In Proceedings of DART'96: ACM CIKM Workshop on Databases: Active & Real-Time, Rockville, Maryland, November 1996.

Timely and fault-tolerant data access from broadcast disks: A pinwheel-based approach*

Sanjoy Baruah
sanjoy@cs.uvm.edu
Department of CS & EE
University of Vermont

Azer Bestavros
best@cs.bu.edu
CS Department
Boston University

October 30, 1996

Abstract: *A combination of* AIDA *(the Adaptive Information Dispersal Algorithm) and* pinwheel scheduling *is proposed as a solution to the problem of designing data access strategies in environments where (i) the data is frequently updated at a server, and needs to be made available to clients upon demand, (ii) the server-to-client bandwidth far exceeds the bandwidth from any individual client back to the server, and (iii) timely retrieval of data is essential, even in the presence of transmission or other errors.*

1 Introduction

Mobile computers are likely to play an important role at the extremities of future large-scale distributed real-time databases. One such example is the use of on-board automotive navigational systems that interact with the database of an Intelligent Vehicle Highway System (IVHS). IVHS systems allow for automated route guidance and automated rerouting around traffic incidents by allowing the mobile vehicle software to query and react to changes in IVHS databases [24, 23]. Other examples include wearable computers for soldiers in the battlefield and computerized cable boxes for future interactive TV networks and video-on-demand. Such systems are characterized by the significant discrepancy between the *downstream* communication capacity from servers (*e.g.* IVHS backbone) to clients (*e.g.* vehicles) and the *upstream* communication capacity from clients to servers. This discrepancy is the result of: (1) the huge disparity between the transmission capabilities of clients and servers (*e.g.*, broadcasting via satellite from IVHS backbone to vehicles as opposed to cellular modem communication from vehicles to IVHS backbone), and (2) the scale of information flow (*e.g.*, thousands of clients may be connecting to a single computer for service). Moreover, the limited power capacity of some mobile systems (*e.g.*, wearable computers) requires them to have no secondary I/O devices and to have only a small buffer space (relative to the size of the database) that acts as a cache for the information system to which the mobile system is attached.

Broadcast disks: The concept of *Broadcast Disks* (Bdisks) was introduced by Zdonik *et al.* [31] as a mechanism that uses communication bandwidth to emulate a storage device (or a memory hierarchy in general) for mobile clients of a database system. The basic idea is to exploit the abundant bandwidth capacity available from a server to its clients by *continuously and repeatedly* broadcasting data to clients, thus in effect making the broadcast channel act as *a set of disks* (hence the term "Broadcast Disks") from which clients could fetch data *"as it goes by."* Work on Bdisks is different from previous work in both wired and wireless networks [14, 22] in that several sources of data are multiplexed and broadcast to clients, thus creating a hierarchy of Bdisks with different sizes and speeds. On the server side, this hierarchy gives rise to memory management issues (*e.g.*, allocation of data to Bdisks based on priority/urgency). On the client side, this hierarchy gives rise to cache management and prefetching issues (*e.g.*, cache replacement strategies to improve the hit ratio or reduce miss penalty). In [4], Acharya, Franklin and Zdonik discuss Bdisks organization issues, including client cache management [1], client-initiated prefetching to improve the communication latency for database access systems [3], and techniques for disseminating updates [2].

Real-time considerations: There are many reasons for subjecting Bdisk data retrieval to timing constraints. Perhaps the most compelling is due to the

*This work has been partially supported by the NSF (grants CCR-9308344 and CCR-9596282).

absolute temporal consistency constraints [28] that may be imposed on data objects. For example, the data item in an Airborne Warning and Control System (AWACS) recording the position of an aircraft with a velocity of 900 km/hour may be subject to an absolute temporal consistency constraint of 400 msecs, in order to ensure a positional accuracy of 100 meters for client transactions (*e.g.* active transactions that are fired up to warn soldiers to take shelter). Notice that not all database object will have the same temporal consistency constraint. For example, the constraint would only be 6,000 msecs for the data item recording the position of a tank with a velocity of 60 km/hour. Other reasons for imposing timing constraints on data retrieval from a Bdisk are due to the requirements of database protocols for admission control [10], concurrency control, transaction scheduling [26], recovery [21], and bounded imprecision [29, 30].

The real-time constraints imposed on Bdisks protocols become even more pressing when issues of fault-tolerance are to be considered. Current Bdisks protocols assume that the broadcast infrastructure is not prone to failure. Therefore, when data is broadcast from servers to clients, it is assumed that clients will succeed in fetching that data as soon *"as it goes by."* The result of an error in fetching data from a Bdisk is that clients have to wait until this data is re-broadcast by the server. In a real-time environment, waiting for a complete retransmission may imply missing a critical deadline, and subjecting clients to possibly severe consequences.

In [9], Bestavros showed how to allocate data items to Bdisks so as to mask (or otherwise minimize) the impact of intermittent failures in a real-time environment. In that respect, he proposed the use of the Adaptive Information Dispersal Algorithm (AIDA) [8], which allows for a controllable and efficient tradeoff of bandwidth for reliability, and derived lower bounds on the bandwidth requirements for AIDA-based fault-tolerant real-time Bdisks.

In this paper, we explore the issue of integrating AIDA and *pinwheel scheduling* [18] to design Bdisk layouts that offer real-time guarantees even in the presence of faults. In Section 2, we provide a brief introduction to AIDA. In Section 3, we model the data layout problem on Bdisks as a pinwheel scheduling problem; further, we observe that a simple version of fault-tolerance requirements can also be so modelled with little additional effort. In Section 4, we define a more general concept of fault-tolerance than the ones considered in [9], and discuss extensions to pinwheel scheduling theory that are necessary in order to design Bdisk layouts which possess this generalized form of fault-tolerance.

2 Information Dispersal and Retrieval

Let F represent the original data object (hereinafter referred to as the *file*) to be communicated (or retrieved). Furthermore, assume that file F is to be communicated by sending N independent transmissions. Using Rabin's IDA algorithm [27], the file F can be processed to obtain N distinct blocks in such a way that recombining *any* m of these blocks, $m \leq N$, is sufficient to retrieve F. The process of processing F is called the *dispersal* of F, whereas the process of retrieving F by collecting m of its pieces is called the *reconstruction* of F. Figure 1 illustrates the dispersal, communication, and reconstruction of an object using IDA. Both the dispersal and reconstruction operations can be performed in real-time. This was demonstrated in [7], where an architecture and a CMOS implementation of a VLSI chip that implements IDA was presented.

In our research, we assume that broadcasted blocks are self-identifying. In particular, each block has two identifiers. The first specifies the data item to which the block belongs (*e.g.*, this is page 3 of object Z). The second specifies the sequence number of the block relative to all blocks that make-up the data item (*e.g.*, this is block 4 out of 5). This is necessary so that clients could relate blocks to objects, and more importantly, to allow clients to correctly choose the inverse transformation when using IDA.

Adaptive information dispersal and retrieval: Several fault-tolerant *redundancy-injecting* protocols have been suggested in the literature. In most of these protocols, redundancy is injected in the form of parity blocks, which are only used for error detection and/or correction purposes [13]. The IDA approach is different in that redundancy is added *uniformly*; there is simply *no* distinction between data and parity. It is this feature that makes it possible to scale the amount of redundancy used in IDA. Indeed, this is the basis for the *adaptive* IDA (AIDA) [8]. Using AIDA, a *bandwidth allocation* operation is inserted after the dispersal operation but *prior* to transmission. This bandwidth allocation step allows the system to *scale* the amount of redundancy used in the transmission. In particular, the number of blocks to be transmitted, namely n, is allowed to vary from m (*i.e.* no redundancy) to N (*i.e.* maximum redundancy).

The reliability and accessibility requirements of various data objects in a distributed real-time appli-

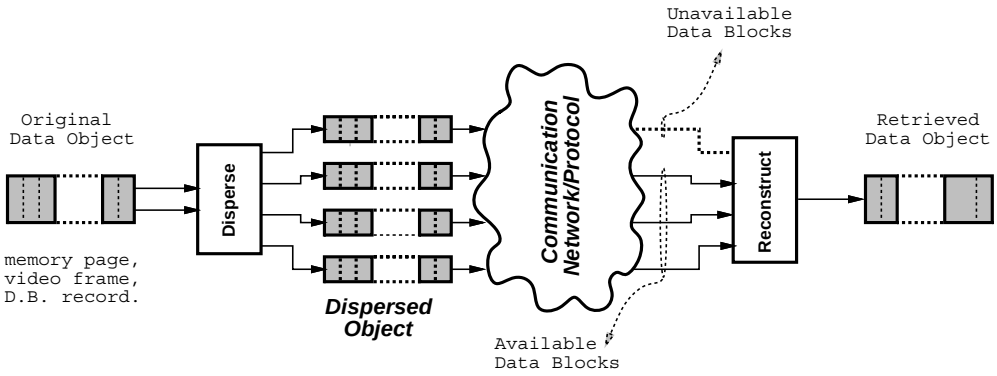

Figure 1: Dispersal and reconstruction of information using IDA.

cation depend on the system *mode* of operation. For example, the fault-tolerant timely access of a data object (*e.g.*, "location of nearby aircrafts") could be critical in a given mode of operation (*e.g.*, "combat"), but less critical in a different mode (*e.g.*, "landing"), and even completely unimportant in others. Using the proposed AIDA, it is possible to dynamically adjust the reliability and accessibility profiles for the various objects (files) in the system by controlling their level of dispersal. In other words, given the requirements of a particular mode of operation, servers could use the bandwidth allocation step of AIDA to scale down the redundancy used with unimportant (*e.g.*, non-real-time) data items, while boosting it for critical data items.

3 Pinwheel Task Systems

Pinwheel task systems were introduced by Holte et al. [18], in the context of offline scheduling for satellite-based communication. Since its introduction, this task model has been used to model the requirements of a wide variety of real-time systems. For example, Han & Lin [15] have used pinwheel techniques to model distance-constrained tasks; Hsueh, Lin, and Fan [20] have extended this research to distributed systems. Baruah, Rosier, and Varvel [6] have used pinwheel scheduling to construct static schedules for sporadic task systems. Recently, Han & Shin [16, 17] have applied pinwheel techniques to real-time network scheduling.

Consider a shared resource that is to be scheduled in accordance with the *Integral Boundary Constraint*: for each integer $t \geq 0$, the resource must be allocated to exactly one task (or remain unallocated) over the entire time interval $[t, t+1)$. (We refer to this time interval as *time slot* t.) For our purposes, a *pinwheel task* i is characterized by two positive integer parameters – a *computation requirement* a and a *window size* b – with the interpretation that the task i needs to be allocated the shared resource for at least a out of every b consecutive time slots. The ratio of the computation requirement of a task to its window size is referred to as the *density* of the task. The density of a system of tasks is simply the sum of the densities of all the tasks in the system. Observe that for a task system to be schedulable, it is necessary (although not sufficient) that the density of the system be at most one.

The issue of designing efficient scheduling algorithms for pinwheel task systems has been the subject of much research. Holte et al [19] presented an algorithm which schedules any pinwheel task system of two tasks with density at most one. Lin & Lin [25] have designed an algorithm which schedules any pinwheel task system of three tasks with a density at most five-sixth's (this algorithm is optimal in the sense that there are three-task systems with density $5/6 + \epsilon$ that are infeasible, for ϵ arbitrarily small). When the number of tasks is not restricted, Holte et al [18] have a simple and elegant algorithm for scheduling any pinwheel task system with density at most one-half. Chan and Chin [12, 11] have significantly improved this result, designing a series of algorithms with successively better density bounds, culminating finally in one that can schedule any pinwheel system with a density at most $7/10$ [11].

Pinwheel scheduling for Bdisks: Suppose that a broadcast file F_i is specified by a *size* $m_i \in \mathbf{N}$ in blocks and a *latency* $T_i \in \mathbf{N}$ in seconds. Given $F_1, F_2, \ldots, F_n$, the problem of determining minimum bandwidth (in blocks/sec) reduces to determining the smallest $B \in \mathbf{N}$ such that the system of pinwheel tasks $\{(1, m_1, BT_1), (2, m_2, BT_2), \ldots, (n, m_n, BT_n)\}$

can be scheduled. Since the algorithm of Chan and Chin [11] can schedule any pinwheel task system with density at most $7/10$, $\sum_{i=1}^{n} \frac{m_i}{BT_i} \leq \frac{7}{10}$ is sufficient for this purpose. That is, a bandwidth

$$B = \left\lceil \frac{10}{7} \sum_{i=1}^{n} \frac{m_i}{T_i} \right\rceil$$

is sufficient; since $\sum_{i=1}^{n} \frac{m_i}{T_i}$ is clearly necessary, this represents a reasonably efficient upper bound, in that at most 43% extra bandwidth is being required. Furthermore, this upper bound is easily and efficiently realised — given this much bandwidth, the scheduling algorithm of Chan and Chin [11] can be used to determine the actual layout of blocks on the Bdisk.

Fault-tolerant pinwheel scheduling for Bdisks: Suppose now that each file F_i is characterized by a fault-tolerance requirement r_i in addition to m_i and T_i. That is, access to the file is expected to be accomplished with a latency of at most T_i seconds even in the presence of up to r_i faulty blokcs. Determining the minimum bandwidth for such systems is again not difficult — as above, we can derive

$$B = \left\lceil \frac{10}{7} \sum_{i=1}^{n} \frac{m_i + r_i}{T_i} \right\rceil$$

and argue that this is again efficient with an at most 43% overhead cost.

4 Generalized Fault-tolerant Real-Time Bdisks

In certain applications, it may be desirable to associate with each file different latencies depending upon the occurrence and severity of faults. Thus, we may want very small latency under normal circumstances, but be willing to live with a certain degradation in performance when faults occur. This model is examined below.

Model and Definitions: Let us assume that the available bandwidth is known. A *generalized fault-tolerant real-time broadcast file* F_i is specified by a positive integer size m_i and a positive integer latency vector $\vec{d_i} \stackrel{\text{def}}{=} [d_i^{(0)}, d_i^{(1)}, \ldots, d_i^{(r_i)}]$, with the interpretation that it consists of m_i blocks, and the worst-case latency tolerable in the presence of j faults is equal to the time required to transmit $d_i^{(j)}$ blocks, $0 \leq j \leq r_i$.

It is important to note that the generalized fault-tolerant real-time Bdisks constitute a generalization of the broadcast disk models discussed above. "Regular" real-time Bdisks — those with real-time but no fault-tolerance constraints — are represented in this model by setting r_i to zero for each file. "Regular" fault-tolerant real-time Bdisks — those with both real-time and fault-tolerance constraints — may be represented by setting all the latencies of a file equal to each other: $d_i^{(0)} = d_i^{(1)} = \ldots = d_i^{(r_i)}$.

We would like to map the problem of scheduling such generalized fault-tolerant Bdisks to related problems in pinwheel scheduling. Unfortunately, it turns out that this mapping is not as straightforward as in the case of regular Bdisks.

We start with some definitions:

1. A *broadcast program* P for a system of n files $F_1, F_2, \ldots, F_n$ in a generalized Bdisks system is a function from the positive integers to $\{0, 1, \ldots, n\}$, with the interpretation that $P(t) = i$, $1 \leq i \leq n$, iff a block of file F_i is transmitted during time-slot t, and $P(t) = 0$ iff nothing is transmitted during time-slot t.
2. $P.i$ is the sequence of integers t for which $P(t) = i$.
3. Broadcast program P satisfies *broadcast file condition* $\mathsf{bc}(i, m_i, \vec{d_i})$ iff $P.i$ contains at least $m_i + j$ out of every $d_i^{(j)}$ consecutive positive integers, for all $j \geq 0$, where $\vec{d_i} \stackrel{\text{def}}{=} [d_i^{(0)}, d_i^{(1)}, \ldots, d_i^{(r_i)}]$ is a vector of positive integers.
4. Broadcast program P satisfies *pinwheel task condition* $\mathsf{pc}(i, a, b)$ iff $P.i$ contains at least a out of every b consecutive positive integers.
5. Broadcast program P satisfies *a conjunct of (pinwheel task or broadcast file) conditions* iff it satisfies each individual condition.
6. Let S_1 and S_2 be (broadcast/ pinwheel/ conjunct) conditions. We say that $S_1 \Rightarrow S_2$ iff any broadcast program satisfying S_1 also satisfies S_2. We say $S_1 \equiv S_2$ iff $S_1 \Rightarrow S_2$ and $S_2 \Rightarrow S_1$.

Observe that constructing a broadcast schedule for a given set of files $F_1, F_2, \ldots, F_n$, with F_i characterized by size m_i and latency vector $\vec{d_i}$, is exactly equivalent to determining a broadcast program that satisfies $\bigwedge_{i=1}^{n} \mathsf{bc}(i, m_i, \vec{d_i})$.

From the definitions of broadcast file condition and pinwheel task condition (the $\mathsf{bc}()$ and $\mathsf{pc}()$ conditions above), we obtain

$$\mathsf{bc}(i, m_i, \vec{d_i}) \equiv \bigwedge_{j \geq 0} \mathsf{pc}(i, m_i + j, d_i^{(j)}) \quad . \qquad (1)$$

Lemma 1 follows as a direct consequence:

Lemma 1 The problem of constructing a broadcast schedule for $F_1, F_2, \ldots, F_n$ is equivalent to the following pinwheel scheduling problem: Determine a broadcast program that satisfies

$$\bigwedge_{i=1}^{n} \left(\bigwedge_{j \geq 0} \mathsf{pc}(i, m_i + j, d_i^{(j)}) \right) \qquad (2)$$

■

Obtaining broadcast programs for generalized Bdisks: Recall that Chan and Chin [11] have designed an algorithm for scheduling any system of pinwheel tasks that has a density of at most 0.7. In our notation, this algorithm determines a P satisfying $[\mathsf{pc}(1, a_1, b_1) \wedge \mathsf{pc}(2, a_2, b_2) \wedge \ldots \wedge \mathsf{pc}(n, a_n, b_n)$, provided $(\sum_{i=1}^{n} a_i/b_i) \leq 0.7$.

An important observation about this algorithm of Chan and Chin [11] is that *it can only schedule pinwheel task systems where each task is constrained by a single pinwheel condition.* That is, we do not have any i such that both $\mathsf{pc}(i, a, b)$ and $\mathsf{pc}(i, a', b')$ must be satisfied.

Definition 1 A conjunct of pinwheel conditions $\bigwedge_{i=1}^{n} \mathsf{pc}(k_i, a_i, b_i)$ is **nice** if and only if $k_j \neq k_\ell$ for all $j \neq \ell$.

Since the Chan and Chin algorithm can only determine schedules satisfying nice conjuncts of pinwheel conditions, it is necessary that we reformulate Equation 2 into a nice form if we are to be able to use the Chan and Chin algorithm. That is, we are looking to convert a conjunct of pinwheel conditions on a single task into either a single pinwheel condition, or to a conjunct of pinwheel conditions on several tasks, such that these new conditions imply the original ones. Since the test of [11] is density-based, we would like to be able to perform such a conversion while causing the minimum possible increase in the density of the system. That is, we are attempting to solve the following problem:

> **Conversion to nice pinwheel:** Given a conjunct of pinwheel conditions, determine a nice conjunct of pinwheel conditions of minimum density which implies the given conjunct.

This seems to be a very difficult problem — indeed, we conjecture that it is NP-hard. In [5], we present several heuristic rules for obtaining a nice conjunct of pinwheel conditions that implies a given conjunct of pinwheel conditions; the design of more and better such heuristics remains an open and potentially fruitful research area. All these rules guarantee that the nice conjunct will in fact imply the given conjunct; further, they all attempt to obtain a minimal-density nice conjunct.

5 Conclusion

With the advent of mobile computers and cellular communication, it is expected that most clients in large-scale distributed environments will have limited storage capacities. More importantly these clients will have a limited upstream bandwidth (if any) for transferring information to servers, as opposed to a large downstream broadcast bandwidth for receiving information from servers. The significant asymmetry between downstream and upstream communication capacities, and the significant disparity between server and client storage capacities have prompted researchers to suggest the use of the downstream bandwidth as a "broadcast disk", on which data items that may be needed by clients are continuously and repeatedly transmitted by servers. The execution of critical tasks in such asymmetric client-server environments requires that data retrievals be *successfully* completed before some set *deadlines*. Previous work on broadcast disks did not deal explicitly with the fault-tolerance and timeliness constraints imposed by such critical tasks. In this research, we have defined a formal model for the specification of fault-tolerance and real-time requirements for broadcast disk files. We have shown a close link between the design of broadcast programs for such disks and the previously studied problem of pinwheel scheduling.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of ACM SIGMOD conference*, San Jose, CA, May 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. In *Proceedings of VLDB'96: The 1996 International Conference on Very Large Databases*, India, September 1996.

[3] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

[4] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6), December 1995.

[5] S. Baruah and A. Bestavros. Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems. Technical Report TR-1996-023, Department of Computer Science, Boston University, August 1996.

[6] S. Baruah, L. Rosier, and D. Varvel. Static and dynamic scheduling of sporadic tasks for single-processor systems. In *Proceedings of the Third Euromicro Workshop on Real-time Systems*, June 1991.

[7] Azer Bestavros. SETH: A VLSI chip for the real-time information dispersal and retrieval for security and fault-tolerance. In *Proceedings of ICPP'90, The 1990 International Conference on Parallel Processing*, Chicago, Illinois, August 1990.

[8] Azer Bestavros. An adaptive information dispersal algorithm for time-critical reliable communication. In Ivan Frisch, Manu Malek, and Shivendra Panwar, editors, *Network Management and Control, Volume II*. Plenum Publishing Corporation, New York, New York, 1994.

[9] Azer Bestavros. AIDA-based real-time fault-tolerant broadcast disks. In *Proceedings of RTAS'96: The 1996 IEEE Real-Time Technology and Applications Symposium*, Boston, Massachusetts, May 1996.

[10] Azer Bestavros and Sue Nagy. Value-cognizant admission control for rtdbs. In *Proceedings of RTSS'96: The 16^{th} IEEE Real-Time System Symposium*, Washington, DC, December 1996.

[11] M. Y. Chan and Francis Chin. Schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, June 1992.

[12] M. Y. Chan and Francis Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9:425–462, 1993.

[13] Garth Gibson, Lisa Hellerstein, Richard Karp, Randy Katz, and David Patterson. Coding techniques for handling failures in large disk arrays. Technical Report UCB/CSD 88/477, Computer Science Division, University of California, July 1988.

[14] David Gifford. Ploychannel systems for mass digital communication. *Communications of the ACM*, 33, February 1990.

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

[16] C. C. Han and K. G. Shin. A polynomial time optimal synchronous bandwidth allocation scheme for the time-token MAC protocol. In *Proceedings of IEEE INFOCOMM'95*, April 1995.

[17] C. C. Han and K. G. Shin. Real-time communication in FieldBus multiaccess networks. In *Proceedings of the Real-Time Technology and Applications Symposium*, May 1995.

[18] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference on System Science*, pages 693–702, Kailua-Kona, January 1989.

[19] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1):105–135, 1992.

[20] C. W. Hsueh, K. J. Lin, and N. Fan. Distributed pinwheel scheduling with end-to-end timing constraints. In *Proceedings of the Real-Time Systems Symposium*, December 1995.

[21] Jing Huang and Le Gruenwald. An update-frequency-valid-interval partition checkpoint technique for real-time main memory databases. In *Proceedings of RTDB'96: The 1996 Workshop on Real-Time Databases*, pages 135–143, Newport Beach, California, March 1996.

[22] T. Imielinski and B. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37, October 1994.

[23] IVHS America. IVHS architecture development program: Interim status report, April 1994.

[24] R.K. Jurgen. Smart cars and highways go global. *IEEE Spectrum*, pages 26–37, May 1991.

[25] S. S. Lin and K. J. Lin. Pinwheel scheduling with three distinct numbers. In *Proceedings of the EuroMicro Workshop on Real-Time Systems*, Vaesteraas, Sweden, June 1994.

[26] Özgür Ulusoy and Alejandro Buchmann. Exploiting main memory dbms features to improve real-time concurrency protocols. *ACM SIGMOD Record*, 25(1), March 1996.

[27] Michael O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[28] Krithi Ramamritham. Real-time databases. *International journal of Distributed and Parallel Databases*, 1(2), 1993.

[29] Wei-Kuan Shih, Jane Liu, and Jen-Yao Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM journal of Computing*, July 1991.

[30] V. Fay Wolfe, L. Cingiser DiPippo, and J. K. Black. Supporting concurrency, timing constraints and imprecision in objects. Technical Report TR94-230, University of Rhode Island, Computer Science Department, December 1994.

[31] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya. Are 'disks in the air' just pie in the sky? In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.