AIDA-based Real-Time Fault-Tolerant Broadcast Disks[*]

AZER BESTAVROS
(best@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215

Abstract

The proliferation of mobile computers and wireless networks requires the design of future distributed real-time applications to recognize and deal with the significant asymmetry between *downstream* and *upstream* communication capacities, and the significant disparity between server and client storage capacities. Recent research work proposed the use of *Broadcast Disks* as a scalable mechanism to deal with this problem. In this paper, we propose a new broadcast disks protocol, based on our Adaptive Information Dispersal Algorithm (AIDA). Our protocol is different from previous broadcast disks protocols in that it improves communication timeliness, fault-tolerance, and security, while allowing for a finer control of multiplexing of prioritized data (broadcast frequencies). We start with a general introduction of broadcast disks. Next, we propose broadcast disk organizations that are suitable for real-time applications. Next, we present AIDA and show its fault-tolerance and security properties. We conclude the paper with the description and analysis of AIDA-based broadcast disks organizations that achieve *both* timeliness and fault-tolerance, while preserving downstream communication capacity.

Keywords: Broadcast disks; real-time, fault-tolerant, and secure communication; mobile computing; information dispersal and retrieval.

[*] This work has been partially supported by NSF (grant CCR-9308344).

1 Introduction

Mobile computers are likely to play an important role at the extremities of future large-scale distributed real-time systems. Examples include automotive on-board navigational systems, 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 to clients and the *upstream* communication capacity from clients to servers. This discrepancy may result from the huge disparity between the transmission capabilities of clients and servers (*e.g.* broadcasting via satellite from servers to clients as opposed to cellular modem communication from clients to servers), or may be due simply to the scale of information flow (*e.g.* thousands of clients may be connecting to a single computer for service).

The notion of *Broadcast Disks* was introduced by Zdonik *et al.* in [21] as a mechanism that uses communication bandwidth to emulate a storage device (or a memory hierarchy in general). 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 broadcast disks is different from previous work in both wired and wireless networks [12, 14] in that several sources of data are multiplexed and broadcast to clients, thus creating a hierarchy of broadcast disks with different sizes and speeds. On the server side, this hierarchy gives rise to memory management issues (*e.g.* allocation of data to broadcast disks 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 [3], Acharya, Franklin and Zdonik discuss broadcast disks organization issues, including client cache management [1], and client-initiated prefetching to improve the communication latency for database access systems [2].

Previous work in broadcast disks technology was driven by 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/security constraints are imposed on the system were not considered.

Current broadcast disks protocols assume that the rate at which a data item (say a page) is broadcast is dependent on the *demand* for that data item. Thus, *hot* data items would be placed on fast-spinning disks (*i.e.* broadcast at a higher rate), whereas *cold* data items would be placed on slow-spinning disks (*i.e.* broadcast at a lower rate). Such a strategy is optimal in the sense that it minimizes the average latency amongst all clients over all data items.[1] In a real-time environment, minimizing the average latency seizes to be the main performance criterion. Rather, guaranteeing (either deterministically or probabilistically) that timing constraints will be met becomes the overriding concern. In this paper, we show how to allocate data items to broadcast disks so as to ensure the satisfaction of timing constraints imposed on client tasks that may rely on this data.

Current broadcast disks 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 broadcast disk 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. In this paper, we show how to allocate data items to broadcast disks so as to mask (or otherwise minimize) the impact of intermittent failures. In that respect, we use the Adaptive Information Dispersal Algorithm (AIDA) [6], which allows for a controllable and efficient tradeoff of bandwidth for reliability.

This paper is organized as follows. In section 2, we present three different broadcast disks organizations—namely flat, rate monotonic, and slotted rate monotonic—that are suitable for use with real-time applications. In section 3, we present our AIDA-based approach to improving the fault-tolerance and security of broadcast disks. In particular, we introduce the basics of AIDA and demonstrate its efficient use of redundancy to tolerate failures. In section 4, we show how *both* timeliness and fault-tolerance properties could be guaranteed through the use AIDA to mask failures in flat, rate monotonic, and slotted rate monotonic organizations. We conclude our paper with a summary and a discussion of future work.

[1] Current research work in broadcast disks technology has singled out the *expected latency* (*i.e.* how long does a client have to wait on the average to retrieve a data item) as the main performance measure to be tuned. In a real-time environment, the *worst case latency* is a more appropriate measure that will be used throughout this paper.

2 Organization of Broadcast Disks to Ensure Timeliness

We model a broadcast disks system as comprising of a set of data items (or files) that must be transmitted continuously and periodically to the client population. Let B_d denote the downstream bandwidth (number of bytes per second) available from the server to its client population. Let S_i denote the size of (number of bytes in) data item D_i, where $i = 1, \ldots k$. We assume that each data item comprises a number of fixed-size blocks (or pages). 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. Let T_i denote the tightest time constraint associated with the retrieval of data item D_i. In effect, T_i establishes an upper bound on the tolerable latency for data item D_i.

2.1 Flat Organization of Broadcast Disks

The simplest transmission method in a broadcast-based system is for the server to send the union of all data items that may be needed by its clients in a periodic fashion. This regimen has been termed as the *flat broadcast program* [3]. With a flat broadcast, the worst-case latency is the same for all data and is equal to the broadcast period. This leads to the following lemma about downstream bandwidth requirement.

Lemma 1 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for the retrieval of D_i, where $i = 1, \ldots k$, then the downstream bandwidth B_d for a flat broadcast is bounded by the following inequality.*

$$B_d \geq \frac{\sum_{i=1}^{k} S_i}{\min_{i=1}^{k}(T_i)} \tag{1}$$

Notice that in a flat organization of broadcast disks, the broadcast period τ_{flat} is constrained by two quantities. In particular, it has an upper bound equal to the tightest timing constraint over all data items, and a lower bound that depends on the available downstream bandwidth. These bounds are given below.

$$\min_{i=1}^{k}(T_i) \geq \tau_{\text{flat}} \geq \frac{\sum_{i=1}^{k} S_i}{B_d} \tag{2}$$

If inequality 1 is satisfied, then the server can choose any broadcast period as long as it satisfies inequality 2. Setting τ_{flat} to its minimum possible value results in minimizing the expected latency of retrieval, whereas setting τ_{flat} to its maximum possible value leaves the server with extra bandwidth that could be used to broadcast data items that are not associated with any time constraints (*e.g.* clock synchronization messages, or other control information).

The flat organization of broadcast disks makes the process of adding new data items to the broadcast program very simple. In particular, to add a new data item D_{new}, the server must first establish whether or not there is enough downstream bandwidth to accomodate D_{new}. This is done by using inequality 1. If successful, D_{new} is simply added to the broadcast program and the broadcast period is adjusted according to inequality 2, if need be.

2.2 Rate Monotonic Organization of Broadcast Disks

The flat organization of broadcast disks is wasteful of bandwidth because it does not take into consideration the timing constraints imposed on the various data items. To achieve the most utilization of the available downstream bandwidth, the worst-case latency for each data item must be equal to the timing constraint associated with that data item. In other words, data item i must be broadcast periodically at a rate equal to $\frac{1}{T_i}$. This leads to the following lemma about downstream bandwidth requirement.

Lemma 2 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for the retrieval of D_i, where $i = 1, \ldots k$, then the downstream bandwidth B_d for a rate monotonic broadcast is bounded by the following inequality.*

$$B_d \geq \sum_{i=1}^{k} \frac{S_i}{T_i} \tag{3}$$

Furthermore, the above inequality establishes a lower bound on the downstream bandwidth B_d for any broadcast program that satisfies the timing constraints of all data items.

In a rate monotonic organization of broadcast disks, each data item represents an independent disk *"spinning"* at a rate reciprocal to the timing constraint associated with it. To obtain a broadcast program, the server must determine the length of that program, which is set to be the least common multiple of all timing constraints, $T_{\text{lcm}} = \text{LCM}_{i=1}^{k}(T_i)$. Next, the data items are sorted in an ascending order based on their timing constraints. Next, the broadcast program is constructed by stepping through the sorted list of data items, and allocating time slots for each block in D_i within each T_i minor cycle of T_{lcm}.[2]

[2] This process is similar to CPU Rate Monotonic Scheduling algorithms [16].

Despite the fact that it achieves an optimal utilization of downstream bandwidth, the rate monotonic organization of broadcast disks is not practical in a large system with thousands of data items, each with a different timing constraint. In such systems, broadcasting each data item at a rate reciprocal to its timing constraint would result in a very complicated broadcast program. More importantly, the process of adding new data items or changing the timing constraints of existing data items may require a complete overhaul of the broadcast program, thus rendering dynamic reprogramming an impossibility.

2.3 Slotted Rate Monotonic Organization of Broadcast Disks

Flat and rate monotonic organizations of broadcast disks underline two extreme trade-offs. Flat organization trades off bandwidth utilization for ease of broadcast programming, whereas rate monotonic organization trades off ease of broadcast programming for bandwidth utilization. The slotted rate monotonic organization of broadcast disks strikes a balance between these two extremes. The basic idea is to coalesce data items together so that they could be used as part of the same broadcast disk. The effect of such coalescence is to create a partition on the data items to be broadcast to the client population. Data items in the same partition share the same broadcast disk and, thus, have identical worst-case latencies.

Let $C_u, u = 1, \ldots, q$ denote the broadcast disks to be used, and let τ_u, denote the broadcast period of disk C_u. According to the slotted rate monotonic organization of broadcast disks, a data item D_i with a time constraint T_i is assigned to the broadcast disk with the largest broadcast period smaller than T_i. In other words, D_i is assigned a broadcast disk C_v if and only if $T_i \geq \tau_v$ and $(T_i - \tau_v) < (T_i - \tau_u)$, where $u \neq v$. This leads to the following lemma about downstream bandwidth requirement.

Lemma 3 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for the retrieval of D_i, where $i = 1, \ldots k$, then the downstream bandwidth B_d for a slotted rate monotonic broadcast consisting of disks $C_u, u = 1, \ldots, q$ with broadcast periods τ_u is bounded by the following inequalities.*

$$B_d \geq \sum_{u=1}^{q} \frac{\sum_{\forall D_i \in C_u} S_i}{\tau_u} \tag{4}$$

$$\forall D_i \in C_u : T_i \geq \tau_u \tag{5}$$

Notice that in a slotted rate monotonic organization of broadcast disks, each broadcast period τ_v is constrained by two quantities. In particular, it has an upper bound equal to the tightest timing constraint over all data items in C_v, and a lower bound that depends on the available downstream bandwidth. These bounds are shown below.

$$\min_{\forall D_i \in C_v} (T_i) \geq \ \tau_v \quad \geq \quad \frac{\sum_{\forall D_i \in C_v} (S_i)}{B_d - \sum_{\forall u, D_i \in C_u, u \neq v} (\frac{S_i}{\tau_u})} \tag{6}$$

If inequality 4 is satisfied, then the server can choose any broadcast periods as long as they satisfy inequality 6. For a broadcast disk C_v, setting τ_v to its minimum possible value results in minimizing the expected latency of retrieval from that disk, whereas setting τ_v to its maximum possible value leaves the server with extra bandwidth that could be used to broadcast control information, or alternately to be used to broadcast data items added to that disk at a later time.

The slotted rate monotonic organization of broadcast disks makes the process of adding new data items to the broadcast programs much simpler than that of rate monotonic organization. In particular, to add a new data item D_{new}, the server must first establish whether or not there is enough downstream bandwidth to accomodate D_{new}. This is done by using inequality 4. If successful, D_{new} is simply added to the broadcast program of the appropriate broadcast disk and the broadcast periods are adjusted, if need be, to satisfy inequality 6.

In the above discussion, we have assumed that the number of broadcast disks q as well as the broadcast periods of these disks τ_u, for $u = 1, \ldots, q$ are known quantities. In a typical system, this is not likely to be the case. In particular, these quantities must be computed by the server given the set of data items to be broadcasted and their timing constraints. This can be achieved through an interative process, whereby the server starts with an initial assignment of one data item per broadcast disk.[3] In each iteration, the server proceeds by coalescing two broadcast disks by decreasing the broadcast period of the slower disk to match the faster one. The two broadcast disks to be coalesced are chosen in a way that minimizes the bandwidth wasted by this “speeding-up” process, while satisfying inequality 6. The process is completed when no coalescing is possible—namely when it is not possible to coalesce any two disks without violating inequality 6.[4]

[3] This is always possible as long as inequality 3 holds, which is a necessity as established in lemma 2.

[4] We are currently investigating other algorithms for identifying the “best” set of broadcast disks for a slotted rate monotonic organization under various assumptions.

3 Organization of Broadcast Disks to Ensure Fault-tolerance

Current techniques for organizing broadcast disks do not accomodate for transmission failures. In particular, when an error[5] occurs in the retrieval of one (or more) blocks from a data item (or file), 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 transmit before getting back to the block in question. For real-time systems, such a delay may result in missing critical timing constraints. In this section we show how to use AIDA to add fault-tolerance properties to broadcast disks.

AIDA is a novel 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 Michael O. Rabin [19], which we have previously shown to be a sound mechanism that considerably improves the performance of I/O systems and parallel/distributed storage devices [4, 9]. The use of IDA for efficient routing in parallel architectures has also been investigated [18].

3.1 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 messages (or 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 and distributing it over N sites 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 [5], where we presented an architecture and a CMOS implementation of a VLSI chip[6] that implements IDA.

[5] In this paper we assume that all transmission failures are manifested as communication errors that are detected by clients (*e.g.* through the generation of a checksum/parity error).

[6] The chip (called SETH) has been fabricated by MOSIS and tested in the VLSI lab of Harvard University, Cambridge, MA. The performance of the chip was measured to be about 1 megabyte per second. By using proper pipelining and more elaborate designs, this figure can be boosted significantly.

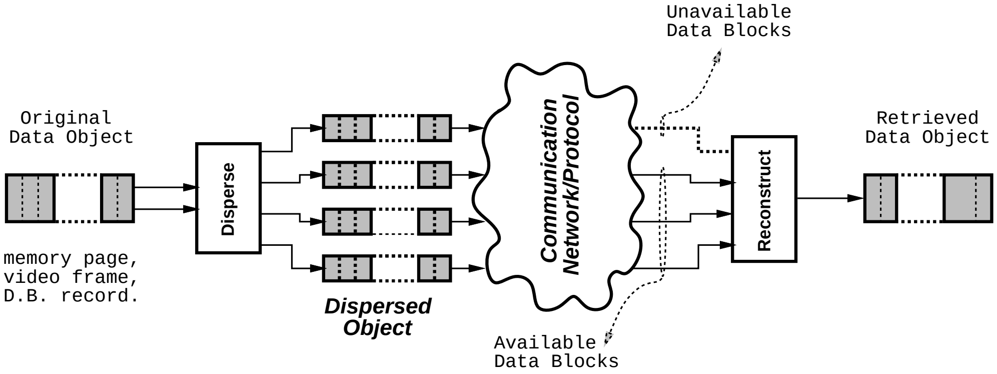


Figure 1: Dispersal and reconstruction of information using IDA.

The dispersal and reconstruction operations are simple linear transformations using *irreducible polynomial arithmetic.*[7] 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 inversible. 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.[8] 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.

[7] For more details, we refer the reader to Rabin's paper [19] on IDA and our paper on IDA implementation [5].

[8] Another alternative is to broadcast a directory (or index [15]) at the beginning of each broadcast period. This approach is less desirable because it does not lend itself to a clean fault-tolerant organization.

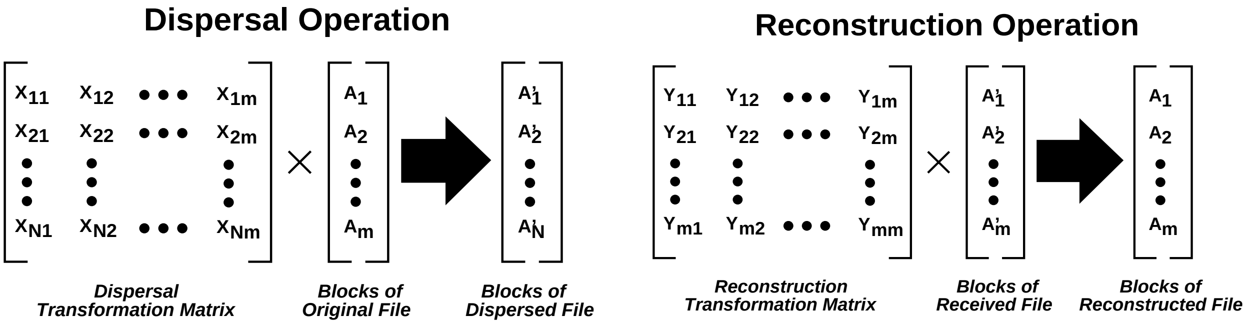


Figure 2: The Dispersal and Reconstruction operations of IDA.

3.2 Adaptive Information Dispersal and Retrieval

Several fault-tolerant *redundancy-injecting* protocols (similar to IDA) 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 [11]. The IDA approach is radically 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) [6].

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, while 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 profiles of reliability and accessibility requirements for the various objects (files) in the system by controlling the levels of distribution and dispersal for these objects. 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.

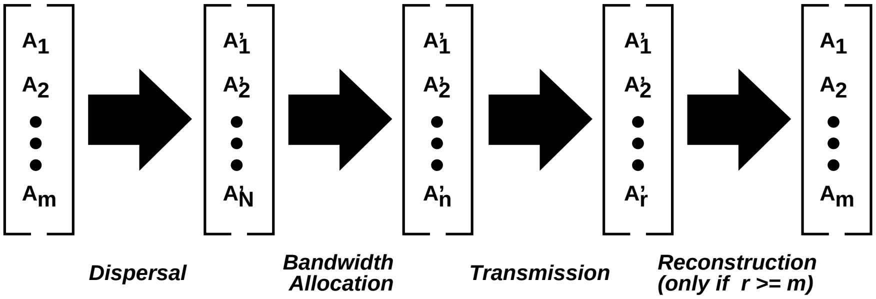


Figure 3: Dispersal and reconstruction of information using AIDA.

3.3 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. This leads to the following eay-to-prove lemma.

Lemma 4 *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.*

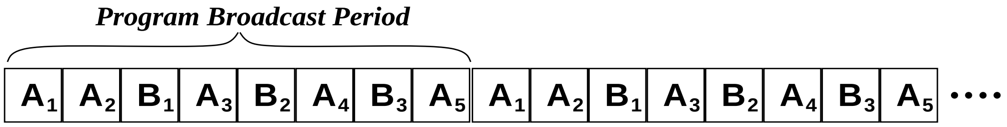
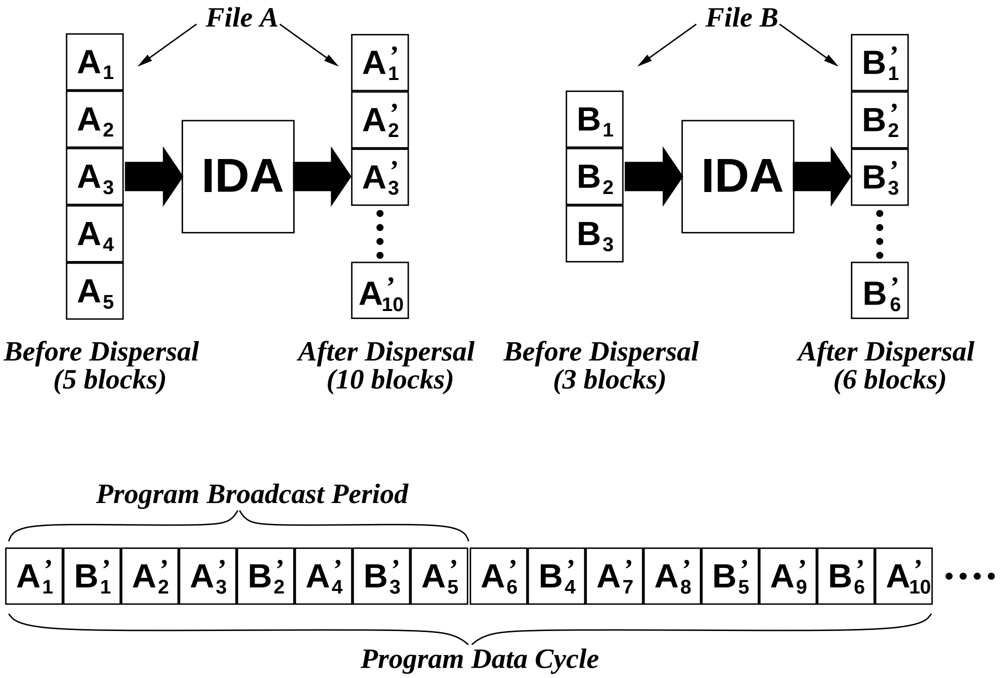


Figure 4: Example of a flat broadcast program.

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.



File A
A1
A2
A3
A4
A5
IDA
A'1
A'2
A'3
A'10
Before Dispersal (5 blocks)
After Dispersal (10 blocks)
File B
B1
B2
B3
IDA
B'1
B'2
B'3
B'6
Before Dispersal (3 blocks)
After Dispersal (6 blocks)
Program Broadcast Period
A'1 B'1 A'2 A'3 B'2 A'4 B'3 A'5 A'6 B'4 A'7 A'8 B'5 A'9 B'6 A'10 ····
Program Data Cycle

Figure 5: Example of 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. Using the broadcast programs of figures 4 and 5, one can easily establish estimates for the worst-case delays as a function of the number of transmission failures. These are shown in figure 6. This observation could be easily generalized to yield the following lemma.

Number of Errors	Worst-Case Delay	
	With IDA	Without IDA
0	0	0
1	3	8
2	4	16
3	6	24
4	7	32
5	8	40

Figure 6: Worst-case delays as a function of the number of block transmission errors

Lemma 5 *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 $r\pi$ units of time, where r is the number of block transmission errors.*

The comparison in figure 6 is for a toy example, with only two files in a broadcast period. In a typical broadcast disk, the difference between the two regimes is much more accentuated. From lemmas 4 and 5, 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. For example, if the broadcast program consists of 200 blocks from 10 different files, each consisting of 20 blocks, then it is possible to spread the blocks in such a way that blocks from the same file are located at most $\pi = \frac{200}{20} = 10$ blocks away from each other. This results in a 20-fold speedup in error recovery.

Generally speaking, the value of $\frac{\tau}{\pi}$ depends on many parameters, including the granurality of the blocks, the length of the broadcast program, and the relative sizes of the files included in the broadcast program.

3.4 Security and Privacy Properties

The broadcasting of data in a distributed system raises significant security and privacy issues. In particular, data intended to a particular client community (*e.g.* subscribers to a particular stock market service) may be accidentally or deliberately read and interpreted by unauthorized clients.

A common technique to ensure communication security is to store and communicate information using some form of encryption, where only authorized clients are enabled to decrypt the information through the use of appropriate *secret keys* [20]. The proven difficulty of decrypting the information without knowing the secret key guarantees a high level of *security.* The main disadvantage of this technique is that the information (although encrypted) is available as a whole to unauthorized clients. This may make it possible (albeit hard) for adversaries to break the secret encryption key. The AIDA-based protocol we are proposing in this paper adds another level of protection against unauthorized access to broadcast data. This level of protection could be use *in addition to*, or *in lieu of* encryption.

The AIDA dispersal and reconstruction operations require the server and clients to agree on the $[x_{ij}]_{N\times m}$ transformation matrix. Such an agreement could be accomplished *a priori* by coding these transformations into the communication protocol. Alternately, these transformations could be dynamically broadcast to clients. This could be useful in enhancing security, by changing periodically the $[x_{ij}]_{N\times m}$ transformation matrix, thus making unauthorized clients incapable of "listening in" on the broadcasted information. The security properties of AIDA are attractive because they do not add any complexity to the communication process. In other words, communication privacy is boosted simply by using (and dynamically changing) the secret IDA transformation matrices, at no additional cost.

4 Adding Timeliness to Fault-tolerant Broadcast Disks

In the previous sections, we discussed techniques that could be used to improve the timeliness and fault-tolerance properties of broadcast disk systems. In this section, we discuss how to compose these properties.

4.1 Shadowed Broadcast Disk Organization

Using a shadowed broadcast disk organization, each file on a broadcast disk must be transmitted $r+1$ times within each broadcast period to completely mask the effect of up to r block transmission errors. To explain how this could be achieved, consider the broadcast program shown in figure 7. Files A and B are replicated so that they are transmitted twice in each broadcast period. If an error is encountered in retrieving any single block from a file—say A—then that error will not result in missing any time constraints associated with the retrieval of file A at the client because that same block will be retransmitted again within the same broadcast period. The cost of masking such errors is to reduce the downstream bandwidth utilization. This result is established in the following lemma.

Lemma 6 *Using a shadowed organization of broadcast disks, it is possible to tolerate any r failures per broadcast period by sacrificing $\frac{r}{r+1}$ of the bandwidth available to the broadcast disk.*

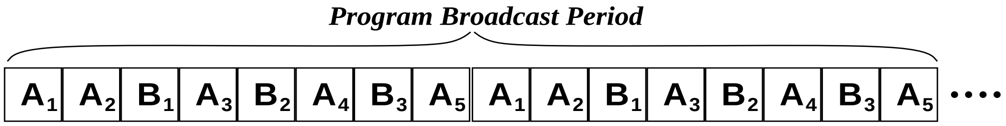

Program Broadcast Period
A_1 A_2 B_1 A_3 B_2 A_4 B_3 A_5 A_1 A_2 B_1 A_3 B_2 A_4 B_3 A_5 ....

Figure 7: Example of a shadowed broadcast program.

4.2 AIDA-based Broadcast Programs

AIDA could be used not only to reduce the impact (recovery delay) of a transmission error, but also to completely mask the impact of such error by sacrificing a minimal percentage of the downstream bandwidth. In other words, AIDA could be used to ensure that for a preset maximum number of failures, a transmission error will not result in delays that would jeopardize the timely retrieval of data from a broadcast disk.

To explain how this could be achieved, consider the broadcast program shown in figure 8. File A is dispersed using AIDA in such a way that any 5 of its blocks would be enough to reconstruct it. Yet, in the broadcast program, 6 blocks are used within a single broadcast period. Similarly, file B is dispersed using AIDA in such a way that any 3 of its blocks would be enough to reconstruct it. Yet, in the broadcast program, 4 blocks are used within a single broadcast period. The result of packing an "extra" block from files A and B within the broadcast period is to make clients able to completely mask the impact of a single block transmission error (per file per broadcast period). In other words, if an error is encountered in retrieving any single block from a file—say A—then that error will not result in missing any time constraint associated with the retrieval of file A at the client. The cost of masking such errors is to reduce the downstream bandwidth utilization. This result is established in the following lemma.

Lemma 7 *Using an AIDA-based organization of broadcast disks, it is possible to tolerate any r failures per broadcast period by sacrificing a fraction*

$$\frac{r}{r + \frac{1}{k}\sum_{i=1}^{k}(m_i)}$$

of the bandwidth available to the broadcast disk, where m_i is the minimum number of blocks necessary to reconstruct data item D_i, and k is the total number of data items in the broadcast program.

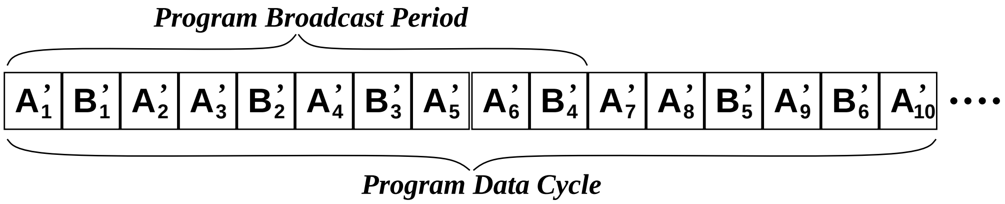


Figure 8: Example of an AIDA-based broadcast program.

In the example of figure 8, masking a single block transmission error costs 20% of the broadcast disk's bandwidth. This is to be compared with 50%, if the same effect is to be achieved using a simple shadowed broadcast disk organization. The value of AIDA-based broadcast programs in masking transmission errors is further appreciated by comparing the bandwidth savings achievable through AIDA when more than a single failure is to be tolerated. This could be established by computing the ratio of the extra bandwidth needed when AIDA is used and the the extra bandwidth needed when shadowing is used. Assuming that broadcast disk consists of k data items of sizes $S_i, i = 1, \ldots, k$, and assuming that the size of a block is b, then from lemmas 6 and 7, this ratio is as follows.

$$\frac{\text{AIDA Extra bandwidth}}{\text{Shadowing Extra Bandwidth}} = \frac{kb}{\sum_{i=1}^{k}(S_i)} \tag{7}$$

For example, if the broadcast program consists of $k = 20$ files, each consisting of 10 blocks, then using AIDA results in a 10-fold decrease in the "extra" bandwidth necessary to mask the effect of any number of failures. Notice that this result depends on the block size used. In particular, if the block size is halved, making each one of the 20 files occupy 20 blocks, then using AIDA results in a 20-fold decrease in "extra" bandwidth. Generally speaking, smaller block sizes result in larger bandwidth savings when AIDA is used. This, however, is not without a cost. Namely, smaller block sizes result in a larger number of blocks, and hence a more complex (costly) reconstruction process. Therefore the savings in bandwidth must be weighed against the computation cost of the reconstruction process.

Using the results of lemma 7, we are now ready to reformulate lemmas 1, 2, and 3 to obtain the minimum downstream bandwidth requirements to be able to satisfy *both* the real-time and fault-tolerance constraints imposed on flat, rate monotonic, and slotted rate monotonic broadcast disks, respectively.

Lemma 8 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for its retrieval in the presence of at most r block transmission errors, where $i = 1, \ldots k$, then the downstream bandwidth B_d for an AIDA-based flat broadcast with a block size b is bounded by the inequality.*

$$B_d \quad \geq \quad \frac{\sum_{i=1}^{k}(S_i + rb)}{\min_{i=1}^{k}(T_i)} \tag{8}$$

Lemma 9 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for the retrieval of D_i in the presence of at most r block transmission errors, where $i = 1, \ldots k$, then the downstream bandwidth B_d for an AIDA-based rate monotonic broadcast with a block size b is bounded by the inequality.*

$$B_d \quad \geq \quad \sum_{i=1}^{k} \frac{(S_i + rb)}{T_i} \tag{9}$$

Lemma 10 *If S_i is the size of data item D_i and T_i is the worst-case latency tolerable for the its retrieval in the presence of at most r block transmission errors, where $i = 1, \ldots k$, then the downstream bandwidth B_d for an AIDA-based slotted rate monotonic broadcast consisting of disks $C_u, u = 1, \ldots, q$ with broadcast periods τ_u and a block size b is bounded by the inequalities.*

$$B_d \quad \geq \quad \sum_{u=1}^{q} \frac{\sum_{\forall D_i \in C_u}(S_i + rb)}{\tau_u} \tag{10}$$

$$\forall D_i \in C_u : T_i \quad \geq \quad \tau_u \tag{11}$$

5 Conclusion and Future Work

With the advent of mobile computers and cellular communication, it is expected that most clients in a distributed environment 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. Example applications include intelligent navigational systems, wearable battlefield computers, and computerized interactive TV cable boxes. 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 paper, we have proposed a number of broadcast disk organizations that would alleviate such constraints. In particular, we have presented and evaluated a novel real-time, fault-tolerant, secure broadcast disk organization technique based on the Adaptive Information Dispersal Algorithm (AIDA).

AIDA does not *guarantee* that time-constraints will be satisfied, rather it guarantees a (possibly dynamic) lower bound on the *probability* of meeting these constraints. This probability can be made *arbitrarily* high if enough bandwidth is sacrificed. This, however, may not be feasible if the system is running close to capacity. One possible approach to deal with such a situation is to allow the *quality* of service to degrade gracefully. The integration of AIDA with *best-effort* techniques such as those presented in [10, 13] and *imprecise computation* techniques such as those suggested in [17] is an interesting research problem yet to be pursued.

The question of propagating updates to data stored on broadcast disks is an interesting topic that received no attention from the research community. Current work ignores this problem under the assumption that modified data will eventually be rebroadcasted, thus invalidating client caches, and possibly restarting transactions or computations carried on the old *stale* data. Delays from such restarts could be fatal in a real-time environment. Techniques that allow for very frequent broadcasting of invalidation messages (or incremental updates) coupled with speculative client processing policies [8] could be useful in alleviating this problem. Other related issues include real-time database concurrency control and indexing.

Broadcast disks are likely to be used by clients in retrieving information from a large body of data. Despite the abundance of downstream bandwidth, it is likely that this bandwidth is not going to be "enough" to broadcast all the information that clients may ever need. One approach to deal with this problem is for clients to dispatch agents by sending appropriate control messages through the limited upstream bandwidth. These agents gather the needed information by searching the Global Infosphere (say the WWW) and by contacting servers to transmit that information down to clients. In a real-time setting, there are a number of issues to be considered. For example, agents must be designed to meet timing constraints established by clients. This could be done through the use of (say) imprecise computation techniques. Also, admission control and scheduling strategies must be incorporated in broadcast disks protocols to ensure that agents are able to transmit results to clients in due time. Other related issues include client-initiated caching and prefetching strategies.

The selection of *"what data to locate"* on broadcast disks and *"how frequently to broadcast it"* are interesting problems, reminiscent of the specualtive data dissemination protocols [7] we proposed and evaluated for the WWW. The use of broadcast disks, however, poses new challenges for the implementation of these protocols. In particular, when broadcast disks are used, servers cannot keep track of the access patterns necessary for data dissemination. Therefore, new protocols must be devised to allow servers to reconstruct these access patterns.

Broadcast disks offer an attractive mechanism for *"linking"* mobile clients to the Global Infosphere. Nevertheless, broadcast disks introduce problems of their own. One such problem is security. If data is broadcast to a client, then it is available to all. More importantly, if clients are to rely on agents they dispatch, then mechanisms must be devised to authenticate messages received from such agents. In this paper we investigated the potential use of information dispersal to boost the security/privacy of broadcast disks through the use of secret dispersal keys. More work needs to be done to embed encryption/authentication protocols in broadcast disks protocols.

In order to evaluate real-time fault-tolerant broadcast disks protocols, it is necessary to develop a testbed and a set of benchmarks. To that end, our current effort involves the development and use of such utilities to establish the correctness and compare the performance of various broadcast disks protocols.

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. Prefetching from a broadcast disk. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

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

[4] Azer Bestavros. IDA-based disk arrays. Technical Memorandum 45312-890707-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, July 1989.

[5] 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.

[6] 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.

[7] Azer Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

[8] Azer Bestavros and Spyridon Braoudakis. Value-cognizant speculative concurrency control for real-time databases. *Information Systems: Special Issue on Real-Time Databases*, (To Appear in 1996).

[9] Azer Bestavros, Danny Chen, and Wing Wong. The reliability and performance of parallel disks. Technical Memorandum 45312-891206-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, December 1989.

[10] D. Ferrari. Design and application of a delay jitter control scheme for packet-switching internetworks. In *Proceedings of the second International Conference on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[11] 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.

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

[13] M. Gilge and R. Gussella. Motion video coding for packet-switching networks – an integrated approach. In *Proceedings of the SPIE Conference on Visual Communications and Image Processing*, Boston, MA, September 1991.

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

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

[16] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Assocation of Computing Machinery*, 20(1):46–61, January 1973.

[17] Jane Liu and Victor Lopez-Millan. A congestion control scheme for a real-time traffic switching element using the imprecise computations technique. In *Proceedings of the IEEE IPPS 1st Workshop on Parallel and Distributed Real-Time Systems*, pages 89–93, Newport Beach, CA, April 1993.

[18] Yuh-Dauh Lyuu. Fast fault-tolerant parallel communication and on-line maintenance using information dispersal. Technical Report TR-19-1989, Harvard University, Cambridge, Massachusetts, October 1989.

[19] 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.

[20] A. Shamir. How to share a secret? *Communication of the ACM*, 22:612–613, November 1979.

[21] 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.