can be admitted and at what frequency we can optimally schedule its update task. We can pursue either a local optimum or global optimum with the difference being that local optimum algorithm does not adjust the periods of tasks that are already admitted and global optimum algorithm considers all tasks in devising an optimal scheduling.

Local optimal

1. Assign the smallest value that is $\geq p_i^P$ but $\leq \delta_i^B$ to the new task such that the total utilization of the task set is still under $2n(2^{1/2n} - 1)$ (This ensures that the whole task set is schedulable under rate-monotonic [14] or distance-constrained scheduling [4]).
2. If Step 1 fails, then reject the object.

Global optimal

1. Insert term p_i^P/p_i^U into the ordered list that we obtained in static or previous allocations.
2. Rerun the static allocation algorithm.
3. If Step 2 fails, then reject the new object.

The running time for local optimum algorithm is linear (with respect to the number of objects); The time complexity for the global optimum algorithm is the same as that of the static allocation algorithm which is polynomial.

4 Minimizing system overhead

This section discusses the problem of minimizing system overhead in maintaining a given bound on the ATD. Since system overhead is mainly caused by the scheduling of updates from the primary to backup, the minimization of this overhead means the minimization of the number of updates sent to the backup, which is equivalent to the maximization of the periods of the update tasks under the given ATD. We assume that the transmission of an update message of any object takes the same amount of primary resource.

Since the temporal distance is determined by the frequency of the updates that are sent to the backup, the bound on the ATD can be converted to a bound on the periods of the updating tasks. Moreover, if we normalize the period of each update task with the denominator $\delta_i^P - p_i^P$, the given constraint on ATD can be transformed to the inequality $\sum p_i^U / (\delta_i^P - p_i^P) \leq B$, where B is the bound derived from the given constraint on the ATD. Observe that we do not need to know the actual value of B . The important thing here is that B can be derived from the bound on ATD.

Thus, our optimization problem can be formulated as maximization of function $\sum p_i^U$ under constraint $\sum p_i^U / (\delta_i^P - p_i^P) \leq B$, where B is some positive constant. Unfortunately, this problem turns out to be intractable.

Theorem 2: The problem of the minimizing the number of updates from primary to backup in maintaining a given bound on the ATD is NP-complete. \square

However, the problem becomes polynomial if we impose a temporal constraint on every individual object (instead of global temporal constraint on the ATD for the whole object set). This polynomial solution can be used as an approximation to the solution of the NP-complete version of the problem. Suppose we have the additional constraint $p_i^U \leq \delta_i^B, i = 1, 2, \dots, n$, then the following algorithm achieves the minimization of overhead in maintaining the temporal constraints for the corresponding objects in the system:

1. For each object, assign period: $p_i^U = \delta_i^B$.
2. Perform schedulability test. If the test fails, then we cannot guarantee the temporal consistency of the object set. Reject the object set.
3. Otherwise, accept the object.

The rationale behind the algorithm is that to minimize the overhead in achieving the given temporal

constraint, we simply schedule the least number of updates to the backup as possible for each object subject to the individual temporal constraint which is achieved by assigning the update period from the primary to backup to the individual temporal constraint. The running time complexity of the above algorithm is linear.

5 Optimization under faults

The discussion in the previous two sections does not take into consideration message loss. Update messages from the primary to backup can be lost due to a send omission, receive omission, or link failure. In this section, we devise a new scheduling protocol that takes message loss into consideration. We define the probability of a message loss from the primary to the backup to be ρ , and the probability of the temporal consistency guarantee we want to achieve to be P . Then for an update to reach backup with probability P , the number of transmissions needed is $\log(1 - P)/\log(\rho)$. Hence, the problem of minimizing the ATD between primary and backup can be formulated as: **maximize** objective function $\min((p_i^P \log(\rho))/(p_i^U \log(1 - P)))$ **under constraint** $\sum_{i=1}^n (e_i^U/p_i^U + e_i^P/p_i^P) \leq 2n(2^{1/2n} - 1)$ and $(p_i^P \log(\rho))/\log(1 - P) \leq p_i^U \leq \delta_i^B$

The goal is to send updates to the backup more frequently in case of a message loss. The frequency of update transmission can be increased up to the frequency at which any update will reach the backup with the probability P . The maximization of the minimum term in the objective function results in the overall minimization of periods of the update tasks, which consequently results in the minimization of the ATD with probability P . The constraints in the formulation ensure schedulability while maintaining temporal consistency of each individual object and avoiding unnecessary updates to the backup.

With proper substitution of terms, the same algorithm introduced in section 3 can be applied here. Due to space limitation, we omit it. Similarly, the minimization of system overhead in maintaining a given temporal bound on the ATD when message loss occurs becomes the **maximization** of $\sum_{i=1}^n P_i^U$ **under the constraint** of $\sum p_i^U \log(1 - P)/[\log \rho(\delta_i^P - p_i^P)] \leq B$. Again, the problem is NP-complete but an alternate formulation of the problem, where a temporal bound on each individual object is imposed, can be solved in polynomial time. The same algorithm described in Section 4 can be used here if we substitute the upper bound δ_i^B in the constraining inequality by

$$\delta_i^B \log(\rho)/\log(1 - P).$$

6 Implementation

We have integrated the optimization techniques developed in this paper into the RTPB prototype that was built in our previous work [20]. The new prototype is implemented as a user-level x -kernel [7] based server on the MK 7.2 microkernel from the Open Group.³ 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 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.

Figure 2 shows the RTPB system architecture within the x -kernel protocol stack. At the top level is the RTPB API 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. The RTPB protocol sits right below the API layer and 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 MK kernel in this 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. The primary host interacts with the backup host through the underlying RTPB protocol. There are two identical versions of the client application residing on the primary and backup hosts. 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 a consistent system state.

6.1 Admission control

Before a client starts to send periodic updates of a data object to the primary, it first registers the object with the RTPB service so that the primary can

³formerly known as the Open Software Foundation (OSF).

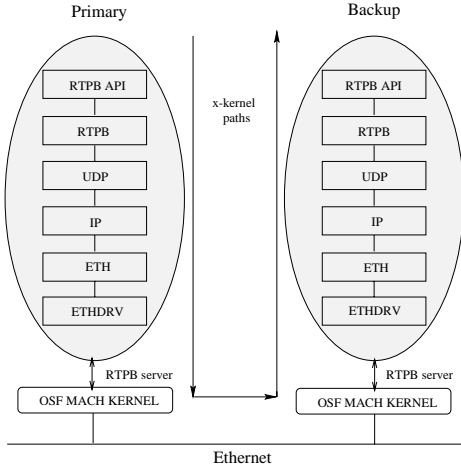


Figure 2: system architecture and protocol stack

perform admission control. During registration, the client reserves the necessary resource for the object on the primary and backup servers. In addition, the client specifies the update period p_i and the temporal constraints for the object on both the primary δ_i^P and backup δ_i^B . 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.

The primary server performs the temporal constraint test by comparing 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. If the condition does not hold, the primary will not admit the object. Each inter-object temporal constraint δ_{ij} for object i, j can be met at the primary if $p_i \leq \delta_{ij}$ and $p_j \leq \delta_{ij}$. The temporal constraint for object i on the backup is also checked to ensure that it can be met [20].

After testing that the temporal constraints hold for object i , the primary needs to 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 by performing a schedulability test based on the rate-monotonic or distance-constrained scheduling algorithm. If the test is successful, the object is admitted into the system.

6.2 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. If ATD minimization is desired, then the algorithm described in section 3 is used. If the minimization of system overhead is sought, then the algorithm discussed in section 4 is used. But in either of these cases, if a client modifies an object i , the primary must send an update for the object to backup within the next $\delta_i - \ell$ time units, where $\delta_i = \delta_i^B - \delta_i^P$ is the window of allowed inconsistency between the primary and the backup; otherwise the object on backup may fall out of the consistency window. For inter-object temporal constraint, the primary need not send updates to the backup within the next $\delta_i - \ell$ time units after the primary is updated. But rather, it schedules the two updates for object i and j within δ_{ij} time units. If both the probability of message loss from the primary to backup and the probability to achieve guarantees of temporal consistency in the system are given, then we apply the optimization technique that deals with faults described in section 5.

6.3 Failure detection and recovery

Failure detection and recovery is a key component of the replication service. Our approach requires that all replication servers exchange periodic *ping* messages which serve as the heartbeats among those servers. Each server acknowledges the *ping* message from the other one. If a server receives no acknowledgment for repeated *ping* messages, it 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 invokes a backup version of the client application at the local machine, feeds the new client with information stored in its memory via an upcall, 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 application task. Our implementation supports the integration of a new backup after a failure is detected.

7 Performance Evaluation

This section evaluates the proposed optimization techniques against the RTPB service developed in [20]. we consider two metrics: average temporal distance and *average duration of backup inconsistency*, which are influenced by several parameters including client

write rate, number of objects being accepted, window size, and message loss probability.

7.1 The ATD metrics

To demonstrate the optimization technique proposed in this paper, we measured the ATD under two conditions with and without optimization. Figure 3(a) and (b) compare the results of optimization to that of without optimization assuming no message loss. As

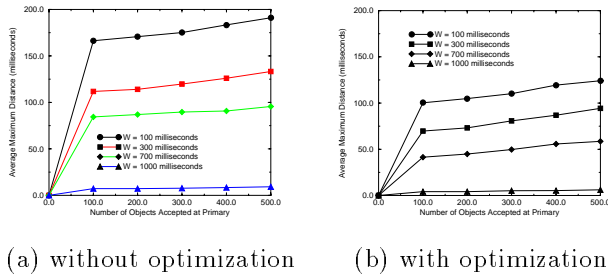


Figure 3: Average primary-backup temporal distance

shown by the two graphs, the ATD with optimization in effect is about 40% smaller than that without optimization under the same set of parameters. The optimized RTPB attempts to send as many updates as possible to the backup. It must be noted that in both graphs, larger window size results in smaller ATD, which conforms to the result presented in [20].

Figure 4 does the same comparison but with consideration of message losses. The two graphs show an

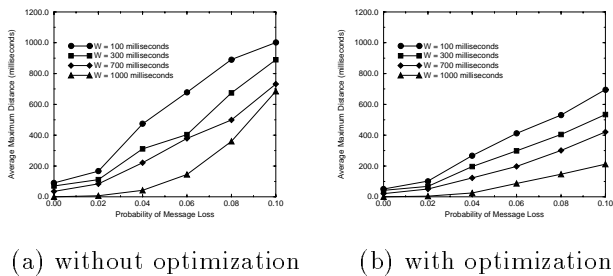


Figure 4: ATD under fault assumption

approximate 35% improvement on the ATD due to the application of the optimization techniques described in this paper. Furthermore, we note that the shape of the graph under optimization is much smoother than

that when no optimization is used, because the effect of message loss is compensated by more frequent scheduling of update messages.

7.2 Duration of backup inconsistency

Since the optimized RTPB minimizes the ATD between the primary and backup, it is expected that the duration of backup inconsistency should also be reduced under the new modified RTPB model. Figure 5(a) and (b) show the duration of backup inconsistency as a function of the probability of message loss between the primary and backup. The figures show

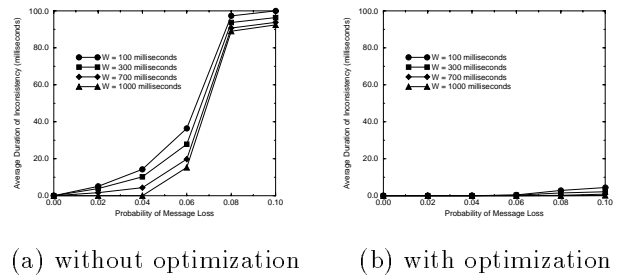


Figure 5: duration of backup inconsistency

that the optimized RTPB has a higher degree of tolerance to message loss than that of normal scheduling. With optimization, there is no backup inconsistency until the message loss rate exceeds 6%, and after that the duration of inconsistency increases slowly. But the one without optimization suffers backup inconsistency when the message loss rate exceeds approximately 1%, and the duration of inconsistency increases rapidly as message loss rate increases. In both cases, for the same message loss rate, the larger the window size, the shorter is the period in which the backup stays in an inconsistent state. 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.

Before we leave this section, it should be mentioned that we have studied the client response time with and without optimization and found out that their performance are not much different. The original RTPB model already schedules a minimum number of updates to the backup (hence leaves maximum available resources at the primary for client request processing). See Figure 6 and 7 in [20] for graphs of this metrics.

8 Related work

8.1 Replication models

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 [6] 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. The TTP provides fault-tolerance by implementing active redundancy through a collection of replicated components with each relies on a number of hardware and software mechanisms for error detection to ensure a fail-silent behavior.

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 communicating 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 [2,3] presents 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.

8.2 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 [5,8,9,11–13,16,17].

For example, a recent work [10] [11] 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 [15] 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.

9 Conclusion

This paper presents the optimization of a real-time primary-backup replication service from two perspectives. By applying the appropriate optimization technique, we can minimize either the average temporal distance or the system overhead in maintaining temporal consistency between the primary and backup. Experimental results indicate that the techniques developed in this work can indeed improve system performance over the original RTPB model.

Avenues for future studies include extension of the concepts of this paper to active replication and a probabilistic analysis of the system.

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] R. Rajkumar et al. 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.
- [3] 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.
- [4] C-C Han and K-J Lin. Scheduling distance-constrained real-time tasks. In *Proc. RTSS'92*.
- [5] 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.
- [6] H.Kopetz and G. Grunsteidl. Ttp - a protocol for fault-tolerant real-time systems. In *IEEE Computer*, volume 27, pages 14–23, January 1994.
- [7] 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.
- [8] 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.
- [9] T-W Kuo and A.K.Mok. Ssp: A semantics-based protocol for real-time data access. In *Proc. RTSS'93*.
- [10] T-W Kuo and A.K.Mok. Ssp: A semantics-based protocol for real-time data access. In *Proceedings of IEEE 14th Real-Time Systems Symposium*, December 1993.
- [11] 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.
- [12] K-J Lin. Consistency issues in real-time database systems. In *Proc. ICSS'89*, pages 654–661.
- [13] K-J Lin and F. Jahanian. Issues and applications. In Sang Son, editor, *Real-time Database Systems*. Kluwer Academic Publishers, 1997.
- [14] 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.
- [15] J.W.S. Liu, W.-K. Shih, and K.-J. Lin. Imprecise computation. In *Proceedings of IEEE*, volume 82, pages 83–94, January 1994.
- [16] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *Proc. of ACM SIGMOD*, pages 377–386, May 1991.
- [17] 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.
- [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] H. Zou and F. Jahanian. Optimization of a real-time primary-backup replication service. Technical Report CSE-TR-367-98, University of Michigan, July 1998.
- [20] H. Zou and F. Jahanian. Real-time primary-backup replications with temporal consistency guarantees. In *Proc. ICDCS*, pages 48–56, May 1998.