Window-Consistent Replication for Real-Time Applications *

Jennifer Rexford, Ashish Mehra, James Dolter and Farnam Jahanian

Real-Time Computing Laboratory
Department of EECS
The University of Michigan
Ann Arbor, MI 48109-2122

E-mail: {jrexford, ashish, jdolter, farnam}@eecs.umich.edu

Abstract

Two widely-studied approaches for structuring fault-tolerant services are the state-machine *and the* primary-backup *replication schemes. For a large class of soft and hard real-time applications, the degree of consistency among servers can be exploited to design replication protocols with* predictable *timing behavior. This is particularly useful in applications, such as automated process control, in which one can tradeoff the quality or precision for timely availability of data.*

This paper presents the architecture and prototype implementation of a primary-backup replication service that employs window consistency *semantics between the primary data repository and the backups. A client registers a data object with the service by declaring the consistency requirements for the data, in terms of a time* window. *The primary ensures that each backup site maintains a version of the object that was valid on the primary within the preceding time window by scheduling update messages to the backups.*

Decoupling the transmission of updates to the backups from the processing of client requests permits the primary to handle a higher rate of operations and provide more timely service to clients. The non-blocking semantics free the client from waiting for updates to the backups to complete. Furthermore, real-time scheduling of update messages can guarantee controlled inconsistency *between the primary and backup repositories.*

*The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the NSF. Also supported by a grant from the Rackham School of Graduate Studies at the University of Michigan.

1 Introduction

A common approach to building fault-tolerant distributed systems is to replicate servers that fail independently. The objective is to give the clients the illusion of a service that is provided by a single server. Two widely-studied approaches for structuring fault-tolerant services are the *primary-backup* and the *state-machine* replication schemes. In both approaches, a *modify* request from a client results in the execution of an agreement protocol that ensures consistency among the replicated servers. For example, in the traditional primary-backup model, each client modify request to the primary repository requires an update transmission to the backup. This approach artificially ties the rate of client modify operations to the rate of updates to the backups, limiting both response time predictability and total system throughput while ensuring consistent data after fail-over. For a large class of soft and hard real-time applications, this restriction can be more detrimental than having a slightly stale copy of the data on the backups.

This paper presents the architecture of a new primary-backup scheme, referred to as the *window-consistent replication service* [1], which exploits the ability of many real-time applications to tolerate controlled time-inconsistency of the repository to provide timely availability of this data. The notion of window consistency relaxes atomic or causal consistency among replicas to obtain less expensive replication protocols. In particular, the proposed replication scheme exploits *temporal constraints* on objects to maintain a less current but acceptable version of the primary data on the backup. A client registers a data object with the service by declaring the consistency requirements for the data, in terms of a time *window*.

The primary ensures that each backup site maintains a version of the object that was valid on the primary within the preceding time window by scheduling selective update messages to the backups. The time window essentially establishes a bounded distance between the primary and the backup states for each object. The following example illustrates the motivation for the proposed approach.

Example - Highly-Available Process Control System: Consider a primary-backup system for automated manufacturing and process control applications, as shown in Figure 1. The primary and the backup nodes share external devices such as sensors. The primary runs in a tight loop sampling sensors, calculating new values, and sending signal to external I/O under its control. The primary also maintains an in-memory data repository which is updated frequently during each iteration of the tight control-loop. One of the requirements on the system is to be able to switch to the backup in case of the primary failure within a few hundred milliseconds.

The in-memory data repository must be replicated on the backup to meet the strict timing constraint on the switch-over. Since there can be hundreds of updates to the data repository during each iteration of the control loop, it is impractical (and perhaps impossible) to update the backup synchronously each time the primary copy is changed. An alternative solution is to exploit the data semantics in a process control system by allowing the backup to maintain a less current but an acceptable copy of the data that resides on the primary. If the data on the backup does not fall too far behind the version on the primary, the backup can recover from a primary failure. For example, updates can be sent in batches or selectively to the backup. If the primary fails, the backup can take over even if the last few updates are lost. The objective is, however, to keep the backup data recent such that it can reconstruct a *consistent* system state by extrapolating from previous values and by reading new sensor values. However, one must ensure that the distance between the primary and the backup copies is bounded within a predefined time. In fact, different objects may have distinct tolerances in how far the backup can lag behind before the object state becomes stale. The challenge is to bound the distance between the primary and the backup such that consistency is not compromised while minimizing the overhead in exchanging messages between the primary and its backup. Under transient overload conditions, the system can gracefully degrade by allowing the backup to increase its distance from the primary. □

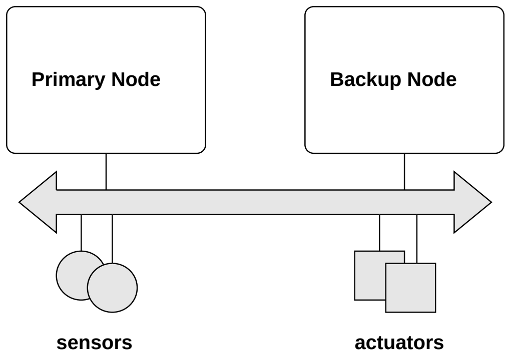


Figure 1: Primary-backup process control system

A replication scheme based on window consistency allows computations that may otherwise be disallowed by existing active or passive protocols that ensure atomic updates to a collection of replicas. Enforcing a weaker correctness criterion has been studied extensively for different purposes and application areas. In particular, a number of researchers have observed in the past that the notion of serializability is too strict a correctness criterion for real-time databases. Hence, several alternatives have been proposed that eliminate or relax serializability as a correctness criteria for managing consistency in real-time transactions. Among these are ϵ-serializability [2], similarity [3],temporal and external consistency [4], triggered real-time databases [5]. The above correctness criteria allow more concurrency by supporting a limited amount of inconsistency in how a transaction views the database state. The idea of imprecise computation is an interesting related approach that sacrifices accuracy for timeliness in real-time computations [6]. Exploiting weak consistency to obtain better performance has also been proposed in other non-real-time applications. For instance, in [7], the notion of quasi-copy is introduced which allows a weaker type of consistency between the central data and its cached copies at remote sites. The objective is to allow a cached copy to deviate from the central copy in a controlled way so that a scheduler has more flexibility in propagating updates. In the same spirit, the notion of window consistency provides controlled inconsistency among the replicas in a real-time application to support fault-tolerance.

2 Window Consistency

The window-consistent replication service consists of a primary and one or more backups, with the data

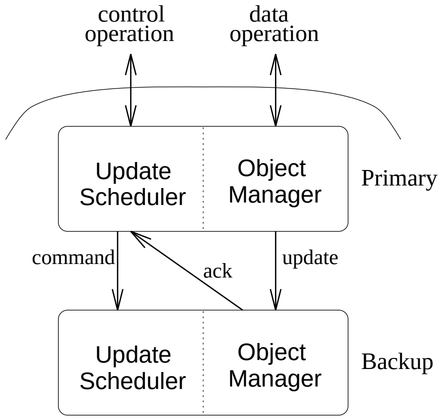


Figure 2: Window-consistent primary-backup architecture

on the primary shadowed at each backup. These sites store collections of objects which change over time, in response to client interaction with the primary. The primary handles client requests and ensures that each backup repository maintains a sufficiently recent version of the objects. In the absence of any failures, the primary satisfies all client requests and supplies a data-consistent repository. If the primary crashes, a window-consistent backup performs a fail-over to become the new primary by notifying the clients and other replicas of the new configuration. Service availability hinges on the existence of a window-consistent backup to replace a failed primary.

The service exports two sets of operations to the clients, namely, *control operations* and *data operations*, as shown in Figure 2. Control operations, such as object creation and deletion, are handled by the primary, and require complete agreement within the replication service. Client query and modify operations on the data are handled locally at the primary, without triggering interaction with the backups. The primary concurrently handles client requests and schedules selective updates to the backup repositories to guarantee the window consistency requirements of the data objects. The primary *object manager* (OM) satisfies client data requests, while sending update messages to the backup at the behest of the primary *update scheduler* (US).

Whenever the primary P modifies an object, P timestamps the new version; these timestamps identify successive versions of the object on the primary. The primary schedules transmissions to the backups to ensure that each backup has a sufficiently recent version of each object. At time t the primary has a copy of object O_i with timestamp $\tau_i^P(t)$, while a backup B

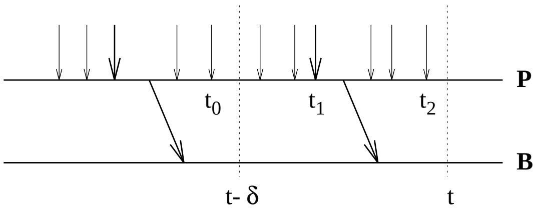


Figure 3: Window consistency semantics

stores a, possibly older, version $\tau_i^B(t)$. While B may have an older version of O_i than P, the copy on B must be "recent enough." The service must ensure that B always believes in data that was valid on P within the last δ_i time units. The value of δ_i is set during object creation, based on the time-consistency requirements of the application; P rejects the registration request if it cannot satisfy the object's window consistency requirements.

B has a *window-consistent* copy of object O_i at time t if and only if $\tau_i^P(t-\delta_i) \leq \tau_i^B(t) \leq \tau_i^P(t)$. For example, in Figure 3, P performs several operations on O_i, on behalf of client requests, but selectively transmits update messages to B. At time t the primary has the most recent version of the object, which was timestamped at time t_2; the backup has a version first recorded on the primary at time t_1. Thus, $\tau_i^P(t) = t_2$, $\tau_i^B(t) = t_1$, and $\tau_i^P(t-\delta_i) = t_0$. Since $t_0 \leq t_1 \leq t_2$, B has a window-consistent version of O_i at time t. The primary US employs real-time scheduling algorithms to coordinate the selective updates for the various objects.

The primary P needs an accurate measure of the window consistency of the backup objects to determine the operating mode of the backup sites. Although P cannot know the exact value of τ_i^B at all times, the primary can estimate the state of the backup. In particular, P does know the time t_i^{xmit} when it last transmitted an update for O_i. The primary sent version $\tau_i^{xmit} = \tau_i^P(t_i^{xmit})$ of the object to the backup, so P knows $\tau_i^B \leq \tau_i^{xmit}$. However, if P has not received an acknowledgement for this update, P cannot guarantee that B has this copy of O_i yet. If the backups tell P what object versions they have received, then the primary knows the latest version τ_i^{ack} that P has seen acknowledged by B. The primary then knows that $\tau_i^{ack} \leq \tau_i^B$. Using τ_i^{xmit} and τ_i^{ack}, P can provide both optimistic and pessimistic measures of the window consistency of O_i on B.

A backup also needs some measure of its own window consistency to determine whether it is a suitable substitute for a failed primary. Since a windowinconsistent backup B cannot supplant a crashed primary, B cannot tolerate long periods without hearing from P. The backups must balance the likelihood of false failure detection with the possibility of having no window-consistent backup to replace the crashed primary. Although B may be unaware of recent client interaction with P for each object, B does know τ_i^B and the time t_i^{xmit} when P transmitted this update. If less than δ_i time units have elapsed since P sent this update, then B has data that P believed within the last δ_i time units.

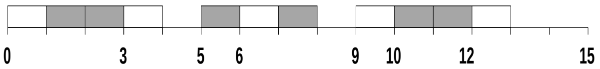
3 Application of Real-Time Scheduling

The window-consistent replication model satisfies a high rate of client query and modify operations by relaxing the consistency constraints between the primary and the backups. Under the limitations of finite processing time and network bandwidth, however, the primary must *schedule* the selective updates to the backup sites. By casting the transmissions of updates as tasks, the primary US can draw upon real-time task scheduling algorithms. While several task models can accommodate window-consistent scheduling, we initially consider the periodic task model [8, 9].

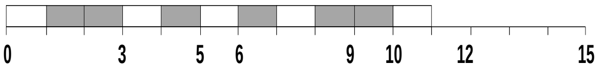
With the periodic model, the primary coordinates transmissions to the backups by scheduling an update task with period p_i and service time e_i for each object O_i [1]. The end of a period serves as both the deadline for one invocation of the task and the arrival time for the subsequent invocation. The scheduler always runs the ready task with the highest priority, preempting execution if a higher-priority task arrives. Rate-monotonic scheduling statically assigns higher priority to tasks with shorter periods [8, 9], while earliest-due-date scheduling favors tasks with earlier deadlines [8].

Given a schedulable set of tasks, the primary US ensures that O_i is sent to the backup once per period p_i, resulting in a maximum time of $2p_i$ between successive transmissions of O_i. The replication service must consider object consistency requirements and transmission delays in determining p_i for each object. In the absence of a link failure, we assume a bound d on the end-to-end latency between the primary and the backups. If a client operation modifies O_i, the primary must send an update for the object within the next $\delta_i - d$ time units; otherwise, the backups may not receive a sufficiently recent version of O_i before the time-window δ_i elapses. For window consistency this permits a maximum period $p_i = (\delta_i - d)/2$.

[1]The size of O_i determines the time e_i required for each update transmission. In order to accommodate preemptive scheduling and objects of various sizes, the primary can send an update message as one or more fixed-length *packets*.



(a) Periodic schedule:
$(p_1 = 5,\ e_1 = 2,\ p_2 = 3,\ e_2 = 1)$



(b) Compressed periodic schedule:
$(p_1 = 5,\ e_1 = 2,\ p_2 = 3,\ e_2 = 1)$

Figure 4: Compressing a periodic schedule

Compressing the periodic schedule: While the periodic model can guarantee sufficient updates for each object, the schedule updates O_i only once per period p_i, even if computation and network resources permit more frequent transmissions. This restriction arises because the periodic model assumes that a task becomes ready to run only at period boundaries. However, the primary can transmit the current version of an object at any time. The scheduler can capitalize on this task readiness to improve both resource utilization and the window consistency on the backups by *compressing* the periodic schedule. Consider two objects O_1 and O_2 as depicted in Figure 4. The scheduler must send an update requiring 1 unit of processing time once every 3 time units (unshaded box) and an update requiring 2 units of processing time once every 5 time units (shaded box). For this example, both the rate-monotonic and earliest-due-date algorithms generate the schedule shown in Figure 4(a). While each update is sent as required in the major cycle of length 15, the schedule has 4 units of slack time. The periodic schedule can provide the *order* of task executions without restricting the *time* the tasks become active. If no tasks are ready to run, the scheduler can advance to the earliest pending task and activate that task by advancing the logical time to the start of the next period for that object. With the compressed schedule the primary still transmits an update for each O_i at least once per period p_i but can send more frequent updates when time allows. As shown in Figure 4(b), compressing the slack time allows the schedule to start over at time 11. In the worst case, the compressed schedule degrades to the standard periodic schedule with the associated guarantees. **Integrating a new**

backup: To minimize the time the service operates without a window-consistent backup, the primary P needs an efficient mechanism to integrate a new or invalid backup. P must send the new backup B a copy of each object and then transition to the periodic schedule to sustain B. B must receive a copy of O_i in the "period" p_i before the periodic schedule begins; this ensures that B can afford to wait until the next p_i interval to start receiving periodic update messages for O_i. P guarantees a smooth transition to the periodic schedule by sending the objects to B in *reverse period* order, such that the objects with larger periods are sent before those with smaller periods. For object O_i, this ensures that only objects with smaller or equivalent periods can follow O_i in the integration schedule; these same objects can precede O_i in the periodic schedule. This guarantees that the integration schedule transmits O_i no more than p_i time units before the start of the periodic schedule, ensuring a consistent transition. The reverse-period integration schedule transmits a single copy of each object, minimizing the time required to establish window consistency on a new backup.

4 Prototype Implementation and Ongoing Work

We are developing a prototype implementation of the window-consistent replication service to demonstrate and evaluate the proposed service model. Each client or repository site is currently a Sun SPARCstation running Solaris 1.1. The sites communicate through UDP datagrams using the `Socket++` library from the University of Virginia, with extensions for *priority-based* access to the active sockets. The prototype implements rate-monotonic scheduling with compression. While Solaris 1.1 provides a stable environment for code development and testing, the platform does not support real-time thread scheduling, bounded communication delays, or synchronized clocks. After initial code development and testing, we will evaluate the service in a real-time distributed system [10].

The prototype provides a general framework for comparing the performance of different scheduling algorithms for coordinating update transmissions to the backups. We are currently investigating the distance-constrained task model [11] which assigns priorities based on separation restrictions. In addition, we are considering adaptive scheduling algorithms that incorporate knowledge of recent client interaction with the primary. These alternative models may permit more optimistic schedulability criteria (with some increased cost in scheduler complexity), allowing the replication service to accept objects with more demanding window consistency requirements.

Window consistency offers a framework for designing replication protocols with predictable timing behavior. By decoupling communication within the service from the handling of client requests, a replication protocol can handle a higher rate of query and modify operations and provide more timely response to clients. Scheduling the selective communication within the service provides bounds on the degree of inconsistency between servers. Although the current prototype implements the primary-backup replication model, we are exploring the application of window consistency to the state-machine approach to server replication. In addition, we are investigating the influence of controlled time-inconsistency on failure detection and recovery in replication protocols.

References

[1] A. Mehra, J. Rexford, J. Dolter, and F. Jahanian, "Window-consistent replication service," Technical report, Department of EECS, University of Michigan, April 1994.

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

[3] T.-W. Kuo and A. K. Mok, "Application semantics and concurrency control of real-time data-intensive applications," in *Proc. Real-Time Systems Symposium*, 1992.

[4] K.-J. Lin, F. Jahanian, A. Jhingran, and C. D. Locke, "A model of hard real-time transaction systems," Technical Report RC 17515, IBM T.J. Watson Reseach Center, January 1992.

[5] H. F. Korth, N. Soparkar, and A. Silberschatz, "Triggered real time databases with consistency constraints," in *Proc. of the 16th VLDB Conference*, August 1990.

[6] J. Liu, K.-J. Lin, W.-K. Shih, R. Bettati, and J. Chung, "Imprecise computations," *to appear in IEEE Proceedings*, January 1994.

[7] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data caching issues in an information retrieval system," *ACM Trans. Database Systems*, vol. 15, no. 3, pp. 359–384, September 1990.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[9] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. Real-Time Systems Symposium*, pp. 166–171, December 1989.

[10] K. G. Shin, D. D. Kandlur, D. L. Kiskis, P. S. Dodd, H. A. Rosenberg, and A. Indiresan, "A distributed real-time operating system," *IEEE Software*, pp. 58–68, September 1992.

[11] C.-C. Han and K.-J. Lin, "Scheduling distance-constrained real-time tasks," in *Proc. Real-Time Systems Symposium*, pp. 300–308, December 1992.