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. The extension of the primary-backup protocol to the real-time environment, however, imposes the additional constraint of timing predictability, which requires a bounded overhead for managing redundancy. There is a trade-off between reducing system overhead and increasing (temporal) consistency between the primary and backup, and the way to achieve balance between them is not so obvious. In this paper, we try to explore ways to optimize scheduling update messages from primary to backup while maintaining the temporal consistency guarantees of the system.

Depending on the purpose of the individual replication service and the environment in which it is running, very different objectives can be sought in the process of optimization. This paper considers the optimization problem from two perspectives with one aimed to minimize the average temporal distance between the primary and backup, and the other aimed to minimize the resource being used in maintaining a given temporal constraint on the system. Corresponding optimization techniques have been developed for these two diverse objectives and an implementation of such techniques is also presented. The implementation is built on top of the existing RTPB model developed in [20] which in turn was developed within the *x*-kernel architecture on the Mach OSF platform running MK (mach kernel) 7.2. Results of an experimental evaluation of the proposed optimization techniques are 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 that service 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. 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. Thus it is not involved in the fault recovery process. However, the design and implementation of the consensus 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 backup. The servers run the failure detection protocol, where the regular heart-beat messages are exchanged among the primary and the backups. When the primary fails the backups execute a failover protocol as well as a recovery protocol, in which one of them becomes the new primary and the system is brought back on-line. The clients need to participate in the recovery process.

While each of the above 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.

With the introduction of the concept of external temporal consistency, inter-object temporal consistency, and phase variance, we [20] have developed a real-time primary-backup (RTPB) replication service that achieves the balance between fault tolerance and timing dependability. However, the system built is not optimal in the sense that it neither minimizes the controlled inconsistency between the primary and backup nor minimizes the resource being used in maintaining a given bound on the inconsistency within the system. Our interest in this paper is to extend the design of the real-time primary-backup replication system to support optimization within the system in the above two aspects. We achieve this by exploring ways to optimize the scheduling of update messages from the primary to backup while maintaining the temporal consistency guarantees of the system.

Although other avenues could be pursued in optimizing the RTPB system, the aforementioned two objectives are the most appropriate target for our optimization effort. After all, a tightly temporal consistent system that consumes the minimum resource is what we are hoping for in building a primary-backup replication service. But due to their nature, these two objectives cannot be met simultaneously. On one hand, we can try to minimize the inconsistency of objects between the primary and backup to the range that is achievable under the maximum utilization of computing power of the machines, which leaves no room for other tasks that may later arrive in the system. On the other hand, we could minimize the resources that are used in maintaining a given bound on the inconsistency between the primary and backup, which consequently leaves maximum available computing power for other applications that may happen in the system at a later time.

The choice of optimization criteria depends on the goal of the system and the environment in which it is running. If our main interest is to maintain a tightly consistent primary-backup replication, the first option may be the best choice. However, if fast response to outside clients is our primary concern, then the second option would be a proper alternative. No matter which choice we make, the optimizations are in addition to our guarantees of temporal constraints imposed on the system and our interest is to find out at what frequency the update for each object should be sent from primary to backup such that our goal is achieved.

This paper tackles the optimization problem in real-time primary-backup replication from the two perspectives described above. This work builds on the *Real-Time Primary-Backup (RTPB)*

Replication with Temporal Consistency Guarantees by Zou and Jahanian [20] but distinguishes itself from that work and focuses on the area of optimization within the system. The most important contributions of this paper are the formulation of the optimization problems and the solutions to them. As a by-product, we proved that the problem of minimizing resources being used in maintaining a given bound on the average temporal distance 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 were collected at the end.

The reminder of the paper is organized as follows: Section 2 describe the underlying RTPB model and defines the concept of average temporal distance. Section 3 discusses the optimization problem of minimizing the average temporal distance between the primary and backup. Section 4 presents the optimization problem of minimizing the resources being used in maintaining a given average temporal distance bound between the primary and backup. Section 5 extends the results of Section 3 and 4 to include the consideration of 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 work by Zou and Jahanian in [20] was an attempt to construct a fault tolerant primary-backup replication service that supports real-time computation and temporal consistency guarantees. The primary-backup replication is a widely-used approach for providing fault tolerance by replicating servers that fail independently. The main characteristic of this approach is that it distinguishes one replica in the system as the primary server, which handles all client requests and the rest of the replicas in the system as the backups. 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.

The overhead incurred in maintaining redundant components in a replication service can be reduced by exploring weak consistency semantics of applications. One category of consistency semantics that is particularly relevant to the primary-backup model 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.

The rationale behind the concept of both external and inter-object temporal consistency stems from the requirement of practical applications: A battlefield tracking system needs to keep the positional data of an airplane very close (in temporal sense) to the actual position of the plane; There is a time bound between accelerating a plane and the lifting of it into air during takeoff because the runway is of limited length and the airplane cannot keep accelerating on the runway indefinitely without lifting off, etc. In our replication system, these two types of temporal consistency must be ensured at all times if the replication service is to function properly in a real-time fashion.

The real-time primary-backup (RTPB) model developed in [20] consists of a primary and a backup with the primary being responsible for processing client requests and keeping the backup up to date in system state. The consistency at the primary is maintained by the timely updates

of objects by the clients of the system. The system must ensure that both the external temporal consistency and the inter-object temporal consistency are maintained at both the primary and backup. Figure 1 illustrates the model.

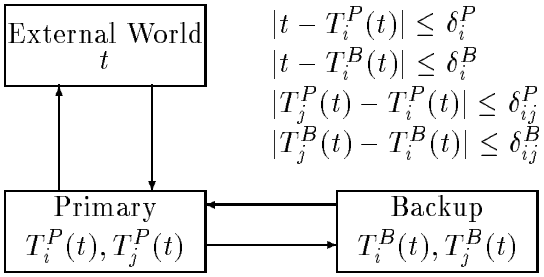


Figure 1: real-time primary-backup replication

In the model shown in Figure 1, the copy of an object's value on the primary may not be the same as the copy on the backup. In case of a primary crash, a failover procedure is triggered and one of the backups becomes the new primary. In order for the recovery process to be smooth, the state of the backup must be close to that of the primary. In other words, the copy of any object on the backup must be close to that on the primary. Hence, it is important to measure the degree of closeness between the object copy on the primary and that on the backup. If we timestamp each data copy on both the primary and backup, then at any given time instant t, the difference between the timestamp of object i at the primary and the timestamp of the same object at the backup is a good measurement of closeness of the primary and backup with regard to object i. We called this closeness *temporal distance* of object i in the system.

Furthermore, if we know the temporal distance for every object in the system, we could define the overall closeness between the primary and backup to be the arithmetic average of the closeness with regard to each object. We call this arithmetic average *average temporal distance* between the primary and backup. Let:

$T_i^P(t)$ denote the timestamp of the copy of object i at primary at time instant t.

$T_i^B(t)$ denote the timestamp of the copy of object i at backup at time instant t.

Here time is measured as multiples of some clock (microsecond or millisecond depending on the individual case).

Then, we have:

Definition 1: The temporal distance between the primary and backup with regard to object i at time instant t is defined to be $|T_i^P(t) - T_i^B(t)|$. The average temporal distance between the primary and backup at time instant t is defined to be $\sum_{i=1}^{n} |T_i^P(t) - T_i^B(t)|/n$, 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 instant t, we need to find out the time instants at which object i was last updated on the primary

and backup, respectively, and take the absolute difference of these two timestamps. To compute the average temporal distance in a RTPB system at any given time instant t, we need to compute the temporal distance for each object and then take their arithmetic average.

Clearly, the smaller the average temporal distance, the better the system will function in case of a failover. In the following sections, we will discuss ways to minimize this average temporal distance or to minimize the resources being used in maintaining a given bound on the average temporal distance.

3 Minimization of average temporal distance

It is straightforward to see that the more frequently updates are sent to backup, the shorter the average temporal distance between the primary and the backup will be. Thus, if the period of the updates sent to the backup for each object is minimized, the average temporal distance 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 be the period of the task that updates object i at the primary.

e_i^P be the execution time needed to update object i once at the primary.

p_i^B be the period of the task that updates object i at the backup.

e_i^B be the execution time needed to update object i once at the backup.

p_i^U be the period of the task that transmits the update of object i from primary to backup.

e_i^U be the execution time needed to transmit object i onto the network.

δ_i^P be the external temporal constraint for object i at primary.

δ_i^B be the external temporal constraint for object i at backup.

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

$$p_i^P \leq p_i^U \leq \delta_i^B, i = 1, 2, \ldots, n$$

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 [13] and Distance-Constrained [3] 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

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

backup since more frequent updates at backup would not make any difference when there is no message loss (means that all messages reach their destinations within the bound of some maximum communication delay).

However, the above 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, i.e. 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 $p_i^U = \{45, 20, 30, 40\}$ respectively. All values except one in the p_i^U vector are on the lower limits (skewed solution) in the constrainting 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:

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

Proof: Without loss of generality, assume:

$$c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$$

and

$$a_i > 0, i = 1, 2, \ldots, n$$

Suppose, in the optimal solution, there exists some k such that:

$$l_k < x_k = \alpha < u_k$$

Then, we claim that the following inequalities stand:

$$\forall i,\ 1 \leq i \leq (k-1) : x_i = u_i$$
$$\forall i,\ (k+1) \leq i \leq n : x_i = l_i$$

We prove this theorem by contradiction. Suppose for some value of m, $m < k$, $x_m = \beta < u_m$. Then we can get a better solution by making $x_m = \beta + \epsilon$, and reducing x_k to $\alpha - a_m * \epsilon / a_k$ and keeping other variables unchanged.

Old solution (supposedly optimal): $x_m = \beta < u_m$, $x_k = \alpha$

New solution: $x_m = \beta + \epsilon$, $x_k = \alpha - a_m * \epsilon / a_k$

Objective function value is increased by:

$$c_m \epsilon - c_k a_m \epsilon / a_k$$

which is

$$a_m \epsilon (c_m/a_m - c_k/a_k)$$

Since $c_m/a_m \geq c_k/a_k$, $a_m > 0$, and $\epsilon > 0$, the above quantity is positive. Furthermore, we can choose ϵ to be small enough such that $\beta + \epsilon < u_m$ and $\alpha - a_m * \epsilon/a_k > l_k$. Thus, the new solution is better than the old one which was supposedly optimal. This is a contradiction.

All other cases can be proved similarly. Hence, the optimal solution in maximizing $\sum_{i=1}^{n} c_i.x_i$ subject to the given constraints can have at most one of its variables bounded strictly between its lower and upper boundaries. □

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, $i = 1, 2, \ldots, n$. This objective function is simple and its solution can be easily found by a linear program. Furthermore, all the periods are made as small as possible together without skew. 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

$$p_i^P \leq p_i^U \leq \delta_i^B, i = 1, 2, \ldots, n$$

The proof that the above formulation indeed minimizes the average maximum temporal distance between the primary and backup is very complex and beyond the scope of this paper. However, it correctness can be explained intuitively as follows: since the minimization of the term 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 average maximum temporal distance.

The above formulation states that to minimize the average temporal distance between the primary and backup, 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 in the subsequent subsection.

3.1 A polynomial time algorithm

Depending on the environment in which the system is running, there are two cases to be considered: static allocation and dynamic scheduling. In the case of static allocation, the set of objects are known ahead of time; we are to assign the update scheduling period for each object such that the vaule of the object function is maximized. The dynamic scheduling case deals with the registration of a new object in a running system in which a set of objects have already been

registered. We want to find out if the new object can be admitted and at what frequency its update task can be optimally scheduled.

Static allocation: Suppose we are given a set of objects. In this case, since all objects are given before hand, the algorithm for scheduling updates from primary to backup is relatively straightforward:

1. Assign $p_i^U = \delta_i^B$ initially.
2. Check schedulability test using rate-monotonic [13] or distance-constrained [3] scheduling. If the test fails, then it is impossible to schedule the task set without violating the given temporal constraint so we stop here. Otherwise go to next step.
3. Sort term $p_i^P/p_i^U, i = 1, 2, \ldots, n$ 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. (This process can be implemented efficiently by using the method of binary insertion)
5. If the utilization becomes saturated in previous step, then stop here since we have obtained the optimal scheduling already. Otherwise, repeat steps 3 and 4.

The above algorithm directly maximizes the minimum of $p_i^P/p_i^U, i = 1, 2, \ldots, n$. The running time of the algorithm is dominated by the number of iterations of steps 3 and 4 which is bounded by $(max(p_i^P/p_i^U) - min(p_i^P/p_i^U))^2$. Since the time complexity of step 3 and 4 take are bounded by $\log n$ (binary insertion), the total running time of our algorithm is therefore $O((max(p_i^P/p_i^U) - min(p_i^P/p_i^U))^2 * \log n)$ which is polynomial.

Dynamic scheduling: In this case, a new object is being registered with a system that has some objects already 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. The difference between local and global optimum is that in the former case, we do not adjust the periods of tasks that are already admitted and in the later case all tasks need to be considered in devising an optimal scheduling.

Local optimal

1. Assign the smallest value that is greater than or equal to p_i^P and less than or equal to δ_i^B to the new task such that the total utilization of the task set is still under $2n(2^{1/2n} - 1)$. The last condition ensures that the whole task set is schedulable under rate-monotonic [13] or distance-constrained scheduling [3] algorithm.
2. If the above allocation is not feasible, 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 the rerun is successful, then accept the object. Otherwise reject it.

The running time for local optimum algorithm is linear; The time complexity for the global optimum algorithm is the same as that of the static allocation algorithm which is polynomial.

4 Minimization of resources being used

In this section, we discuss the problem of minimizing the CPU resource used in sending updates to maintain a given bound on the average temporal distance between the primary and backup.

Since the usage of CPU resource at the primary is proportional to the frequency at which updates are sent to the backup, the minimization of the CPU resource used in maintaining a given temporal constraint on the average temporal distance requires the minimization of the number of updates that are sent to the backup (assume the transmission of an update message of any object takes the same amount of primary resource), which is equivalent to the maximization of the periods of the update tasks under the given temporal constraint.

Since the temporal distance is determined by the frequency of the updates that are sent to the backup, the bound on the average temporal distance 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 average temporal distance 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 average temporal distance. 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 average temporal distance.

Therefore, our optimization problem can be formulated as maximization of the objective 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. To prove the NP-completeness of the problem, let's rephrase the optimization problem in a more convenient way:

Given δ_i^P and p_i^P, positive integers B, $K \geq \sum_{i=1}^n p_i^P$, is there a choice of p_i^U such that $\sum_{i=1}^n p_i^U/(\delta_i^P - p_i^P) \leq B$ and such that $\sum_{i=1}^n p_i^U \geq K$.

Then, we have the following theorem:

Theorem 2: The problem of the minimization of resources used in maintaining a given bound on the average temporal distance between the primary and backup is NP-complete.

Proof: First, we show that the problem is in the NP set. The problem is in NP because a non-deterministic algorithm can guess a subset of p_i^U and check within polynomial time that the given selection satisfies all the constraints stated in the problem.

Then, we prove it to be NP-complete by transforming the integer knapsack problem to our optimization problem. Let a finite set U, for each $i \in U$, a size $s_i \in Z^+$ and a value $v_i \in Z^+$, and positive integers B and K be an instance of our integer knapsack problem, we construct our instance of the optimization problem as follows:

1. let $p_i^P = K/(v_i n), i \in U$
2. let $\delta_i^P = p_i^P + 1/s_i, i \in U$
3. let $B = B$

The above transformation procedure takes only polynomial time to complete. The only thing left for us to do is to show that the integer knapsack problem is solvable if and only if the optimization problem is solvable.

$\Longrightarrow$ if the knapsack problem is solvable, then our optimization problem is also solvable.

Suppose $c_i, i = 1, 2, \ldots, n$ is a solution to our knapsack problem, i.e. $c_i, i = 1, 2, \ldots, n$ are non-negative integers such that $\sum c_i.s_i \leq B$ and such that $\sum c_i.v_i \geq K$.

From $\sum c_i s_i \leq B$, we have $\sum c_i/(\delta_i^P - p_i^P) \leq B$.

Since $\sum p_i^P v_i = \sum K/n = K$, and $\sum c_i v_i \geq K$, we have: $\sum c_i v_i \geq \sum p_i^P v_i$. Hence $\sum c_i \geq \sum p_i^P$. Thus, c_i is also a solution to our optimization problem.

$\Longleftarrow$ if optimization problem is solvable, then the integer knapsack problem is also solvable.

Suppose p_i^U is a solution to the above optimization problem. Then from $\sum p_i^U/(\delta_i^P - p_i^P) \leq B$, we have $\sum p_i^U s_i \leq B$. Furthermore, from $\sum p_i^U \geq \sum p_i^P$, we have $\sum p_i^U v_i \geq \sum p_i^P v_i = K$. Hence p_i^U is also a solution to the integer knapsack problem.

Since the integer knapsack problem is NP-complete [15], the optimization problem is NP-complete. $\square$

However, the problem becomes polynomial if we impose a temporal constraint on every individual object (instead an overall temporal constraint on the average temporal distance for the whole object set). And 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 $p_i^U \leq \delta_i^B, i = 1, 2, \ldots, n$, then the following algorithm achieves the minimization of resources used 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. Check 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 resource being used in achieving the given temporal constraint, we simply schedule as less 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 the bakcup to the individual temporal constraint.

The running time complexity of the above algorithm is linear.

5 Optimization under faults

The optimization discussion in the previous two sections did not take into consideration message loss between primary and backup. Since update messages from the primary to backup could be potentially lost due to a send omission, receive omission, or link failure, the methods derived in previous sections in minimizing the average temporal distance between the primary and backup may not be suitable. 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 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 minimization of average temporal distance between the primary and backup can be formulated as:

maximize objective function:

$$min(\frac{p_i^P \log(\rho)}{p_i^U \log(1-P)}) \tag{5.1}$$

under constraint:

$$\sum_{i=1}^{n}(\frac{e_i^U}{p_i^U} + \frac{e_i^P}{p_i^P}) \leq 2n(2^{1/2n} - 1) \tag{5.2}$$

and

$$\frac{p_i^P \log(\rho)}{\log(1-P)} \leq p_i^U \leq \delta_i^B \tag{5.3}$$

The goal is to send updates to the backup more frequently in case of a message loss. This frequency can be increased up to the point that the frequency of update transmissions equals the frequency at which any update will reach the backup with probability P. The maximization of the minimum term in Formula 5.1 results in the overall minimization of periods of the update tasks which consequently results in the minimization of the average temporal distance between the primary and backup with probability P. Inequality 5.2 is needed to guarantee that all tasks at the primary is schedulable under a rate-monotonic and distance-constrained scheduler. Finally, Inequality 5.3 sets the upper and lower limits of the periods of the update tasks with the upper limit guaranteeing temporal consistency of each individual object and the lower limit avoiding unnecessary updates being sent to the backup.

With proper substitution of terms, the same algorithm introduced in section 3 can be applied here. Specifically, the algorithm is as follows:

1. Assign $p_i^U = \delta_i^B$ initially.
2. Check schedulability test using rate-monotonic [13] or distance-constrained [3] scheduling algorithm. If the test fails, then it is impossible to schedule the task set without violating the given temporal constraint so we stop here. Otherwise go to next step.
3. Sort term $p_i^P \log(\rho)/(p_i^U \log(1-P)), i = 1, 2, \ldots, n$ into ascending order.
4. Shorten the update period of the first term in the ordered list until either it becomes the second smallest term, or the update period reaches the lower bound specified in 5.3, or the utilization rate is saturated (This process can be implemented efficiently by using binary insertion).
5. If the utilization becomes saturated in previous step, then the algorithm terminates since we have obtained the optimal scheduling. Otherwise, repeat steps 3 and 4.

Similarly, the minimization of resources used in maintaining a given temporal bound on the average temporal distance 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 the alternative version 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]. This new modified RTPB system is implemented as a user-level x-kernel [6] based server on the MK 7.2 microkernel from the Open Group.[2] The protocol objects communicate with each other through a set of uniform x-kernel 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 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 within the 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.

The primary host interacts with the 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 application programming interface developed for the system.

[2] formerly known as the Open Software Foundation (OSF).

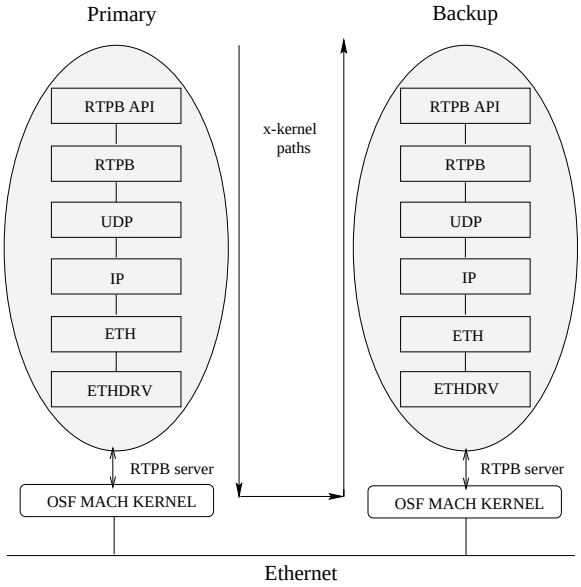


Figure 2: system architecture and protocol stack

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 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 update period p_i as well as the temporal consistency allowed for the object on both the primary site δ_i^P, and the backup site δ_i^B, where the temporal consistency specified by the client is relative to the external world. 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 performs the temporal constraint test according to the rules developed in [20]. Specifically, it 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. 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. Specifically, given objects i, 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. 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 consistency 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. Simply put, 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.

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. 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. If the minimization of the average temporal distance between the primary and backup is desired, then the algorithm described in section 3.1 is adapted. If the minimization of resources being used 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 the 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 the backup may fall out of the consistency window. Because UDP, the underlying transport protocol we use, does not provide reliability of message delivery, we built 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 the 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, the primary schedules the two updates for object i and j within δ_{ij} time units.

If the probability of message loss from the primary to the backup and the probability to achieve guarantees of the 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. The approach requires 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 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 sensing task. Our implementation supports the integration of a new backup after a failure is detected.

7 Performance Evaluation

This section summarizes the results of a detailed performance evaluation of the RTPB replication service introduced in this paper. There are many ways to measure the performance of the system, but the two most important ones are those that show the performance improvement due to the proposed optimization in Section 3. Specifically, we will consider the following metrics:

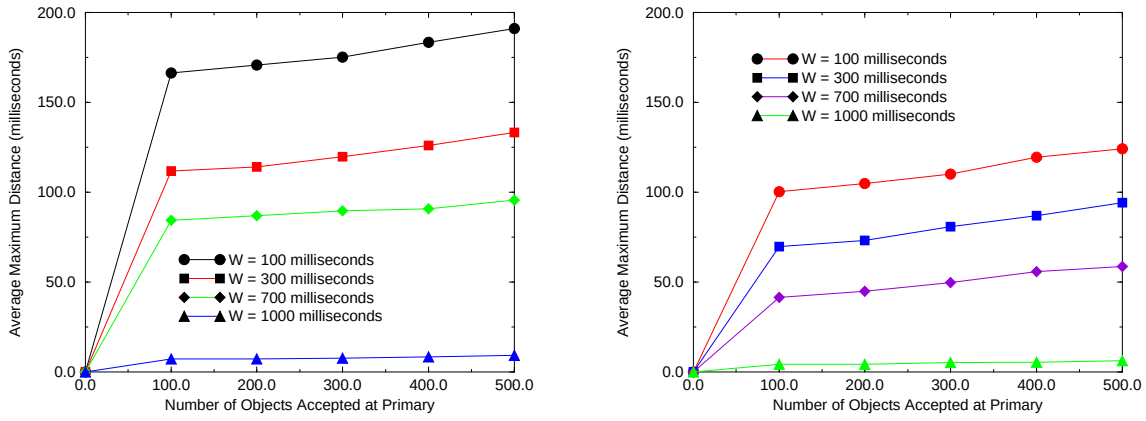
- Average temporal distance between primary and backup
- Average duration of backup inconsistency

These metrics are influenced by several parameters, including client write rate, number of objects being accepted, window size, and communication failure. We wrote an application that registers and updates objects in the system under various conditions such that all cases are covered.

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.

7.1 Average temporal distance between primary and backup

To show that the optimization technique is working, we measured the average temporal distance between the primary and backup 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. Figure 3 (a) shows the average temporal distance between the primary and backup when no optimization is used, and Figure 3 (b) shows the same graph with the optimization technique described in Section 3 being applied.



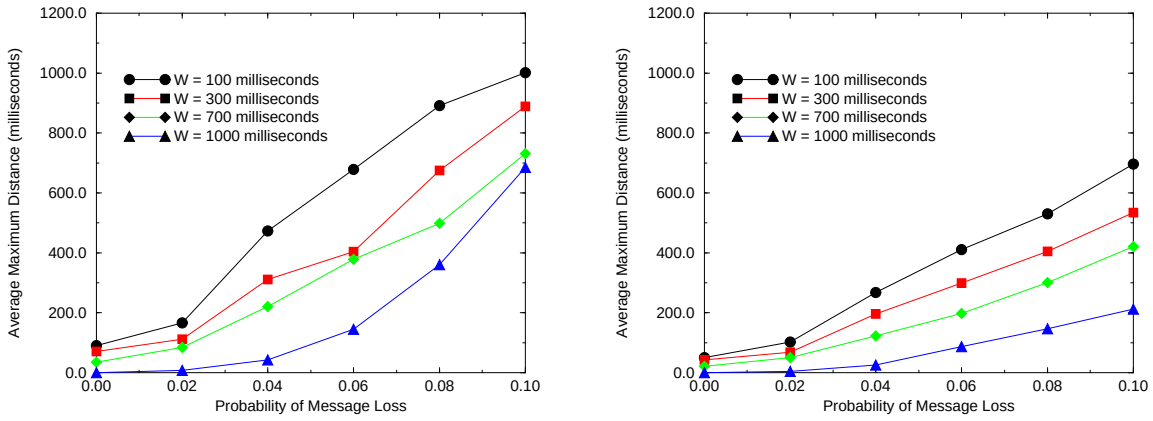
(a) without optimization (b) with optimization

Figure 3: Average temporal distance between primary and backup

As shown by the two graphs, the average temporal distance with optimization in effect is about

40% smaller than that without optimization under the same set of parameters. The optimized RTPB sends as much update as possible to the backup while the one without optimization does not. It must be noted that in both graphs, larger window size results in smaller average temporal distance, which conforms to the result presented in [20].

Figure 4 (a) and (b) compare the results of optimization to that of without optimization when message loss is considered. Figure 4 (a) shows the average temporal distance between the primary and backup as a function of message loss rate when no optimization is used, and Figure 4 (b) shows the same graph but using the optimization technique described in Section 3.



(a) without optimization

(b) with optimization

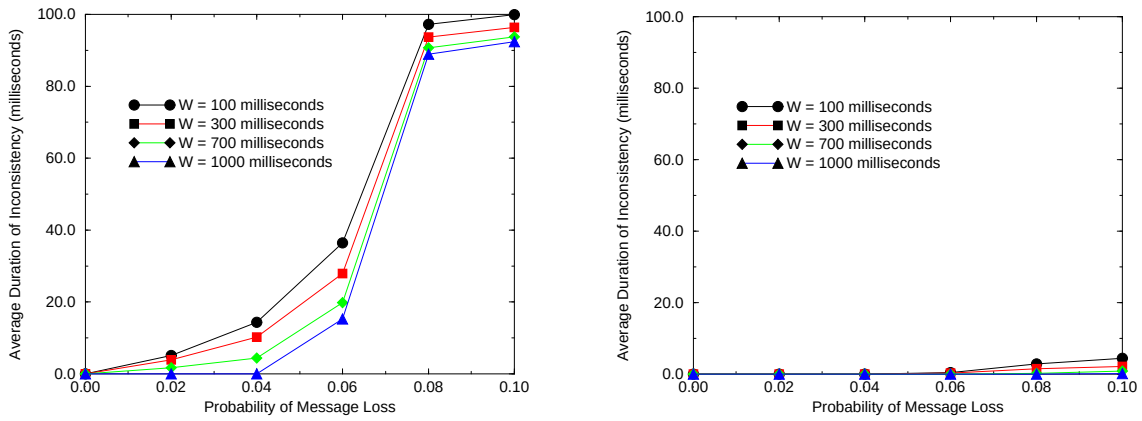
Figure 4: Average temporal distance between primary and backup under fault assumption

The two graphs show an approximate 35% improvement on the average temporal temporal distance between the primary and backup 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 the scheduling method which compensates for message loss.

7.2 Average duration of backup inconsistency

Since the optimized RTPB minimizes the average temporal distance between the primary and backup, it is expected that the duration of backup inconsistency should also be minimized 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 difference between these two graphs is that Figure 5(a) shows the result when no optimization technique is used while Figure 5(b) shows the result in which the proposed optimization in Section 3 is applied.

The figures show that the optimized one has a higher degree of tolerance to message loss than that of normal scheduling. With optimization, there is no backup inconsistency until when 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



(a) without optimization

(b) with optimization

Figure 5: duration of backup inconsistency

approximately 1%, and after that the duration of inconsistency increases rapidly as message loss rate increases. But for both cases, for the same message loss rate, the larger the window size, the shorter the backup stays in an inconsistent state. This is so because the frequency of update messages is determined 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.

Before we leave the performance analysis, 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 (and hence leaves maximum available resources at the primary for client request processing).

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 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, TTP [5] 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 TTP architecture provides fault-tolerance by implementing active redundancy. A Fault-Tolerant Unit (FTU) in a

TTP 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 [19] 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, 17] 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 [4, 7–12, 16, 18].

For example, a recent work [8] [10] 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 [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.

9 Conclusion

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

Avenues for future studies include extension of the concepts developed for real-time primary-backup replications to real-time state-machine replications and probabilistic analysis of temporal consistency under conditions that are not discussed in this paper.

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] 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.

[3] C-C Han and K-J Lin. Scheduling distance-constrained real-time tasks. In *Proc. Real-Time Systems Symposium*, December 1992.

[4] 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.

[5] H.Kopetz and G. Grunsteidl. Ttp - a protocol for fault-tolerant real-time systems. In *IEEE Computer*, volume 27, pages 14–23, January 1994.

[6] 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, Janruary 1991.

[7] 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.

[8] 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.

[9] T-W Kuo and A.K.Mok. Real-time database - similarity semantics and resource scheduling. In *ACM SIGMOD Record*, March 1997.

[10] 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.

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

[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, January 1994.

[15] G. S. Lueker. Two np-complete problems in nonnegative integer programming. Technical Report Report No. 178, Computer Science Laboratory, Princeton University, 1975.

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

[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*, pages 66–75, May 1995.

[18] 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.

[19] T.Abdelzaher, A.Shaikh, S.Johnson, F.Jahanian, and K.G.Shin. Rtcast: Lightweight multicast for real-time process groups. In *IEEE Real-Time Technology and Applications Symposium*, 1996.

[20] Hengming Zou and Farnam Jahanian. Real-time primary-backup replications with temporal consistency guarantees. In *IEEE Proceedings International Conference on Distributed Computing System*, pages 48–56, May 1998.