Proceedings of ICDE'97: The IEEE International Conference on Data Engineering, April 1997, Birmingham, England.

Pinwheel Scheduling for Fault-tolerant Broadcast Disks in Real-time Database Systems*

Sanjoy Baruah
The University of Vermont
Department of Computer Science
sanjoy@cs.uvm.edu

Azer Bestavros
Boston University
Computer Science Department
best@cs.bu.edu

Abstract

The design of programs for broadcast disks which incorporate real-time and fault-tolerance requirements is considered. A generalized model for real-time fault-tolerant broadcast disks is defined. It is shown that designing programs for broadcast disks specified in this model is closely related to the scheduling of pinwheel task systems. Some new results in pinwheel scheduling theory are derived, which facilitate the efficient generation of real-time fault-tolerant broadcast disk programs.

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 [20, 19]. 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.* [27] 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 (illustrated in Figure 1) 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 [13, 17] 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 [2], 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 [4], and techniques for disseminating updates [3].

Previous work in Bdisks technology was driven by

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

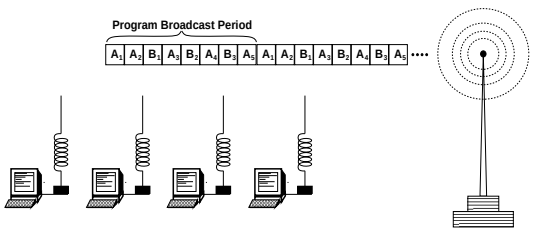

Figure 1. The Concept of Broadcast Disks

wireless applications and has concentrated on solving the problems associated with the limited number of uplink channels shared amongst a multitude of clients, or the problems associated with elective disconnection (as an extreme case of asymmetric communication), when a remote (*e.g.* mobile) client computer system must pre-load its cache before disconnecting. Problems that arise when timing *and* reliability constraints are imposed on the system were not considered.

Real-time considerations: There are many reasons for subjecting Bdisk data retrieval to timing constraints. Perhaps the most compelling is due to the absolute temporal consistency constraints [24] 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 objects 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 [9], concurrency control, transaction scheduling [22], recovery [16], and bounded imprecision [25, 26].

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. For non-real-time applications, such a mishap is tolerable and is translated to a longer-than-usual latency, and thus deserves little consideration. However, in a real-time environment, waiting for a complete retransmission may imply missing a critical deadline, and subjecting clients to possibly severe consequences.

This research: The contributions of this paper are twofold. First, we show that the problem of designing real-time Bdisk programs is intimately linked to the pinwheel scheduling problem [14], and make use of this link to (1) derive upper bounds on the bandwidth requirements for real-time fault-tolerant Bdisks (corresponding to the lower bounds in [8]), and (2) obtain efficient algorithms for designing fault-tolerant real-time Bdisk programs. Next, we present a more general model for real-time fault-tolerant Bdisks that subsumes the simple model presented in [8]. We derive a pinwheel algebra—some simple rules for manipulating pinwheel conditions—and demonstrate through examples how these rules may be used to efficiently construct broadcast programs for generalized fault-tolerant real-time Bdisks.

The rest of this paper is organized as follows. In Section 2, we discuss the basics of AIDA-based organization of Bdisks for timeliness and fault tolerance as proposed in [8]. In Section 3, we review pinwheel scheduling theory, and describe how AIDA-based Bdisks are related to pinwheel systems. In Section 4, we introduce the concept of generalized real-time fault-tolerant Bdisks, and describe pinwheel-based procedures for organizing data on such disks.

2. AIDA-based Bdisks

We model a Bdisks system as being comprised of a set of data items (or files) that must be transmitted continuously and periodically to the client population. Each data item consists of a number of blocks. A block is the basic, indivisible unit of broadcast (*e.g.*, page). We assume that the retrieval of a data item by a client is subject to a time constraint imposed by the real-time process that needs that data item.

When an error occurs in the retrieval of one or more blocks from a data item, then the client must wait for a full broadcast period before being able to retrieve the erroneous block. This broadcast period may be very long since the broadcast disk may include thousands of other blocks, which the server must transmit before getting back to the block in question. For real-time systems, such a delay may result in missing critical timing constraints. In [8], Bestavros proposed the use of AIDA to mask (or otherwise minimize) the impact

of such failures in a real-time environment. AIDA is a technique for dynamic bandwidth allocation, which makes use of minimal, controlled redundancy to guarantee timeliness and fault-tolerance up to *any* degree of confidence. AIDA is an elaboration on the Information Dispersal Algorithm of Rabin [23].

2.1. 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, 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. Both the dispersal and reconstruction operations can be performed in real-time [6].

The dispersal and reconstruction operations are simple linear transformations using *irreducible polynomial arithmetic.*[1] The dispersal operation shown in Figure 2 amounts to a matrix multiplication (performed in the domain of a particular irreducible polynomial) that transforms data from m blocks of the original file into the N blocks to be dispersed. The N rows of the transformation matrix $[x_{ij}]_{N \times m}$ are chosen so that any m of these rows are mutually independent, which implies that the matrix consisting of any such m rows is not singular, and thus invertible. This guarantees that reconstructing the original file from *any* m of its dispersed blocks is feasible. Indeed, upon receiving any m of the dispersed blocks, it is possible to reconstruct the original data through another matrix multiplication as shown in Figure 2. The transformation matrix $[y_{ij}]_{m \times m}$ is the inverse of a matrix $[x'_{ij}]_{m \times m}$, which is obtained by removing $N - m$ rows from $[x_{ij}]_{N \times m}$. The removed rows correspond to dispersed blocks that were not used in the reconstruction process. To reduce the overhead of the algorithm, the inverse transformation $[y_{ij}]_{m \times m}$ could be precomputed for some or even all possible subsets of m rows.

In this paper, we assume that broadcasted blocks are self-identifying.[2] 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 $[y_{ij}]_{m \times m}$ when using IDA.

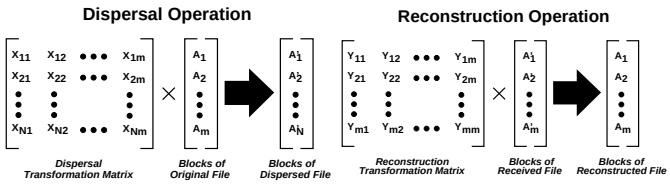

Figure 2. The Dispersal and Reconstruction operations of IDA.

2.2. Adaptive IDA

In most fault-tolerant *redundancy-injecting* communication protocols, redundancy is injected in the form of parity blocks, which are only used for error detection and/or correction purposes [12]. 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) [7]. Using AIDA, a *bandwidth allocation* operation is inserted after the dispersal operation but *prior* to transmission as shown in Figure 3. 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 application 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.

[1] For more details, we refer the reader to the papers by Rabin [23] and Bestavros [6] on IDA implementation.

[2] Another alternative is to broadcast a directory (or index [18]) at the beginning of each broadcast period. This approach is less desirable because it does not lend itself to a clean fault-tolerant organization.

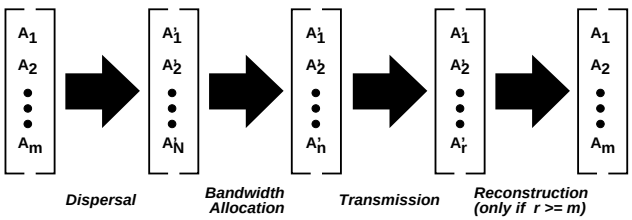

Figure 3. AIDA dispersal and reconstruction

2.3. AIDA-based Broadcast Programs

Figure 4 illustrates a simple example of a flat broadcast program in which two files A and B are transmitted periodically by scanning through their respective blocks. In particular, file A consists of 5 blocks $A_1, \ldots, A_5$ and file B consists of 3 blocks $B_1, \ldots, B_3$. The broadcast period for this broadcast disk is 8 (assuming one unit of time per block). A single error encountered when retrieving a block results in a delay of 8 units of time, until the erroneous block is retransmitted.

Lemma 1 *If the broadcast period of a flat broadcast program is τ, then an upper bound on the worst-case delay incurred when retrieving that file is $r\tau$ units of time, where r is the number of block transmission errors.*

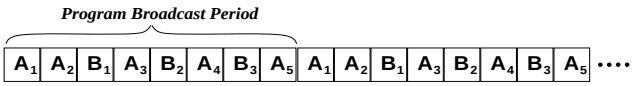

Figure 4. A flat broadcast program

Now, consider the same scenario if files A and B were dispersed using AIDA such that file A is dispersed into 10 blocks, of which *any* 5 blocks are enough to reconstruct it, and file B is dispersed into 6 blocks, of which *any* 3 blocks are enough to reconstruct it. Figure 5 shows a broadcast program in which files A and B are transmitted periodically by scanning through their respective blocks. Notice that there are two "periods" in that transmission. The first is the broadcast period, which (as before) extends for 8 units of time. The length of the broadcast period for a broadcast disk is set so as to accomodate *enough* blocks from every file on that disk—enough to allow clients to reconstruct these files. In the example of Figure 5, at least 5 different blocks and 3 different blocks are needed from files A and B, respectively. While the broadcast period for the broadcast disk is still 8, the server transmits *different* blocks from A and B in subsequent broadcast periods. This leads to the second "period" in the broadcast program, which we call the *program data cycle*. The length of the program data cycle for a broadcast disk is set to accomodate *all* blocks from *all* the dispersed files on that disk. In the example of Figure 5, all 10 blocks and all 6 blocks from dispersed files A and B exist in the program, resulting in a program data cycle of 16.

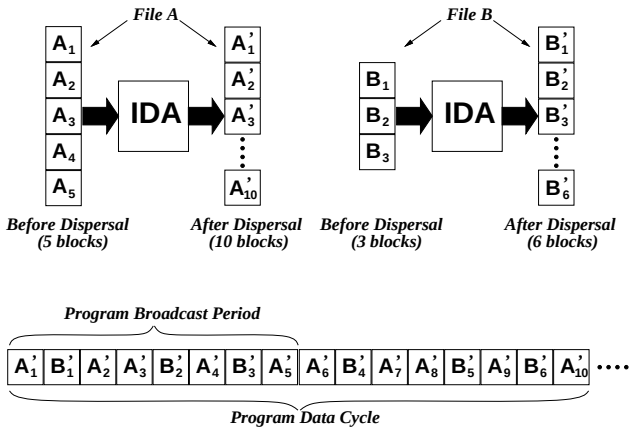

Figure 5. A flat broadcast program using IDA

Unlike the example of Figure 4, a single error encountered when retrieving a block (say from file A) results in a delay of *at most* 2 units of time, until *any* additional block from file A is transmitted. For example, assume that a client received the first 4 blocks, A_1, A_2, A_3, A_4 from file A correctly, but failed to receive the fifth block. In the regime of Figure 4, the client must wait for 8 cycles until A_5 is transmitted again. In the regime of Figure 5, the client has to wait only until A'_6 is transmitted, which implies a delay of only 1 unit of time.

The value of AIDA-based broadcast programs is further appreciated by comparing the delays that a client may experience if errors clobber more than one block during the retrieval of a particular file:

Lemma 2 *If the maximum time between any two blocks of a dispersed file in an AIDA-based flat broadcast program is π, then an upper bound on the worst-case delay incurred when retrieving that file is $r\pi$, where r is the number of block transmission errors.*

From lemmas 1 and 2, an AIDA-based flat broadcast program yields error recovery delays $\frac{\tau}{\pi}$ times shorter than those of a simple flat broadcast program. To maximize the benefit of AIDA-based organization in reducing error recovery delays, the various blocks of a given file should be "uniformly" distributed throughout the broadcast period. In the remainder of this paper, we discuss the use of *pinwheel scheduling* to achieve such

uniform distribution.

3. Pinwheel Task Systems

Pinwheel task systems were introduced by Holte et al. [14], in the context of offline scheduling for satellite-based communication. 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 task may represent a real-time embedded process or a data-transfer operation (between, for example, a sensor and a CPU) that executes continually during the lifetime of the embedded system; the parameter a typically represents the computation requirement of the process, or the amount of data to be transferred, while the parameter b represents a real-time constraint – a deadline – on the execution of the process, or for the completion of the data transfer. Such a pinwheel task is represented by the ordered 3-tuple (i, a, b). A *pinwheel task system* is a set of pinwheel tasks that share a single resource. The following examples illustrate the notion of pinwheel task systems:

Example 1 *The pinwheel task system* $\{(1,1,2), (2,1,3)\}$ *consists of two tasks — one with a computation requirement 1 and window size 2, and the other with computation requirement 1 and window size 3. A schedule for this task system may be represented as follows:* $1,2,1,2,1,2,1,2,\ldots$*, indicating that the shared resource is allocated to tasks* 1 *and* 2 *on alternate slots.*

The pinwheel task system $\{(1,2,5), (2,1,3)\}$ *consists of two tasks — one with a computation requirement 2 and window size 5, and the other with computation requirement 1 and window size 3. A schedule for this task system is:* $1,2,1,O,2,1,2,1,O,2,1,2,1,O,2,\ldots$*, where "O" indicates that the resource remains unallocated during the corresponding time slot.*

The pinwheel task system $\{(1,1,2), (2,1,3), (3,1,n)\}$ *consists of three tasks — one with a computation requirement 1 and window size 2, a second with computation requirement 1 and window size 3, and a third with computation requirement 1 and window size n. It is not difficult to see that this system cannot be scheduled for any finite value of n.*

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, as the third instance in Example 1 shows) that the density of the system be at most one.

The issue of designing efficient scheduling algorithms for pinwheel task systms has been the subject of much research. Holte et al [15] presented an algorithm which schedules any pinwheel task system of two tasks with density at most one. Lin & Lin [21] 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, as the third example pinwheel task system in Example 1 shows, 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 [14] have a simple and elegant algorithm for scheduling any pinwheel task system with density at most one-half. Chan and Chin [11, 10] 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 [10].

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* [14]

$$\{(1, m_1, BT_1), (2, m_2, BT_2), \ldots, (n, m_n, BT_n)\}$$

can be scheduled. Since the algorithm of Chan and Chin [10] can schedule any pinwheel task system with density at most 7/10, a bandwidth

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

is sufficient for this purpose; 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 [10] can be used to determine the actual layout of blocks on the Bdisk.

The fault-tolerance case – when up to r faults must be tolerated – is similarly handled. In this case, the

problem of determining minimum bandwidth reduces to determining the smallest $B \in \mathbf{N}$ such that the pinwheel task system $\{(1, m_1 + r, BT_1), (2, m_2 + r, BT_2), \ldots, (n, m_n + r, BT_n)\}$ can be scheduled. By an analysis similar to the one above, it follows that

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

is sufficient; once again, this represents an efficient solution, with at most a 43% bandwidth overhead for scheduling.

As a further generalization, suppose that each file F_i had a *different* fault-tolerance requirement r_i. There could be several reasons for this: First, some files are more important than others, and therefore less able to tolerate errors. Second, consider a broadcast medium model in which individual transmission errors occur independently of each other, and the occurrence of an error during the transmission of a block renders the entire block unreadable. Thus, larger files (those with greater m_i) will need to tolerate a larger number of faults (larger r_i).

This generalization is easy to solve — as above, we can derive

$$B = \left\lceil \frac{10}{7} \sum_{i=1}^{n} \frac{m_i + r_i}{T_i} \right\rceil \qquad (2)$$

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 several 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.

4.1. 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 studied in Section 3. "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: $d_i^{(0)} = d_i^{(1)} = \ldots = d_i^{(r_i)}$.

In the remainder of this section, we study the design of broadcast programs for generalized fault-tolerant real-time Bdisks—henceforth termed *generalized Bdisks*. As in Section 3, we would like to map the problem to related problems in pinwheel scheduling.

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

$$\mathsf{bc}(i, m_i, \vec{d_i}) \equiv \bigwedge_{j \geq 0} \mathsf{pc}(i, m_i + j, d_i^{(j)}) \ . \qquad (3)$$

Lemma 3 follows as a direct consequence:

Lemma 3 *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*

$$\bigwedge_{i=1}^{n} \left(\bigwedge_{j \geq 0} \mathsf{pc}(i, m_i + j, d_i^{(j)}) \right) \tag{4}$$

■

4.2. Obtaining Broadcast Programs for Generalized Bdisks

Recall that Chan and Chin [10] 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 [10] 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 the constraint (4) of Lemma 3 into a nice form if we are to be able to use the Chan and Chin algorithm. In order to do so, we must be able 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 [10] 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; in the remainder of this section, we present several heuristic rules for obtaining a nice conjunct of pinwheel conditions that implies a given conjunct of pinwheel conditions. 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.

---

a, b, x, y, n are all non-negative integers.

R0 $\mathsf{pc}(i, a - x, b + y) \Leftarrow \mathsf{pc}(i, a, b)$

R1 $\mathsf{pc}(i, na, nb) \Leftarrow \mathsf{pc}(i, a, b)$

R2 $\mathsf{pc}(i, a - x, b - x) \Leftarrow \mathsf{pc}(i, a, b)$

R3 $\mathsf{pc}(i, a, b) \Leftarrow \mathsf{pc}(i, 1, \lfloor b/a \rfloor)$

R4 $\mathsf{pc}(i, a, b) \wedge \mathsf{pc}(i, a + x, b + y) \Leftarrow \mathsf{pc}(i, a, b) \wedge \mathsf{pc}(i', x, b + y) \wedge \mathsf{map}(i', i)$,
where $\mathsf{map}(i', i)$ indicates that tasks i' and i are semantically indistinguishable (i.e., although the scheduler will schedule for the two tasks separately, blocks from file F_i are broadcast whenever either task is scheduled).

R5 $\mathsf{pc}(i, a, b) \wedge \mathsf{pc}(i, na, nb - x) \Leftarrow \mathsf{pc}(i, a, b) \wedge \mathsf{pc}(i', x, nb) \wedge \mathsf{map}(i', i)$

Figure 6. Some pinwheel algebra rules (Proofs may be found in [5])

In Figure 6, we present some rules for manipulating pinwheel conditions. In each, we have some condition on the LHS that is implied by some (hopefully, more useful) condition on the RHS. We may use these rules to obtain some fairly useful generic transformations, which are formally proved in [5]:

Transformation rule 1 (TR1)

$$\mathsf{bc}(i, m_i, \vec{d_i}) \Leftarrow \mathsf{pc}(i, 1, \min_{j \geq 0} \{ \left\lfloor \frac{d_i^{(j)}}{m_i + j} \right\rfloor \})$$

Transformation rule 2 (TR2)

$$\begin{aligned} \mathsf{bc}(i, m_i, \vec{d_i}) \Leftarrow \quad & \mathsf{pc}(i, m_i, d_i^{(0)}) \\ \wedge \quad & \mathsf{pc}(i_1, 1, d_i^{(1)}) \wedge \mathsf{map}(i_1, i) \\ \wedge \quad & \mathsf{pc}(i_2, 1, d_i^{(2)}) \wedge \mathsf{map}(i_2, i) \\ \wedge \quad & \mathsf{pc}(i_3, 1, d_i^{(3)}) \wedge \mathsf{map}(i_3, i) \\ \wedge \quad & \ldots \\ \wedge \quad & \mathsf{pc}(i_{r_i}, 1, d_i^{(r_i)}) \wedge \mathsf{map}(i_{r_i}, i) \end{aligned}$$

where r_i is the dimension of $\vec{d_i}$; i.e., $d_i = [d_i^{(o)}, d_i^{(1)}, \ldots, d_i^{(r_i)}]$.

Observe that $\max_{j \geq 0}\{(m_i + j)/d_i^{(j)}\}$ is a lower bound on the density of any pinwheel condition (or nice conjunct of pinwheel conditions) that may imply $\mathsf{bc}(i, m_i, \vec{d_i})$. (This bound may not be actually achievable — for example, $\mathsf{bc}(i, 2, [5, 7])$ is not implied by any nice conjunct of pinwheel conditions of density $\leq 3/7$). We refer to $\max_{j \geq 0}\{(m_i + j)/d_i^{(j)}\}$ as the *density lower bound* of broadcast file condition $\mathsf{bc}(i, m_i, \vec{d_i})$.

By rule TR1, a broadcast file with a density lower bound in the range $(1/(k+1), 1/k]$ gets transformed to a pinwheel condition with density $1/k$. In general, for broadcast files with a *low* density lower bound, this is an adequate transformation (Examples 2 and 3 below); for broadcast files with higher density lower bounds, however, rule TR2, along with a certain amount of manipulation using R0–R5, may yield significant savings in density.

In general, then, the strategy should be as follows. Given the specifications of a set of broadcast files,

1. Use rule TR1 to determine a candidate transformation.
2. - Use Lemma 3 to obtain equivalent pinwheel conditions, not necessarily in nice form.
   - Use the rules R0 – R3 and R5 to simplify, if possible.
   - Use rule R4 on the simplified pinwheel conditions to obtain another candidate transformation.

Choose the candidate transformation from among the two above with the smaller density.

We conclude this section with some examples illustrating how these transformation rules may be used to obtain nice pinwheel conjuncts that imply a given broadcast file specification.

Example 2 *F_i has $m_i = 5$, and $\vec{d_i} = [100, 105, 110, 115, 120]$. This is represented by $\mathsf{bc}(i, 5, [100, 105, 110, 115, 120])$, and has a density lower bound of $\max\{0.05, 0.0571, 0.0636, 0.0696, 0.075\} = 0.075$. By Rule TR1, this broadcast file condition is implied by $\mathsf{pc}(i, 1, 13)$, which has a density of 0.0769 — within 2.5% of the density lower bound.*

Example 3 *F_i has $m_i = 6$, and $\vec{d_i} = [105, 110]$. This is represented by $\mathsf{bc}(i, 6, [105, 110])$, and has a density lower bound of 0.0636. By Rule TR1, this is implied by $\mathsf{pc}(i, 1, 15)$, which has a density of 0.06667. By TR2, it is implied by $\mathsf{pc}(i, 6, 105) \wedge \mathsf{pc}(i', 1, 110) \wedge \mathsf{map}(i', i)$, in which case the density is $6/105 + 1/110 = 0.0662$. Hence, the latter transformation is selected, and the actual density of the nice conjunct is within 4.1% of the lower bound.*

■

The examples below illustrate how rules R0–R3 and R5 may be sometimes used to simplify the conjunct of pinwheel conditions obtained by the application of transformation rules TR1 and TR2.

Example 4 *F_i has $m_i = 4$, and $\vec{d_i} = [8, 9]$. This is represented by $\mathsf{bc}(i, 4, [8, 9])$, and has a density lower bound of 0.5556. By Rule TR1, this is implied by $\mathsf{pc}(i, 1, 1)$, which has a density of 1.0. By TR2, it is implied by $\mathsf{pc}(i, 4, 8) \wedge \mathsf{pc}(i', 1, 9) \wedge \mathsf{map}(i', i)$, in which case the density is $4/8 + 1/9 = 0.6111$.*

By observing that $\mathsf{pc}(i, 4, 8) \Leftarrow \mathsf{pc}(i, 1, 2)$ (by R1), and applying R5 to conclude that $\mathsf{pc}(i, 1, 2) \wedge \mathsf{pc}(i, 5, 9) \Leftarrow \mathsf{pc}(i, 1, 2) \wedge \mathsf{pc}(i', 1, 10) \wedge \mathsf{map}(i', i)$, we obtain a transformation that is even "better" — one with density equal to $1/2 + 1/10 = 0.6000$; i.e., within 4% of the lower bound.

Example 5 *When $d_i^{(j)} = d_i^{(j+1)}$, rule R0 may be used to rid of one conjunct. Thus, $\mathsf{bc}(i, 2, [5, , 6]) \equiv \mathsf{pc}(i, 2, 5) \wedge \mathsf{pc}(i, 3, 6) \wedge \mathsf{pc}(i, 4, 6)$, which simplifies to $\mathsf{pc}(i, 2, 5) \wedge \mathsf{pc}(i, 4, 6)$. By R1, $\mathsf{pc}(i, 2, 3) \Rightarrow \mathsf{pc}(i, 4, 6)$; by R0, $\mathsf{pc}(i, 2, 3) \Rightarrow \mathsf{pc}(i, 2, 5)$. Therefore, $\mathsf{bc}(i, 2, [5, 6, 6]) \Leftarrow \mathsf{pc}(i, 2, 3)$. Observe that this is an optimal transformation, in that the density of this nice pinwheel condition is equal to the density lower bound of the broadcast condition.*

Example 6 *$\mathsf{bc}(i, 1, [2, 3]) \equiv \mathsf{pc}(i, 1, 2) \wedge \mathsf{pc}(i, 2, 3)$. By R2, $\mathsf{pc}(i, 2, 3) \Rightarrow \mathsf{pc}(i, 1, 2)$; therefore, $\mathsf{pc}(i, 2, 3)$, with a density of 0.6667, is an equivalent nice pinwheel condition.*

Applying TR2 directly to $\mathsf{bc}(i, 1, [2, 3])$ would yield the nice conjunct of pinwheel conditions $\mathsf{pc}(i, 1, 2) \wedge \mathsf{pc}(i', 2, 3) \wedge \mathsf{map}(i, i')$, which has a density of $1/2 + 1/3 = 0.8333$.

5. Summary

Previous work on broadcast disks did not deal explicitly with the fault-tolerance and timeliness constraints that may be imposed on the broadcasted data. In this paper, 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, and have proven some new results in pinwheel scheduling theory. These results enable us to design efficient algorithms for organizing data on broadcast disks.

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. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6), December 1995.

[3] 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.

[4] 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.

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

[6] A. 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.

[7] A. Bestavros. An adaptive information dispersal algorithm for time-critical reliable communication. In I. Frisch, M. Malek, and S. Panwar, editors, *Network Management and Control, Volume II*. Plenum Publishing Corporation, New York, New York, 1994.

[8] A. 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.

[9] A. Bestavros and S. Nagy. Value-cognizant admission control for rtdbs. In *Proceedings of RTSS'96: The 16th IEEE Real-Time System Symposium*, Washington, DC, December 1996.

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

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

[12] G. Gibson, L. Hellerstein, R. Karp, R. Katz, and D. Patterson. Coding techniques for handling failures in large disk arrays. Technical Report UCB/CSD 88/477, Computer Science Division, University of California, July 1988.

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

[14] 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.

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

[16] J. Huang and L. 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.

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

[18] T. Imielinski, S. Viswanathan, and B. Badrinath. Energy efficient indexing on air. In *Proceedings of ACM SIGMOD Conference*, Minneapolis, MN, May 1994.

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

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

[21] 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.

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

[23] M. 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.

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

[25] W.-K. Shih, J. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM journal of Computing*, July 1991.

[26] V. F. Wolfe, L. C. 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.

[27] 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.