[LL88] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Control*, (77):1–36, 1988.

[LYI87] Y.-H. Lee, P. S. Yu, and B. R. Iyer. Progressive transaction recovery in distributed DB/DC systems. *IEEE Transactions on Computers*, C-36(8):976–987, August 1987.

[MG91] A. Moenkeberg and Weikum G. Conflict-driven load-control for the avoidance of data-contention threashing. In *Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan*, pages 632–639, April 1991.

[Mok91] A.K. Mok. Towards mechanization of real-time system design. In *Foundations of real-time computing: formal specifications and methods.* Kluwer Press, 1991.

[PR88] P. Peinl and A. Reuter. High contention in a stock trading database: A case study. In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 260–268, June 1988.

[SAJK91] N. Soparkar, A. Asthana, H. V. Jagadish, and P. Krzyzanowski. Architectural support for real-time database systems. In *Workshop on Architectural Support for Real-Time Systems, San Antonio, TX*, December 1991.

[Sin88] M. Singhal. Issues and approaches to design of real-time database systems. *ACM SIGMOD Record*, 17(1):19–33, March 1988.

[SKS91] N. R. Soparkar, H. F. Korth, and A. Silberschatz. Techniques for failure-resilient transaction management in multidatabases. Technical Report TR-91-10, The University of Texas at Austin, Computer Sciences Department, 1991. Submitted for publication.

[SL90] X. Song and J. W. S. Liu. Performance of multiversion concurrency control algorithms in maintaining temporal consistency. Technical report, University of Illinois at Urbana-Champaign, 1990.

[Son88] S.H. Son, editor. *ACM SIGMOD Record: Special Issue on Real-Time Databases.* ACM Press, March 1988.

[SRL88] L. Sha, R. Rajkumar, and J.P. Lehoczky. Concurrency control for distributed real-time databases. *ACM SIGMOD Record*, 17(1):82–98, March 1988.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, C-39(9):1175–1185, September 1990.

[SS90] N.R. Soparkar and A. Silberschatz. Data-value partitioning and virtual messages. In *Proceedings of the nineth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville*, pages 357–367, April 1990.

[Sta88] J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, pages 10–19, October 1988.

[SZ88] J.A. Stankovic and W. Zhao. On real-time transactions. *ACM SIGMOD Record*, 17(1):4–18, March 1988.

[WHMZ90] G. Weikum, C. Hasse, A. Moenkeberg, and P. Zabback. The COMFORT project: A comfortable way to better performance. Technical Report 137, ETH Zurich, 1990.

[C$^+$89] S. Chakravarthy et al. HiPAC: A research project in active, time-constrained database management. Technical Report XAIT-89-02, Xerox Advanced Information Technology, July 1989. Final report.

[CKKS89] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The case for safe RAM. In *Proceedings of the Fifteenth International Conference on Very Large Databases, Amsterdam*, pages 327–336, 1989.

[CP87] S. Ceri and G. Pelagatti. *Distributed Database Systems, Principles and Systems.* McGraw-Hill, New York, 1987.

[DLW90] S. Davidson, I. Lee, and V. Wolfe. Supporting real-time concurrency. In *Workshop on real-time operating systems and software*, pages 49 – 54, 1990.

[EGLT76] K. Eswaran, J. Gray, R. Lorie, and I. Traiger. The notion of consistency and predicate locks in a database system. *Communications of the ACM*, 1976.

[GR91] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, California, 1991. Manuscript; to be published.

[Gra78] J. N. Gray. Notes on database operating systems. In *Lecture Notes in Computer Science, Operating Systems: An Advanced Course*, volume 60, pages 393–481. Springer-Verlag, Berlin, 1978.

[Gra81] J. N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Seventh International Conference on Very Large Databases, Cannes*, pages 144–154, 1981.

[HCL90a] J.R. Haritsa, M.J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In *Proceedings of the Eleventh Real-Time Systems Symposium, Lake Buena Vista, Florida*, pages 94–103, December 1990.

[HCL90b] J.R. Haritsa, M.J. Carey, and M. Livny. On being optimistic about real-time constraints. In *Proceedings of the nineth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville*, pages 331–343, April 1990.

[KLS90] H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transactions. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane*, pages 95–106, August 1990.

[Koo90] G.M. Koob, editor. *Foundation of Real-Time Computing Research Initiative.* Office of Naval research, October 1990. Third Annual Workshop.

[KSS90] H. F. Korth, N.R. Soparkar, and A. Silberschatz. Triggered real-time databases with consistency constraints. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane*, August 1990.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lev91] E. Levy. Semantics-based recovery in transaction management systems. Ph.D. dissertation. Department of Computer Sciences, University of Texas at Austin, July 1991.

[Lis88] B. Liskov. Distributed programming in argus. *Communications of the ACM*, 31:300–312, March 1988.

[LKS91a] E. Levy, H. F. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. In *Proceedings of ACM-SIGMOD 1991 International Conference on Management of Data, Denver, Colorado*, pages 88–97, May 1991.

[LKS91b] E. Levy, H. F. Korth, and A. Silberschatz. A theory of relaxed atomicity. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1991.

Most of the previous research on real-time transaction management addresses the specific problems associated with combining concurrency control mechanisms and timing constraints directly. Our approach is different in that it localizes commitment and recovery, and translates global time-constraints to local ones for transaction executions. Employing this localization strategy enables us to consider a distributed RTDB to consist of several communicating centralized RTDBs. From the emerging principles of building such centralized RTDBs (e.g., see [Koo90, Son88, WHMZ90]), we have identified some of the problems that will impact on building a distributed RTDB, and we have also provided some partial solutions. Hence, we assume that the techniques for building the component RTDBs exist, and we base the characterization and solution of certain problems in a distributed RTDB on this assumption.

10 Conclusions

The problems associated with maintaining logically correct concurrent transaction executions while meeting their temporal constraints, are difficult to handle. In a distributed environment, the traditional correctness criteria that include atomicity of multi-site transactions compounds these difficulties since aborting the constituent local subtransactions cannot be achieved autonomously. This also impacts on the performance of real-time concurrency control protocols, and may cause problems of priority inversion.

By taking advantage of newly developed techniques for relaxing the logical correctness criteria, we are able to provide adaptive mechanisms for the commitment protocols that may be used to achieve coordination between the separate local executions. In case of time delays or transient overloads, these mechanisms can be adopted by autonomous local decisions. While these mechanisms compromise the traditional atomicity requirements, they ensure the relaxed criterion of semantic atomicity which is amenable to time-critical applications. The examples and suggestions for the implementation indicate that the approach may be a viable option for time-critical environments. Further study with specific applications is needed to gauge the potential of this approach.

References

[AGM88] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. In *Proceedings of the Fourteenth International Conference on Very Large Databases, Los Angeles*, pages 1–12, 1988.

[AGM89] R. Abbott and H. Garcia-Molina. Scheduling real time transactions with disk resident data. In *Proceedings of the Fifteenth International Conference on Very Large Databases, Amsterdam*, pages 385–396, 1989.

[AGM90] R. Abbott and H. Garcia-Molina. Scheduling I/O requests with deadlines: a performance evaluation. In *Proceedings of the Eleventh Real-Time Systems Symposium, Lake Buena Vista, Florida*, pages 113–125, December 1990.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[BMHD89] A.P. Buchmann, D.R. McCarthy, M. Hsu, and U. Dayal. Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency control. In *Proceedings of the Fifth International Conference on Data Engineering, Los Angeles*, pages 470–480, February 1989.

concerned sites. The correctness of this new approach is established using techniques similar to those discussed in [SKS91, Lev91]. The discussion of the overhead involved in handling sensitive transactions is outside the scope of this paper.

9 Discussions

Since it is important to initiate compensations as early as possible, compensating subtransactions should execute with a short deadline in a deadline-driven system. Furthermore, to invoke the compensatory rapidly, the local site that causes the entire transaction to abort could broadcast this information to all the sites instead of waiting for the coordinator to do so. There is the possibility that several compensating transactions may have to be executed during periods of very heavy overload — and we do not speculate on how that can be handled (since such situations correspond to a load beyond the capacity of the system).

Consider the log records necessary for the compensating transactions. In typical DBMSs, these are stored on stable storage, and reading them involves accessing secondary storage devices — a time-consuming activity. These delays may be alleviated through the use of large stable RAM devices (e.g., see [CKKS89, SAJK91]). The log buffer space in the main memory from which data is transferred to the stable storage should not be discarded immediately, and instead, used by the compensating processes. The technology trend of large main memories permits such an approach. The data corresponding to transactions that become semantically atomic can be discarded, and a mechanism similar to the scheme to discard markings (see Section 8) can be utilized. Also, since in an RTDB application the data that becomes outdated need not be moved to the disk at all, that part of the log buffer may be maintained entirely in a stable RAM device.

With regard to enforcing aborts at the time of overload, it is possible that there may be no time for an orderly undo of an aborted transaction. It is only by means of compensation that the system may postpone the undo to a light load period. This use of compensation differs from the one described thus far since the compensating transaction has to recover an incomplete forward transaction. The task of a compensating transaction is more difficult if the associated forward transaction is interrupted before completion. Hence, if we allow the preemption (i.e., the enforced abort) of the forward transaction to occur only at certain pre-defined *break-points*, the difficulties are likely to be alleviated (e.g., see [Son88]). The breakpoints decompose the transaction into logically coherent units of work that can be compensated-for on a completed-unit basis. When the transient overload passes, and the system returns to normal conditions, compensations can undo the completed units up to the break-point of preemption.

The scheme outlined in this paper is similar in several respects to the paradigms used in real-time systems. For example, notice that the scheme is typical of *optimistic* methods where it is assumed that untoward problems occur infrequently (e.g., see [HCL90b, Mok91, WHMZ90]). In such cases, at times of overload, the system adapts to a different mode of operation, and in doing so, must make certain sacrifices (e.g., see the *monitor baseline* approach of [Mok91], the *adaptive control* approach in [MG91], and *contingency plans* of [C+89]). Also, avoiding the problem of priority inversion (e.g., see [SRL90, SRL88]) is benefited if it becomes more feasible to perform rapid aborts — something that is often precluded by typical commit protocols due to blocking. It has been felt that real-time systems have special characteristics that make time-consuming mechanisms to ensure the ACID properties (see Section 3) for transactions unnecessary [Sta88, Son88]. In this regard, our paper presents an approach in which the correctness requirements may be relaxed on demand.

In a particular execution that uses our adaptive commitment approach, suppose that T_{12} aborts, whereas T_{11} commits locally. Compensation for T_{11} includes crediting by the amount a, whereas CT_{12} in this case is a simple abort. Consider a transaction T_2 that performs an audit at the two sites, C_1, and C_2 by reading the balances at each site. The serialization orders illustrated next exhibit the problem (each line represents the serialization order at a site from left to right).

- site C_1: T_{11} T_{21} CT_{11}
- site C_2: T_{12} CT_{12} T_{22}

This particular execution is of interest since at site C_1, T_2 is scheduled after T_1 and prior to CT_{11}, whereas at site C_2, T_2 is serialized to follow CT_{12}. Clearly, in the above scenario, T_2 reads a globally inconsistent state, where the amount a is incorrectly accounted.

The above problem arises because the notion of $\mathcal{R}$-commutativity allows a transaction to be affected by a committed subtransaction that is eventually undone by a compensating subtransaction (e.g., T_{21} being serialized after T_{11} in the above example). Let S_0, S_2 denote the initial states of the accounts at C_1 and C_2, respectively, and let S_1, S_3 denote the corresponding final states. Let $(T_{11} \circ CT_{11} \circ T_{21})(S_0) = S_4$. Although CT_{11} and T_{21} $\mathcal{R}$-commute, and as a result $S_4 \; \mathcal{R} \; S_1$, this does not change the fact that S_1 and S_3 do not satisfy a global consistency constraint of maintaining consistent total balances. Thus, the anomalous situation where T_2 is affected by both compensated-for and locally committed subtransactions cannot be rectified by $\mathcal{R}$-commutativity.

This problem arises because the relation $\mathcal{R}$ is based on local predicates alone, and does not guarantee global consistency among the distributed data items. We refer to transactions that require such a *global* consistency constraint to hold on the data they access as *sensitive* transactions. For a sensitive transaction T_s that interacts with a non-sensitive global transaction T_i, we require, in addition to the requirements of Section 4, that whenever there exists a path from T_{ip} to T_{sp} in SG_p, there should not be a path from CT_{iq} to T_{sq} in every other SG_q.

8.2 Handling Sensitive Transactions

The following scheme is compatible with the protocol of Section 7. We provide a simple marking of data items with respect to transactions. A data item accessed by T_i can be either *unmarked* or *marked* with respect to T_i — where a marked state has a connotation that T_i is undone. If a site decides to abort T_i, or if a compensating subtransaction is scheduled locally, then the data items accessed by T_i is marked with respect to T_i. Hence, a sensitive transaction is handled by ensuring that all data items accessed by T_s are either all marked with respect to T_i at sites common to T_i and T_s, or they are all unmarked with respect to T_i at those sites. To effect this, the vote to the coordinator to commit a sensitive transaction also includes the state of the local markings of the data items accessed. The coordinator validates the execution by the above rule. Hence, in such situations, a traditional 2PC protocol may be used to ensure the rule.

Discarding the markings necessitates a few additional message exchanges. This activity can be decoupled from the execution and commit procedure of the transactions, and may be done as a garbage-collection activity during periods of light loads and for several transactions together — so as to amortize the overhead. To effect this, a coordinator disseminates messages to a set of sites which respond by including the identity of all global transactions whose local compensating subtransactions completed successfully. Markings can be discarded for global transactions all of whose compensations have completed as may be determined by the information obtained from the for a subtransaction T_{ip} (corresponding to a global transaction T_i) executing at site C_p. In the above protocols, if site C_p has not yet sent a message indicating preparedness to commit to the coordinator of T_i, then T_{ip} may be unilaterally aborted. On the other hand, if that message has already been sent, then T_{ip} may be optimistically committed — the expectation is that the final decision regarding the fate of a global transaction will usually be to commit it. Notice that it would be incorrect to abort T_{ip} prior to receiving a final decision to abort from the coordinator of T_i since such an action would be contrary to the intent of the message sent by C_p that indicated a state of preparedness to commit T_{ip}.

Some important points regarding the above protocols need to be stated. Although the above scheme uses strict two-phase locking to guarantee serializability, similar mechanisms could be devised for other concurrency control techniques by using careful synchronization (e.g., see [SKS91]). Also, we note that in the techniques described above, concurrent global transactions may each use a different notion of atomicity. Furthermore, even for the same global transaction, the constituent subtransactions may actually be engaged in different commit protocols. This is important because not all subtransactions may be compensatable. For example, those involving *real actions* [Gra81], such as firing a weapon or dispensing cash, may not be compensatable.[8] Such subtransactions must always follow the traditional 2PC protocol, whereas its sibling subtransactions may continue to use an optimistic 2PC approach. Thus, the non-compensatable subtransactions are informed of a final decision to commit only after the coordinator ascertains that all the compensatable subtransactions also commit. This is achieved by using a complete version of the optimistic 2PC protocol [LKS91a] where each participant informs the coordinator after it commits its corresponding subtransaction.

8 Anomalous Behaviors

Our relaxed correctness criteria may not be sufficient for certain types of applications. The following example of a high-performance DBMS application illustrates the problem and motivates the subsequent solution.

8.1 The Problem

Consider a distributed environment consisting of sites each of which monitors the trends in the stock prices tracked at that site (e.g., see [PR88]). These trends are recorded locally, and also sent to some global coordinator to reach marketing-strategy decisions. The coordinator gathers the trends and makes decisions regarding the sale or buying of options, and sends these to the sites to effect the transactions. The need for a coordinated set of actions at the different sites is evident. Also, the need for fast, real-time responses is necessary to exploit the trends in the market [AGM89]. We assume that there is a need to transfer money from one site to another, or to audit the total amount of money that exists in accounts managed at more than one site. Such systems are usually designed using a transaction processing paradigm.

Consider a global transaction T_1 that transfers funds from an account A at site C_1 to an account B at site C_2. The decomposition of T_1 into local subtransactions is:

- T_{11} – debit account A by amount a
- T_{12} – credit account B by amount a

[8] This provides a reason why our scheme is not universally applicable to distributed transactions.

suffer from the following shortcomings with regard to distributed RTDBs. First, notice that once a participant indicates its preparedness to commit, it cannot allow the subtransaction in question to relinquish the locks held by committing or aborting until such time that the final decision is obtained from the coordinator. This is clearly problematic since it may cause other local subtransactions awaiting the decision to miss their deadlines. Second, if the final commit message arrives after the deadline for the local subtransaction has passed, the commitment must still be effected to achieve traditional atomicity. Also, note that since an indication of preparedness requires that the participant be in a position to change the database as dictated by the subtransaction in question despite failures, it is necessary to save the appropriate log records on stable storage prior to a notification of preparedness. Accessing stable storage is a time-consuming activity, and thus, there is an additional delay before the participant may notify its preparedness — however, this factor is not specific to real-time environments alone.

To alleviate the above problems, a protocol based on an optimistic 2PC protocol [LKS91a] may be used adaptively under adverse conditions. This protocol is similar to the traditional 2PC up to the point that the request for the state of preparedness is received by the participating sites.[6] At this point, the synchronization to ensure serializability is achieved, and the phase that follows ensures semantic atomicity. Due to the flexibility offered by semantic atomicity, no message need be sent to the coordinator unless the subtransaction is aborted. In the event that the subtransaction is aborted, the coordinator is alerted by a message in a manner similar to the case of simple synchronization of Section 6, and this causes the coordinator to trigger compensations at all sites that committed their subtransactions so as to maintain semantic atomicity.[7] We assume in this discussion that the requirements of $\mathcal{R}$-commutativity as detailed in Section 4 are met.

Notice that in the above protocol, locks may be released at any time after the first phase by simply aborting the subtransaction — which is not possible after the point that preparedness is guaranteed in the traditional 2PC protocol. Therefore, the problem of blocking due to remote failures or delays is avoided, and also, the same expedient of aborting the subtransaction may be employed in the case of a local transient overload. Furthermore, if a subtransaction attempting to commit fails to do so — say due to the lack of resources — then the site could choose to abort it, and to subsequently inform the coordinator. As regards the time delay in saving the log records, note that they can be saved at any time prior to the final commit for the subtransaction, and therefore, the delay does not affect the commitment as severely.

We now describe the adaptive strategy that assures semantic atomicity as a contingency measure. The idea is that in situations of overload, or when blocking becomes imminent, sites should decide locally to switch from the traditional 2PC to the optimistic 2PC. This decision may be taken at any time after the first phase of the 2PC protocol. The decision as to when exactly that should take place is an application-dependent. Note that an abort can always be effected unilaterally during the first phase of the 2PC protocol. In the event that the global coordinator decides to abort the entire transaction, compensating subtransactions may be executed at each site where the subtransactions in question were locally committed. This ensures semantic atomicity as defined in Section 4.

Our adaptive strategy provides a method to deal with the question of a fast approaching deadline

[6]Requisite changes need to be made if the indication of the last operation having been executed is combined with the indication of preparedness. These changes effectively render the protocol to be similar to the one described in the main text.

[7]As noted in the footnote in Section 6, the role of the coordinator may be eliminated in this regard.

the message overhead for synchronization in this mode is negligible since except for the exchange of data as necessitated by the execution of the global transaction, there is no overhead exchange of messages necessary. Moreover, for the baseline mode of operation, the desirable properties of traditional atomicity and serializability are maintained.

When a local site decides to abort a transaction T_i for any reason, it effectively adapts to a different mode of operation autonomously. The site sends a message to the coordinator informing it that it has aborted the corresponding local subtransaction under its control. Upon receiving such a message, the coordinator initiates the semantic abort of the entire transaction T_i.[4] This can be achieved by informing all the other participating sites to do so. Each site that receives a message indicating that the global transaction T_i is to be aborted, performs the one of the following actions. If the subtransaction in question is still active, then it is forcibly aborted. On the other hand, if the subtransaction has been committed, then a compensating subtransaction is invoked at that site. As discussed in Section 4, this approach cannot improve matters any further if the compensations do not occur within their time constraints.

7 Adaptive Atomicity with Sporadic Global Transactions

In systems where sporadic global transactions are possible, the simple approach described in Section 6 will not suffice since the serializability of the transaction executions will not be maintained. For such situations, note that the use of a commit protocol also provides the necessary synchronization as discussed in Section 3. Thus, simply by using commit protocols appropriately, serializable executions can be automatically provided if care is taken to synchronize the serialization events correctly.

For the baseline operation of the system, the 2PC protocol can be used to ensure the traditional notions of correctness. We illustrate its use in conjunction with the use of the strict two-phase locking scheme. The protocol works in the following manner. When the coordinator for a global transaction receives information from each participant that it has executed its last operation, it knows that the participating subtransactions have all acquired the necessary locks — and therefore, it effectively synchronizes all the subtransactions according to a distributed 2PL policy (e.g., see [BHG87, SKS91]). At this point, it sends a message to each participant requesting the state of preparedness to commit the corresponding subtransaction.[5] The receipt of this message concludes the first phase of the 2PC, wherein the synchronization of the subtransactions is effected.

The second phase of the protocol begins when a participant responds to the coordinator's request by indicating its preparedness to commit or its unilateral decision to abort the subtransaction in question. The coordinator decides to commit the transaction only if all the participating sites have indicated their willingness to commit. Otherwise, it decides to abort the transaction. This decision is conveyed to the participants when the coordinator sends its final decision message to each participant which indicated its preparedness, and the message is acted upon accordingly. Following this action, each subtransaction may release the locks it holds.

The above protocol guarantees the desirable ACID properties of correctness. It does, however,

[4]A coordinator need not be involved at all — instead, the site that aborts a subtransaction can itself broadcast the necessary information to all the participating sites.

[5]If the participants know *a priori* which operation is the last one, it is possible to have them send their state of preparedness along with the indication of having executed that last operation. However, such a message must be sent *after* the log records have been saved on stable storage.

aborted, T_{22} would have been prevented from positioning the gun in exactly the same manner. We do not speculate on what T_{22} does in such a situation for this example.

Finally, we define an appropriate relation $\mathcal{R}$ as follows:

$S_1 \mathcal{R} S_2 \equiv$ the last value in the sequence that is used as input to compute the value of x by extrapolation (in S_1) = the value of y (in S_2)

Observe that $S_1 \not\equiv S_2$ which is a consequence of guaranteeing only semantic atomicity rather than traditional atomicity. Also, notice that the above description holds regardless of whether T_{12} is committed locally, or it is committed following the completion of a 2PC protocol. Thus, the availability of a compensating transaction allows for the flexibility of making local decisions — and the importance of this feature becomes apparent below.

6 Simple Optimistic Synchronization

Now that the notions of correctness have been formally established, the adaptive commitment strategy can be described in a simple manner. This section describes a strategy that is limited in its applicability to sets of periodic global transactions.[3] Section 7 describes a more general approach.

Consider a set of global transactions that execute *periodically* in an RTDB. That is, one period consists of a fixed set of transactions, and the next one consists of another invocation of these transactions, and so forth. Assume that a global transaction T_i has local subtransactions $T_{ip}, T_{iq}, \ldots$, at the sites $C_p, C_q, \ldots$, respectively. At each site C_p, assume that the local concurrency control mechanisms order the various subtransactions of the global transactions according to their priority or deadline. Clearly, if these orders are compatible over all the sites, then the global transactions are serialized. This holds despite the possibility of arbitrary *local* transactions that may *not* be periodic (e.g., see [SKS91]). The present description assumes that each site ensures that the subtransactions executing under its control are serialized according to the order prescribed by the priorities or the deadlines (e.g., see [Son88, AGM90, HCL90b]) imposed on the various transactions. In such cases, it is not difficult to see that serializability holds in the absence of any irregularities in the order of the executions, and with no failures.

The optimistic situation described above may fail from time-to-time due to a system overload or delays due to any reason. In particular, a site may prefer to unilaterally abort a subtransaction for the purposes of scheduling a more urgent transaction, or because the deadline for the subtransaction may have passed. Notice that the atomicity of the global transactions is compromised (assuming that the other subtransactions for the global transaction are committed) — although serializability is still maintained. Our approach is to ensure semantic atomicity in such situations through the use of compensating transactions. That is, if a global transaction T_i needs to be aborted due to the failure of some of its subtransactions, compensating subtransactions are initiated at each site where a subtransaction of T_i committed. Thus, in cases where a subtransaction is compensatable, local commitment may be used.

The above ideas can be implemented in the simple case of periodic transactions by a coordinator of a global transaction T_i that optimistically assumes that all the subtransactions commit. This may be regarded as the normal baseline mode of operation for periodic transactions. Note that

[3]Local transactions may occur sporadically.

GT_2: A sequence of data elements at site C_2, each of which is global track data. This sequence records all global track data in chronological order with the most current one forming the head of the sequence. Each element in the sequence is associated with a time-stamp to specify how current that information is. Such a sequence is assumed to be stored at each site — possibly in the form of log records (which are used for standard recovery purposes [BHG87]).

Manipulation and access of GT_2 is done through the following:

- *tail*(*sequence*): Returns the tail of the sequence (i.e., all elements but the head).
- *head*(*sequence*): Returns the head element of the sequence.
- *extrapolate*(*sequence*): Computes and returns global track data which is the extrapolation of the sequence of global track data provided to it.

Next, we provide the pseudo-code for the compensating subtransaction CT_{12}:

```
$GT_2 \leftarrow tail(GT_2)$
**if** $GT_2$ was read by $T_2$ since it was updated by $T_{12}$ **then**
    **begin**
        $x \leftarrow extrapolate(GT_2)$
        set the weapon system according to the global track data $x$
    **end**
```

The first step of CT_{12} is simply to undo T_{12} by removing the head of GT_2. Observe that once this head element is removed, the time-stamp of the new head indicates that the value is outdated. Only if there are transactions that used the (erroneous) value of GT_2, is compensation actually needed. Checking this condition (i.e., "if GT_2 was read since...") can be done as part of the execution of CT_{12} by accessing the log records that contain GT_2. Actual compensation is performed by the routine that sets the weapon system based on the extrapolated value stored in the variable x. The aim is to try to set the weapon system based on its past trajectory since the local site does not know the precisely correct current position for it. Notice that the compensation only re-positions the system, and is not involved with firing the weapons — which may not be compensatable in this manner (see Section 7).

The pseudo-code for T_{22} is as follows:

```
$y \leftarrow head(GT_2)$
**if** the time-stamp of $y$ shows the value is up-to-date **then**
    set the weapon system according to the global track data $y$
```

Let X be the local execution before T_{12} at site C_2. Consider the following equations:

$$S_1 = (X \circ T_{12} \circ T_{22} \circ CT_{12})(S_0) \qquad (1)$$

$$S_2 = (X \circ T_{12} \circ CT_{12} \circ T_{22})(S_0) = (X \circ T_{22})(S_0) \qquad (2)$$

Equation 1 represents an execution where T_{12} was committed erroneously, and was later compens for. Equation 2 represents an execution where T_{12} was aborted on time and its effects were entirely undone. In the execution represented by Equation 2, T_{22} is unable to position the gun since it follows CT_{12}, which rendered outdated the new head value of GT_2. Observe that had T_{12} actually

4.3 Relaxed Temporal Correctness

Generally, compensations should be performed as early as possible to ameliorate the ill-effects of the erroneously committed compensated-for subtransactions. Hence, compensations should be constrained by the use of *soft* deadlines and *value functions* (e.g., see [AGM88]). For simplicity, in this paper, we assume that compensations must be executed within a pre-specified time interval *after* the completion of the forward transaction with which they are associated. This time interval may be application-dependent. Thus, a deadline is also placed on the completion of a compensating subtransaction depending on the completion time of the forward transaction. Note that it is possible that the application places *no* deadline on the compensating subtransaction — in which case, an infinite deadline may be assumed.

The above paradigm poses a problem. Even if it were the case that each site could guarantee the timely execution of a compensating transaction after its invocation, the impossibility result for the Two Generals' problem [Gra78] implies that the invocation itself may not occur in a timely manner. This is because the invocation of the compensating subtransaction is effected by the reception of a message sent from another site, and that message may be delayed. In such situations, a late compensation corresponds to a temporally incorrect execution, and that cannot be avoided due to the impossibility result.[1] Thus, we state that relaxed temporal correctness is maintained if all committed transactions (including compensating transactions) meet their respective deadlines.

4.4 Relaxed RTDB Correctness

We are in position now to define the notion of relaxed RTDB correctness in a manner similar to the traditional RTDB correctness of Section 3. We say that distributed RTDB transaction executions maintain relaxed correctness if they maintain the relaxed criteria for logical and temporal correctness.

5 Using Compensation

We illustrate the notion of compensation and $\mathcal{R}$-atomicity by referring back to the example in Section 2. For the ease of presentation, we shall be concerned here only with the setting of the weapon system. The following transactions[2] are defined:

T_1 : The global transaction that collects and correlates local track data, and disseminates the resultant global track data to the various sites. Suppose that the erroneous reading occurs in subtransaction T_{11} at site C_1. Thus, subtransaction T_{12} at site C_2 contains the recording of the erroneous correlated data into the local database.

T_{22}: A local weapon positioning transaction at site C_2.

CT_{12}: The subtransaction for T_1 at site C_2 that performs compensatory actions at that site.

We now consider the compensating subtransactions and the concept of $\mathcal{R}$-commutativity in this context at site C_2. We shall use the following data variable:

[1]This is one reason why our approach is only a partial solution to the problems associated with atomic commitment.

[2]We emphasize that the example is chosen more to illustrate the concepts rather than to suggest a specific domain of use.

The reader is referred to [Lev91] for more details regarding the design and implementation of compensating transactions.

In a distributed RTDB, each global transaction is decomposed into a collection of local subtransactions, each of which performs a semantically coherent task at a single site. The subtransactions are selected from a well-defined library of routines at each site. For global transactions that can be compensated-for, each forward subtransaction is associated with a pre-defined compensating subtransaction. With regard to concurrency control, there is no difference between the compensating subtransactions and any other subtransactions.

We assume that compensating for a global transaction need not be coordinated as a global activity (e.g., similar to [LYI87]). Consequently, there is no need to use a commit protocol for the termination of the compensating subtransactions that correspond to a particular global transaction. The compensating subtransactions are assumed to have no inter-dependencies, and share no global information. This is important since we require that the local sites should be able to run the compensations autonomously. Situations where this may not hold are described in Section 8.

4.2 Relaxed Logical Correctness

Our correctness criterion is stated in terms of serialization graphs (SGs) that are a slightly extended version of the standard SGs. To make the presentation uniform, we model an aborted subtransaction as a committed subtransaction followed immediately by the corresponding compensating transaction that simply undoes the committed subtransaction. This use of a compensating subtransaction is simply a syntactic device to model an aborted subtransaction, and it does not imply that every subtransaction has a compensating subtransaction defined (aside from the one used to model an abort).

Let $\mathcal{T}$ be a set of global transactions. Let C_p be a site with the set of subtransactions, T_p corresponding to $\mathcal{T}$, and the set $\mathcal{CT}_p$ corresponding to the set of compensating subtransactions. As mentioned above, aside from the compensating subtransactions used to model an aborted subtransaction, not all elements of T_p need necessarily have a corresponding element in $\mathcal{CT}_p$ (also see Section 7). The *local serialization graph* at site C_p for a complete local history H_p (see [BHG87] for precise definitions of complete histories) is a directed graph $SG_p(H)$=(V_p, E_p). The set of nodes V_p consists of a subset of transactions in $\mathcal{T}_p \cup \mathcal{CT}_p$. An edge $A \rightarrow B$ is in E_p if and only if one of A's operations precedes and conflicts with one of B's operations in H_p.

A *global SG* is the union of all the local serialization graphs. For a set of local SGs, represented by $SG_p = (V_p, E_p)$, the corresponding global SG is defined as $SG_{global} = (\cup\ V_p,\ \cup\ E_p)$. Observe that each compensating subtransaction is assigned a separate node in the global SG (in accord with the localized execution of compensating subtransactions as described above).

A global history is logically correct if the following two conditions hold:

- **Serializability.** The global SG is acyclic.

- **Semantic Atomicity.** For each transaction T_i, either all local subtransactions are committed — thereby committing T_i, or for each committed subtransaction of T_i, the corresponding compensating transaction is executed at some point following the commitment of the subtransaction in question — thereby aborting T_i.

maintain logical correctness and all committed transactions complete before the expiration of their deadlines.

4 Relaxed Correctness

We now elaborate on the issue of relaxation of the traditional correctness notions for distributed RTDBs. First, we relax the logical correctness criteria by introducing compensating transactions and semantic atomicity. This leads to the relaxation of the temporal correctness criteria as well.

4.1 Compensating Transactions

A *compensating* transaction is a recovery transaction that is associated with a specific *forward* transaction that is committed, and whose effects must be undone without causing cascading aborts. The purpose of compensation is to "undo" a forward transaction *semantically* without causing cascading aborts. The intention of compensation is to leave, as far as possible, the effects of transactions that follow the forward transaction intact, and yet preserve database consistency. Compensation guarantees that a *consistent* state is established based on semantic information. The state of the database after compensation takes place may only approximate the state that would have been reached, had the forward transaction never been executed. In [KLS90, Lev91] we have formally characterized the outcome of compensation based on the properties of the forward transaction and the transactions that follow it in the execution, and the necessary details are summarized as follows.

A *database* is a set of data *objects* whose values at any instant of time constitute a state S. A transaction is a function from states to states. An *execution* imposes a serialization order among a set of concurrently executing of transactions. An execution, thus, defines both a total order among the transactions, as well as a function from states to states that is the functional composition (denoted by '$\circ$') of the transactions. That is, $X = T_1 \circ \ldots \circ T_n$ denotes the function from states to states defined by applying the functions denoted by the transactions in the same order X. We use $X(S)$ to denote the state resulting from applying the function X to the state S.

To formalize the notion of compensation, we use a binary relation between the transactions as follows. Two transactions, T_1 and T_2, commute with respect to a relation $\mathcal{R}$ on states (in short, $\mathcal{R}$-commute), if for all states S, $(T_1 \circ T_2)(S)\ \mathcal{R}\ (T_2 \circ T_1)(S)$. If $\mathcal{R}$ is the equality relation, the two transactions commute in the usual mathematical sense. For a forward transaction T and its compensating transaction CT, the execution $T \circ X \circ CT$ is atomic with respect to $\mathcal{R}$ (in short $\mathcal{R}$-atomic), if $T \circ X \circ CT(S)\ \mathcal{R}\ X(S)$. If a compensating transaction is serialized immediately after its associated forward transaction, then compensation amounts to the traditional undoing of the forward transaction; formally, $(T \circ CT)(S) = S$. If CT $\mathcal{R}$-commutes with each of the transactions mentioned in X, then the execution $T \circ X \circ CT$ is $\mathcal{R}$-atomic. The relation $\mathcal{R}$ serves to constrain CT, thereby preventing it from violating consistency constraints and other desirable predicates established by the transactions executing in the system. Thus, the relation should ensure that some desirable properties, such as, "if a consistency constraint predicate holds on the final state of X, then it should also hold on the final state of $T \circ X \circ CT$", are maintained in the system.

The design of a compensating transaction is an application-dependent task. Thus, the forward transaction must record enough information in the database for the compensating transaction to execute properly. This information is similar to that saved for standard transaction recovery.

Physical clocks are more important for the purposes of this paper since the applications that we deal with interact closely with the external physical environment (e.g., see [KSS90]), and since time-constraints should be measured by physical clocks.

In this paper we assume the existence of strongly synchronized physical clocks in the sense of [Lam78]. That is, each clock should closely approximate the passage of physical time. Also, the variance between the times measured by different clocks should be very small. The former can be assured by hardware, while the latter is managed by the frequent exchange of messages. We assume that the message traffic and the speed of the network are both sufficiently high so as to guarantee a negligible variance.

The above description does not address the issue of unexpected failures or delays since it is assumed that the underlying clock synchronizing mechanism ensures resilience to the various failures, and hence after a failure, re-synchronization occurs rapidly (e.g., see [LL88]). These are research issues beyond the scope of this paper.

Given our notion of time as described above, it is reasonable to assume that a global time t corresponds, within acceptable tolerance bounds, to time t_p as measured by the local clock at site C_p. Thus, we may ascribe a time-constraint to a global transaction T_i that will correspond to the same time-constraint for each of its subtransactions as measured at their local sites. That is, each time-critical global transaction T_i is assigned a deadline t by which it should commit. This corresponds to requiring that each subtransaction T_{ip} of T_i that executes at a site C_p, should commit by time t as measured by t_p at its local site. We define the actual commit time of the global transaction T_i to correspond to the largest among the commit times of each of its subtransactions. Similarly, we can consider time-critical local transactions to have deadlines as measured locally. As explained below, we shall impose a *hard* deadline policy of requiring that these deadlines be met, or the transaction in question should be aborted.

3.2 Logical Correctness

Database transactions executing *correctly* in the conventional sense are characterized by their atomicity, consistency, isolation, and durability properties [BHG87, CP87] — commonly referred to as the ACID properties. Correctness for concurrent transaction management is defined by *serializability* [EGLT76]. A serializable execution is guaranteed to preserve the *consistency* of the database, regardless of the specifics of the consistency constraints.

In the case of distributed DBMSs, ensuring serializability requires the synchronization of the executions at the different sites. This is accomplished through the exchange of messages to synchronize the serialization events (e.g., lock points in *two-phase locking*, or time-stamps in time-stamp ordering [BHG87]). Since ensuring atomicity also requires such message exchanges (in the form of the commit protocol), the same protocol usually also serves for this synchronization. Thus, in the case of distributed DBMSs, the commit protocol achieves the two objectives of ensuring the atomicity and the serializability of the transaction executions (e.g., see [SKS91]).

3.3 Traditional RTDB Correctness

Having described the traditional notions of temporal and logical correctness for our purposes, we are now in a position to state a simple criterion for traditional correctness in an RTDB environment. Distributed RTDB transaction executions are said to maintain traditional correctness if they a standard transaction-processing system, the execution of a commit protocol (e.g., 2PC) ensures that all the subtransactions of the aborted global transaction do indeed abort.

The price paid, however, for employing a standard commit protocol is severe if the system is in a crisis situation, or if there is an overload. Blocking may cause a situation where none of the sites have a recent global track data, which is undesirable at a time of crisis. Waiting for the coordinator's final decision may unnecessarily cause missing the deadlines of urgent transactions when the system is overloaded. The fast *local commit* of a subtransaction would be much more suitable under these circumstances — optimistically assuming that the global transactions usually commit. However, uncoordinated local commitment may cause some sites to commit the erroneous global track data they receive, and subsequently to expose the data to other transactions. Thus, for example, a transaction that positions the weapon system at a site may base its computation on the prematurely committed, and hence inaccurate, global track data. Therefore, there is a need to recover from the effects of the incorrectly committed data by compensatory actions. In our example, the compensatory actions re-position the weapon system based on the past history of the execution.

Our proposed scheme supports the local commit of subtransactions sooner than is prescribed by the 2PC protocol, thereby avoiding the blocking problems associated with the 2PC protocol. If the early commit turns out to be premature and erroneous, compensatory actions are used to obtain a relaxed degree of atomicity. Such trading of standard atomicity for earlier commitment, however, is exercised only when required (i.e., during overload periods, or if the slack time available is small). The concepts of compensation and relaxed atomicity are formally defined in Section 4.

Given the above scheme, a natural question to ask is why the 2PC protocol should be used at all, and why not *always* use a protocol that uses compensatory actions. The answer is that it is always desirable to preserve standard logical correctness criteria. For instance, in the above example, it is preferable not to have any incorrect data recorded rather than to try and correct the effects of erroneous actions. Second, for certain types of transactions (described in Section 8), it is anyway necessary to use a standard protocol.

3 Traditional Correctness

In this section we explain the traditional correctness criteria for concurrent real-time transactions. The concepts used in this paper regarding time and temporal correctness in a distributed environment are first described, and then combined with the traditional logical correctness criteria for concurrent transactions. Together, these criteria constitute the traditional correctness for real-time database (RTDB) transaction executions. Traditional correctness serves as the notion of correctness in the norm. In the next section we define a concept for relaxed correctness that serves as a contingency measure for periods of transient overload or unprecedented delays in the system.

3.1 Temporal Issues in a Distributed Environment

In order to discuss the time-constraints and temporal correctness, we first describe a simplified notion of time for a distributed real-time system environment. There exists a body of literature — not all germane to real-time systems — that pertains to *clock synchronization* in distributed systems (e.g., see [Lam78, LL88]). This literature mainly deals with *logical* clocks as opposed to *physical* clocks. The latter count events in both the physical world as well as the computer system.

designed, are very application-specific. Finally, we would like to emphasize that our scheme only *alleviates* some of the problems discussed above, and cannot *eliminate* them.

The remainder of this paper is organized as follows. In Section 2, we motivate our ideas by means of an example application scenario. Sections 3 and 4 describe the traditional and relaxed correctness criteria for executions in normal baseline and overload modes, respectively. In Section 5, we continue with our example to illustrate these defined criteria. Section 6 describes our technique for the simple case of periodic transactions. Section 7 extends these ideas to the case of sporadic transactions. A subtle problem that may arise when using our techniques is described in Section 8, and a solution for it is also provided. Section 9 discusses certain implementation suggestions. Finally, Section 10 constitutes the conclusions.

2 Illustration of the Concepts

We provide a hypothetical example to illustrate our ideas — the specific details of the example are less important as compared to the general principles that it is meant to convey. Consider a tracking system for mobile targets (adapted from the examples in [Son88, Koo90]). Assume that there are several processing sites that manage target-sensors, target-tracking guns, and store data pertaining to the readings, positions, etc. in the local database management systems (DBMSs). Periodically, the sensors update the data regarding the targets as sensed at each local site, and this data is also sent to a specific coordinator site. The coordinator site receives track data from several sites and correlates the information gathered to create the global tracking information. It is necessary to do the correlation since the data obtained at each site is individually insufficient to identify the targets accurately [Son88]. The globally correlated data is also disseminated among the sites, and this data affects local decisions at the sites. Finally, global decisions may be taken sporadically to fire the guns located at the various distributed sites.

This application can be designed using a transaction processing paradigm. Henceforth, we use the term *global* transaction to refer to multi-site transactions. Global transactions are decomposed to a set of subtransactions executing at a different site each. The term *local* transaction is reserved for transactions (or subtransactions) that execute at any single, local site.

We assume that at each site, local transactions update the local track data (e.g., see *external-input* transactions of [KSS90]). Also, we assume that collection and correlation of the local track data from the different sites, and the dissemination of the global track data, together constitute one type of global transaction. The reading of the local track data and subsequent writing of the global track data at each site constitute the local subtransaction for the global transaction. There may be other local and global transactions in the system which need not concern us. Notice that it is necessary for each site to execute the subtransactions corresponding to a global transaction at approximately the same time — that is, at times that are closely synchronized. This is important to facilitate correct correlation since a temporally coherent view of the world must be used (e.g., see [SL90]). The same holds true for the dissemination of the global track data since each site should get the latest information so as to effect correct local decisions.

Suppose that an erroneous local track is recorded at one of the stations — perhaps due to a malfunctioning sensor. This fault may be detected only after the local track data is collected and correlated with (correct) track data from other sites (but before the corresponding global track is committed). Consequently, erroneous global track data may be generated and disseminated to several sites. Such a global transaction should be obviously be aborted as soon as possible. In awaiting, then the data held at that site for the concerned transaction may be *blocked* (i.e., become inaccessible until the necessary message is received).

More pertinent to real-time applications is the question of dealing with the time constraints on the subtransactions. For instance, a hard real-time deadline may exist on the execution of a subtransaction T_i, implying that T_i must be either committed prior to its deadline, or be aborted. If, however, it happens that the 2PC protocol blocks the subtransaction, then the deadline may pass without a decision having been reached for T_i. Hence, if the final outcome does arrive late, and it is to commit, then the subtransaction T_i *has* to be committed to preserve logical correctness — despite the fact that the deadline has passed. A slightly different problem is that the local concurrency control may dictate that T should be aborted in favor of other transactions with higher priorities, and that may be impossible to achieve for similar reasons. In fact, there is no way to overcome this problem by any standard atomic commitment protocol [BHG87, SS90, SKS91].

While it is advisable to use the standard correctness criteria where possible so as to accrue their many advantages [GR91, BHG87], it is fact that the problems noted above cannot be tolerated in typical real-time applications. Hence, in order to effectively utilize the transaction paradigm in a time-constrained environment, some of the stringent correctness requirements of transaction management must be *relaxed* (e.g., see [Sin88, Sta88]). In this paper, we adapt the notion of relaxed atomicity from [LKS91a, LKS91b, Lev91] and apply it to the realm of real-time transaction processing, concentrating on the issue of atomicity of real-time transactions in a distributed system. Our approach permits the system to choose a commitment strategy (other than 2PC) dynamically.

The key to our approach lies in the idea of *compensation*. If a transaction T commits "erroneously" (i.e., it is discovered *ex post facto* that T should actually have been aborted), a compensating transaction CT for T is used to perform a "semantic undo" of T [KLS90]. This undo returns the database to a consistent state that is equivalent, in an application-specific semantic sense, to a state resulting from an execution in which T never executed. The key concept here is that the compensation is accomplished without resorting to cascading aborts.

A feature of compensation that makes its use attractive for real-time systems is that compensatory actions may be deferred, while traditional undo operations need to be performed immediately. This allows the execution of the recovery process during periods of light system load despite the expectation that transaction failures (and thus recovery) will occur disproportionately more often during times of system overload. Moreover, it is not necessary for a transaction T to hold data pending the execution of CT. Instead, it can release data that is later (re-)acquired by CT. We allow standard 2PC to be used as the norm for transaction commitment, with compensation-based techniques invoked only when time-constraints require it; this further reduces the overhead associated with commitment.

There are several issues that are *not* discussed in this paper mainly due to reasons of simplicity, space, and application-specificity. First, the application-specific decision of the particular point at which one commitment strategy is chosen over another, is one such issue. Second, failures that are essentially catastrophic in a real-time environment — such as a system crash that results in the loss of volatile memory (thereby necessitating the access of slow secondary storage), is another issue. Instead, we are more concerned with the delay failures that are always possible since there is little that can be done for catastrophic situations that is specific to real-time environments. Suffice to say that the techniques used for typical DBMSs are assumed to apply in the environments of concern for this paper [BHG87]. Third, the issue of the performance analysis of our proposed scheme is also not within the scope of this paper because the work loads, and compensations that can be

1 Introduction

As the volume of data managed by real-time applications increases, there is incentive to apply database system concepts in the development of real-time systems [Son88, Koo90, AGM88, KSS90]. The transaction paradigm, in particular, seems to be applicable to real-time systems [GR91, DLW90, SZ88, Lis88, Koo90, SL90]. However, the timing requirements of real-time systems appear to preclude the use of transaction management techniques — at least in their current form. Among the obstacles to using transaction management technology in real-time systems are:

- High variance in access time for a data object that makes it difficult to place a reasonable bound on transaction response times (e.g., see [BMHD89, KSS90]).
- Concurrency control requirement of achieving high overall throughput do not permit the incorporation of real-time constraints on the execution of individual transactions (e.g., see [BMHD89, Son88]).
- Standard recovery techniques that guarantee the atomic execution of a transaction by restricting other transactions (e.g., delaying access to uncommitted data) are not suited for the characteristics of real-time applications (e.g., see [Sin88, LKS91b]).

The above problems are exacerbated in a distributed environment. The reasons are that the control of the executions is distributed over several agents located at different sites, and due to the fact that unqualified communication guarantees are never available.

Initial studies addressing the concurrency control aspects of time-constrained transaction management in general (e.g., see [KSS90, AGM88, HCL90b]), and in a distributed environment in particular (e.g., see [SRL88, SL90, Sta88, Sin88, Son88]), are reported in the literature. In this paper, we are concerned mainly with problems of ensuring atomicity and meeting the time constraints in a distributed real-time transaction management system where the correctness criterion is serializability. Our approach represents a departure from the norm in that we do not directly address the question of time-constrained scheduling at each local site. Instead, we consider issues pertinent to the distributed environment where the local scheduling is handled through the use of methods that have been previously explored [AGM88, AGM89, BMHD89, $C^{+}89$, HCL90b, HCL90a, KSS90]. Thus, a transaction that accesses data at several sites, does so by the use of a *subtransaction* at each concerned site, and each such subtransaction is subject to the local concurrency control mechanism at its particular site. We adopt such an approach since the key feature that distinguishes distributed (as compared to centralized) transaction management is the coordination of the distributed loci of control, which is achieved through the use of an atomic commit protocol that assures that the fate of a transaction T_i (i.e., whether it is committed or aborted), must be agreed upon by all sites where the subtransactions of T_i access data.

The standard approach to distributed transaction management is to use the *two-phase commit* (2PC) protocol (e.g., see [BHG87]), where a transaction is executed under the control of a centralized coordinator. Consensus is reached by sites agreeing to abide by the decision of the coordinator either to commit the transaction or to abort it. Once a site indicates to the coordinator that it is in a position to commit if necessary, it makes the promise to adhere to the decision of the coordinator. That is, in the second phase of the protocol, a site can neither abort nor commit unilaterally. Therefore, if for any reason a site does not obtain the final message of the 2PC protocol it was

Adaptive Commitment for Real-Time Distributed Transactions [*]

Nandit Soparkar[†]
Eliezer Levy[‡]
Henry F. Korth[§]
Avi Silberschatz[†]

Abstract

Real-time distributed transaction management systems are useful for both real-time and high-performance database applications. Guaranteeing response times in such environments is difficult to achieve mainly due to the inherent asynchrony present. The standard approach to distributed transaction management is to employ the two-phase locking scheme in each of the participating sites, and to coordinate the executions of the various subtransactions through the use of the two-phase commit protocol. Such an approach ensures the atomicity and serializability properties of the transactions. Unfortunately, the unpredictability, the cost and the fault-tolerance properties of the two-phase commit protocol render it unsuitable for real-time applications.

The approach taken in this paper is to identify ways in which a commit protocol can be made adaptive in the sense that under situations that demand it, such as a transient local overload, the system can dynamically change to a different commitment strategy. The decision to do so can be taken autonomously at any site. The different commitment strategies exploit a trade-off between the cost of commitment and the obtained degree of atomicity. The inexpensive protocols incur a reduced cost as they are based on the optimistic assumption that transactions failures are the exception rather than the rule. When transactions do fail, these protocols rely on local compensatory actions to recover from non-atomic executions. We provide the necessary framework to ensure the logical and temporal correctness criteria, and describe examples to illustrate the use of our strategies.

Keywords

Transaction management; Real-time; Distributed systems; Concurrency control; Fault-tolerance; Adaptive control

[*]This material is based in part upon work supported by the Texas Advanced Technology Program under Grant No. ATP-024, the National Science Foundation under Grant Nos. IRI-9003341 and IRI-9106450, and grants from the IBM, NEC, and Hewlett-Packard corporations. Most of this work was done while the second and the third authors were with the University of Texas at Austin.

[†]Department of Computer Sciences, University of Texas, Austin, TX 78712-1188 USA, Fax (512) 471-7028, {soapy,avi}@cs.utexas.edu, (512) 471-9718 & (512) 471-9706.

[‡]Logic CAD IDC-2, Intel Israel (74) Ltd., M.T.M. Scientific Industries Center, Haifa 31015 P.O.B. 1659 Israel, eliezer@iil.intel.com, (972) 4 355026.

[§]Matsushita Information Technology Laboratory, 182 Nassau St., 3rd Floor, Princeton, NJ 08542-7072 USA, Fax (609) 497-4013, hfk@mitl.com, (609) 497-4600.

ADAPTIVE COMMITMENT FOR DISTRIBUTED REAL-TIME TRANSACTIONS

Nandit Soparkar
Eliezer Levy
Henry F. Korth
Abraham Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-92-15 April 1992