Appears in IEEE Data Engineering Bulletin, Vol. 19, No. 3, September 1996.

Dissemination-Based Information Systems*

Michael Franklin
UMIACS and Dept. of Computer Science
University of Maryland
College Park, MD 20742 USA
franklin@cs.umd.edu

Stan Zdonik
Department of Computer Science
Brown University
Providence, RI 02912 USA
sbz@cs.brown.edu

Abstract

This paper examines the construction of distributed information systems that incorporate the ability to push data out to clients (i.e., dissemination) in addition to having clients pull data from servers. One key issue that distinguishes such dissemination-based information systems (DBIS) from more traditional ones is communications asymmetry. Asymmetry arises in many new applications due to both physical characteristics such as network bandwidths, as well as to workload characteristics. We outline a number of different data delivery mechanisms, and then focus on one particular dissemination mechanism that we have developed, called Broadcast Disks. Finally, we discuss issues that arise in the design of a DBIS architecture that can provide many different modes of data delivery.

1 Introduction

Ongoing advances in communications including the proliferation of the Internet and intranets, the development of mobile and wireless networks, and the impending availability of high bandwidth links to the home, have fueled the development of a wide range of new information-centered applications. Many of these new applications involve *data dissemination*, that is, the delivery of data from a set of *producers* to a (typically) larger set of *consumers*. Examples of dissemination-based applications include information feeds (e.g., stock and sports tickers or news wires), traffic information systems, electronic newsletters, and entertainment delivery.

Although dissemination-based applications share some common characteristics with other data-intensive applications such as query processing or decision support, the specific style of information flow that arises due to dissemination imposes a special set of demands on an information system architecture. One key aspect of dissemination-based applications is their inherent communications *asymmetry*. That is, the communication capacity or data volume in the downstream direction, (i.e., from servers-to-clients) is much greater than that in the upstream direction (i.e., from clients-to-servers). Dissemination can be both the cause of and solution to asymmetry in an information system. A stock ticker application is an example of the first case. It is asymmetric because the information flow is unidirectional: from the single (or small number of) source(s) where the ticker originates, to the much larger set of clients that monitor the stock ticker. An example of the second case is a wireless network in which fixed base stations can transmit at a bandwidth that is much higher than that at which the mobile clients can transmit. In this latter example, data dissemination is a natural way to exploit the base stations' advantage in bandwidth.

In our work on the Broadcast Disk paradigm for data delivery [Zdon94, Acha95b], we have identified a number of ways that the asymmetric nature of data dissemination can impact the design of an information system.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work has been partially supported by NSF grant IRI-9501353, ONR grant number N00014-91-J-4085 under ARPA order number 8220, and by a gift from Intel Corporation.

The issues that we have studied include broadcast scheduling and client cache management [Acha95a], client prefetching [Acha96a], and the dissemination of updates [Acha96b]. Our current work is focused on integrating the notion of Broadcast Disks and other forms of data dissemination into a coherent framework, which we refer to as a *Dissemination-Based Information System* (DBIS) architecture.

In what follows, we first examine the causes of communications asymmetry and their relationship to data dissemination. We next outline a number of different data delivery mechanisms, and then focus on Broadcast Disks as one example of such a mechanism. Finally, we discuss our notion of a DBIS and sketch some of the issues that arise in the design of such a system.

2 Communications Asymmetry

As stated previously, a key aspect of dissemination-based applications is that they are asymmetric in the amount of data transmitted from servers to clients versus from clients to servers. It is interesting to note, that while much existing communications infrastructure is symmetric (e.g., the Internet), most information intensive applications that use that infrastructure are asymmetric. Symmetric applications, of course, do exist. Examples include teleconferencing, email, and multi-user games. As bandwidth and connectivity continue to improve, however, networks are increasingly being used to deliver sophisticated content to communities of users. Content delivery is an asymmetric process regardless of whether it is performed over a symmetric channel such as the Internet, or over an asymmetric one, such as a cable television (CATV) network. Thus, regardless of whether or not one expects future networks to be symmetric or asymmetric, it is clear that many important applications of future networks will exhibit asymmetry. Techniques and system architectures that can efficiently support asymmetric applications will therefore be a requirement for future information systems.

Asymmetry can arise from a number of factors, both environmental and application-based. Each of these different types of asymmetry imposes a unique set of demands on a system architecture. Below, we identify several causes of asymmetry. In the subsequent section we examine the available options for data delivery and discuss how each of those options addresses the problems that can arise due to the different types of asymmetry.

2.1 Network Asymmetry

Perhaps the most obvious form of asymmetry occurs when the network bandwidth differs in the downstream (server to client) and upstream (client to server) directions. An example of an asymmetric network is the mobile case mentioned above — mobile computers are supplied with data at a high rate while mobile clients are unable to transmit or can do so only over a very slow cellular connection. Another important example can be found in technologies that are being developed to provide multi-megabit per second data delivery to the home. Three industry segments are vying to be the central provider of such services: the telephone companies (through ADSL and other technologies), the cable television companies (through the use of high-bandwidth cable modems), and direct broadcast satellite providers, such as Hughes' DirecPC. All three of these potential providers are planning, at least for the near term, to provide asymmetric bandwidth. For example, cable systems will allow home computers (or televisions) to be supplied with video and other data at speeds approaching five to ten megabits per second while the home machines will be able to transmit back to the server over channels of only tens of kilobits per second. Furthermore, in some situations, these slow "backchannels" must be shared among several households.

2.2 Client to Server Ratio

A system with a small number of servers and a very large number of clients can be viewed as another example of an asymmetric system because the server message and data processing capacity must be divided among all of the clients. For example, it is quite common for popular World-Wide-Web (WWW) servers to become "swamped" due to an unexpectedly large volume of requests. Such behavior is particularly likely when new features are announced, such as the availability of a new version of a popular software package. When a server becomes swamped, it will in the best case, simply refuse to accept additional connections. In the worst (but currently not unlikely) case, such a server will crash.

2.3 Data Volume

A third way that asymmetry arises is due to the volume of data that is transmitted in each direction. Information retrieval applications typically involve a small request message (e.g., containing a few query terms) and a much larger result. For such applications, the downstream bandwidth requirements for a given client are much higher than the upstream bandwidth requirements. For example, consider an application such as Intel's Intercast, in which URLs of related web pages are downloaded along with a television broadcast. If a viewer wishes to follow up on the related information, they can click on the appropriate URL, which could result in the downloading of significant amounts of data such as maps, related video clips, etc. In this case, the data sent upstream is quite small (i.e., it is basically the "key and mouse clicks") while the data sent downstream can be multiple megabytes or more.

2.4 Updates and Information Creation

A fourth type of asymmetry arises when the data to be sent to clients is frequently updated and/or new data of potential interest to clients is frequently created. In these situations, even though the backchannel might have a reasonably high bandwidth, the extremely high load that would be created by having clients poll to extract the latest updates creates an effective asymmetry. That is, it is not efficient (or in some cases, even feasible) to allow clients to continuously poll for new information as such polling places a high load on both the network and the server(s) as well as on the clients themselves.

2.5 Implications of Asymmetry

These different types of asymmetry all impose somewhat different demands on an information system. Network asymmetry places severe limitations on the use of the client-to-server backchannel, either in terms of the size of messages that can be sent, the number of messages that can be sent, or both. A high client/server ratio also imposes restrictions on backchannel usage, and in addition, makes it expensive for the server to send individual reply messages to each client. Data volume asymmetry implies that a significant amount of upstream bandwidth will go unused in a symmetric network. Finally, in the case of updates and new information, clients could spend significant portions of their own resources (in addition to those of the backchannel and server(s)) if they are required to poll continuously for new data. The frequency of such polling also has the potential to impact the integrity (i.e., currency) of data that is cached at clients.

3 Options for Data Delivery

A dissemination-based information system must incorporate a number of different styles of data delivery in order to efficiently cope with the various types of asymmetry described in the previous section, as well as to support symmetric applications. In addition, it is important to provide different types of data delivery so that the system can be optimized for various criteria. For example, even in an asymmetric network where it is infeasible to allow all clients to use the client-to-server backchannel, it may be desirable to provide upstream bandwidth to those users who are willing to pay a premium for it.

In this section, we identify several characteristics that can be used to compare data delivery mechanisms. Figure 1 shows the three main characteristics we consider: (1) push vs. pull; (2) periodic vs. aperiodic; and (3) point-to-point vs. one-to-N. The leaves of the tree in the figure show how several common mechanisms relate to these characteristics.

3.1 Client Pull vs. Server Push

The first distinction we make among data delivery styles is that of "pull vs. push". Current database servers and object repositories manage data for clients that explicitly request data when they require it. When a request is received at a server, the server locates the information of interest and returns it to the client. This *request-response* style of operation is *pull-based* — the transfer of information from servers to clients is initiated by a client pull. In contrast, data dissemination involves sending information to a client population, possibly in advance

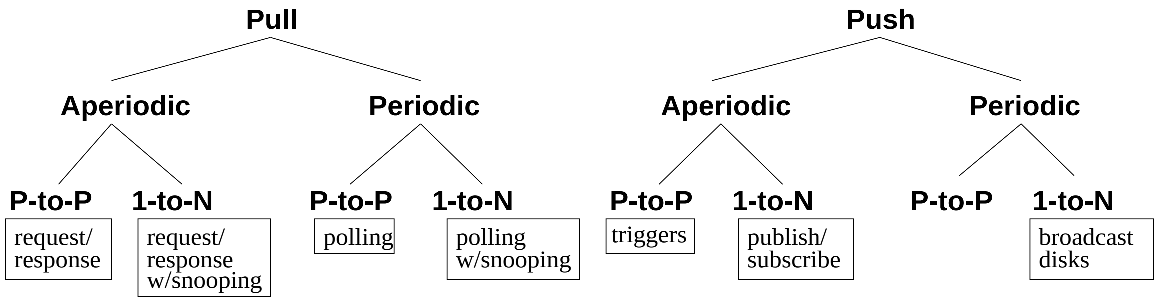

Figure 1: Data Delivery Options

of any specific request. This is referred to as a *push-based* approach since information transfer is initiated by a server push.

The tradeoffs between push and pull revolve around the costs of initiating the transfer of data. A pull-based approach requires the use of a backchannel for each request. Furthermore, the server must be interrupted continuously to deal with such requests and has limited flexibility in scheduling the order of data delivery. Also, the information that clients can obtain from a server is limited to that which the clients know to ask for. Thus, new data items or updates to existing data items may go unnoticed at clients unless they periodically poll the server.

Push-based approaches, in contrast, avoid the issues identified for client-pull, but have the problem of deciding which data to send to clients in the absence of specific requests. Clearly, sending irrelevant data to clients is a waste of resources. A more serious problem, however, is that in the absence of requests it is possible that the servers will not deliver the specific data that are needed by clients in a timely fashion (if ever). Thus, the usefulness of server push is dependent on the ability of a server to accurately predict the needs of clients. A solution to this problem to allow the clients to provide a *profile* of their interests to the servers. A popular mechanism for providing such profiles is known as a *publish/subscribe* protocol (e.g., [Oki93]). Using publish/subscribe, clients subscribe to given classes of information by providing a set of expressions that describe the data of interest. These subscriptions form a profile — they are in effect, a continuous query. The information providers publish information that is tagged in a way that allows the matching of the data with the subscriptions of the users. When new data items are created or existing ones updated, the items are disseminated to the subscribers whose profiles match the items. The distributed system infrastructure is responsible for delivering data to all relevant subscribers. While publish/subscribe is useful for delivering new or modified data to clients, it can not be used to efficiently deliver previously existing data to clients that subsequently realize they need it. Such data is most easily obtained through a client pull.

3.2 Aperiodic vs. Periodic

Both push and pull can be performed in either an aperiodic or periodic fashion. Periodic delivery is performed on a regular, repeating schedule, while aperiodic has no such schedule. For example, an application that sends out stock prices on a regular basis is an example of periodic push, whereas one that sends out stock prices only when they change is an example of aperiodic push. Using periodic push, a set of data items is sent out repeatedly according to a prespecified schedule. Since the pattern repeats, a client that misses a data item on one period of the pattern, it can get it the next. In contrast, using aperiodic push, data are sent out, but not according to any particular schedule. This type of data delivery is what would arise in a publish/subscribe system.

Periodic push is useful for cases in which clients might not be available at all times or might be unable to react to what has been sent. Unavailability can occur in mobile settings where clients can become disconnected. It might also occur in cases in which the client processing load is high, thus preventing the continual servicing of the incoming data stream. On the other hand, aperiodic push assumes that clients are either always listening and able to respond to what is being sent or that missing some information is not crucial to the application. Aperiodic push has the advantage of making more effective use of the downstream communication channel, as the choice of items to deliver can be adjusted more quickly to the changing needs of an application.

Aperiodic pull arises in a system that uses the traditional request/response style of data delivery. Data is pulled from the server to clients whenever clients request it, for example, on a page fault or a cache (buffer) miss. In contrast, periodic pull arises when a client uses polling to obtain data from servers. The client requests data on a regular schedule. Such polling may be useful in certain applications such as remote-sensing, but as discussed previously, can be expensive in terms of server and client processing as well as backchannel usage.

3.3 Point-to-point vs. One-to-N

The third characteristic of data delivery mechanisms we identify is whether they are based on point-to-point or one-to-N communication. Using point-to-point communication, data is sent from a data source (e.g., a single server) to one other machine, while one-to-N communication such as broadcast or multicast allows multiple machines to receive the data sent by a data source. The tradeoffs between these approaches depend upon the commonality of interest of the clients. Using broadcast or multicast, scalability can be improved by allowing multiple clients to receive data sent using a single server message. Such benefits can be obtained, however, only if multiple clients are interested in the same items. If not, then scalability may actually be harmed, as clients may be continually interrupted to filter data that is not of interest to them.

Two types of one-to-N data delivery can be distinguished: multicast and broadcast. With multicast, data is sent to a specific subset of clients. Multicast is typically implemented by sending a message to a router that maintains the list of recipients. The router reroutes the message to each member of the list. Since the list of recipients is known, it is possible to make multicast reliable; that is, network protocols can be developed that guarantee the eventual delivery of the message to all clients that should receive it. In contrast, broadcasting sends information over a medium on which an unidentified and unbounded set of clients can listen. This differs from multicast in that the clients who may receive the data are not known *a priori*.

3.4 Classification of Delivery Mechanisms

It is possible to classify some existing data delivery mechanisms using the framework described above. Such a classification is shown in Figure 1.

- *Request/Response* - Traditional request/response mechanisms use aperiodic pull over a point-to-point connection. If instead, a one-to-N connection is used, then clients can "snoop" on the requests made by other clients, and obtain data that they haven't explicitly asked for. It is interesting to note, therefore, that in the one-to-N case, request/response looks like a pull-based approach to the client that initiates a request, but appears to be push-based to any other clients that are snooping on the broadcast.

- *Polling* - In some applications, such as remote sensing or control applications, a system may periodically send requests to other sites to obtain status information or to detect changed values. If the requested information is returned over a point-to-point channel, then this pull-based approach is what is usually considered to be polling. If the information is returned over a one-to-N link, then as with request/response, other clients can snoop to obtain data as it goes by.

- *Publish/Subscribe* - As stated previously, publish/subscribe protocols are becoming a popular way to disseminate information in a network. Publish/subscribe is push-based; data flow is initiated by the data sources (i.e., servers), and is aperiodic, as there is no predefined schedule for sending data. Publish/subscribe protocols are typically performed in a one-to-N fashion, but a similar protocol can be used over a point-to-point channel, as is done for triggers or notifications in an active database system.

- *Broadcast Disks* - The last category we examine is the combination of periodic and push. Periodic push has been used for data dissemination in many systems such as TeleText [Amma85, Wong88] and DataCycle [Herm87, Bowe92]. Clients that need to access a data item that is pushed periodically can wait until the item appears. In this way, periodic push can be thought of as a storage device whose average latency for an item is half the period at which that item repeats [Acha95a]. As with aperiodic push, periodic push can also be used with both point-to-point and one-to-N channels, but one-to-N is likely to be much more prevalent. We discus this method of data delivery in more detail in the following section.

4 Broadcast Disks

Having described the notion of data dissemination and outlined the different styles of data delivery that can be employed in a DBIS, we now focus on one particular approach that we have developed, called Broadcast Disks [Zdon94, Acha95b]. Using Broadcast Disks, a broadcast program containing all the data items to be disseminated is determined and this program is transmitted in a periodic manner. As stated above, the repetitive nature of the broadcast allows the medium to be used as a type of storage device; clients that need to access a data item from the broadcast can obtain the item at any point in the broadcast schedule that the item is transmitted. Periodic broadcast has been studied by a number of researchers (e.g., [Amma85, Herm87, Giff90, Imie94]). The Broadcast Disk model differs from previous data broadcasting work, however, in that it integrates two main components: 1) a novel "multi-level" structuring mechanism that allows data items to be broadcast non-uniformly, so that bandwidth can be allocated to data items according to the importance of those items; and 2) client-side storage management algorithms for data caching and prefetching that are tailored to the multi-disk broadcast. As is discussed below, these client-side policies differ in significant ways from those that would be used for the other styles of data delivery (e.g., pull-based, or aperiodic push-based).

4.1 Server-Side Broadcast Scheduling

The multi-level approach to broadcast scheduling used in Broadcast Disks allows data items to be placed on sub-broadcasts of varying size and periodicity. An arbitrarily fine-grained memory hierarchy can be created by constructing a multi-level broadcast such that the highest level is relatively small and repeats relatively frequently, while subsequent levels are progressively larger and repeat progressively less frequently. Such a broadcast program is analogous to a traditional memory hierarchy ranging from small, fast memories (e.g., cache) to larger, slower memories (e.g., disks or near-line storage). By placing data items on faster levels, the access time for higher-priority data can be reduced at the expense of increasing the latency for lower-priority items. For instance, in a highway traffic information system, information on accidents or potential delays can be placed on faster levels, while less time critical information (such as local point-of-interest listings) can be placed on slower levels. In this way, newly arriving motorists can receive data about incidents more quickly than they could if all information was broadcast in a uniform (i.e., flat) manner.

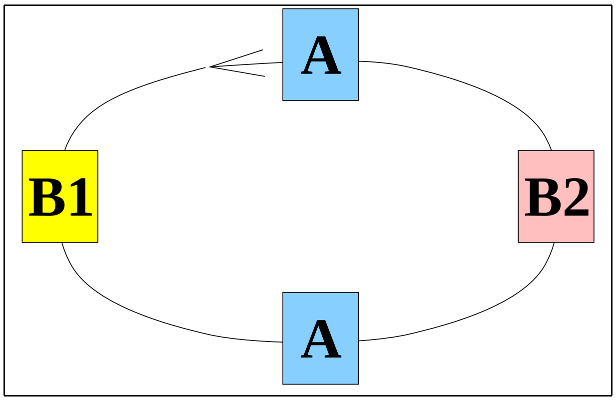

Figure 2: Multidisk with 3 distinct pages on 2 disks

Figure 2 shows a simple example broadcast of three pages arranged on two disks. The page labeled A is on a disk that is spinning twice as fast as the two pages that are labeled B1 and B2. The pages B1 and B2 are said to be on a slower spinning disk that is twice the size of the fast disk. The desirable characteristics for a multi-level broadcast program have been outlined in [Acha95a]. Briefly, a good broadcast program is periodic, has fixed (or nearly fixed) inter-arrival times for repeated occurrences of a data item, and allocates bandwidth to items in accordance with (but not in direct proportion to) their access probabilities. Multiple levels are superimposed on a single broadcast channel by interleaving the data items of the various levels in a manner that results in the desired relative frequencies. For example, the multidisk shown in Figure 2 would be generated by repeatedly broadcasting the pattern :"A, B1, A, B2, ...". The algorithm used by the server to generate the broadcast program requires the following inputs: the number of levels , the relative frequency of each level and assignments of data items to the levels on which they are to be broadcast. The algorithm is described in detail in [Acha95a]. More recently, algorithms for automatically determining these inputs have been proposed [Baru96].

4.2 Profiles, Feedback, and Client-side Storage Management

In order to decide how to structure a broadcast schedule, the server must use its best knowledge and synthesis of the needs of the clients that require disseminated data. There are two potential pitfalls with this approach. First, the server is likely to have inaccurate data about the needs of the clients, as these needs can change quickly and are often hard even for the clients themselves to predict. Second, even given perfect knowledge of the access profiles of the clients, the server must *synthesize* an "average" profile on which to base its broadcast schedule; this average profile is unlikely to be optimal for any *individual* client. A key to solving this problem is the use of client storage resources for caching and prefetching data items. By intelligent management of local storage, a client can (to a large extent) isolate itself from many of the mismatches between its local priorities and the global priorities determined at the server.

We have developed implementable caching and prefetching algorithms to address this issue and have compared them to ideal policies [Acha95a, Acha96a]. The policies take into account both the local access probability (using heuristics such as LRU or usage frequency) and the expected latency of a data item; items that reside on slower levels have higher average latencies. The caching policies [Acha95a] favor slow-level data items, allowing clients to use their local storage resources to retain pages which are important locally, but are not sufficiently popular across all clients to warrant placement on a faster level. Broadcast Disks presents an excellent opportunity for prefetching because objects continually flow past the receivers. We have also developed client *prefetching* algorithms [Acha96a]. These algorithms are more dynamic than the caching policies because they take into consideration the time-to-arrival for items at a given instant. The prefetching policies are able to retain items in a client's cache during those portions of the broadcast cycle where their time-to-arrival is at its highest, which results in a significant performance improvement.

Intelligent client cache management goes a long way towards relaxing the need for strict accuracy in user profile management. The feedback from user requests and status information delivered over a backchannel, however, can provide important information to help maintain the profile information at server. Algorithms for dynamically maintaining profiles and adjusting the broadcast accordingly are thus, important directions in our current work.

4.3 Data Consistency

We have extended the mechanisms described above in order to incorporate data item updates and have studied their effect on performance [Acha96b]. As in the previous work, the design considerations divide into client-side and server-side issues. The server must decide how to modify its broadcast program in order to most effectively communicate updates to the client. The server can use both invalidation and propagation. Timely invalidation (i.e., notification of updates) is the minimal requirement in order to enable the preservation of data consistency. The server can also choose to propagate the new values of updated items in order to enhance performance. The client must perform appropriate actions to bring its cache contents back to a good state after an update. For example, the clients can prefetch items that have been invalidated from its cache. We have implemented this *automatic* prefetching and have integrated it with the regular (non-update) prefetching algorithms. Alternatively, a client can choose to ignore an invalidation if it determines that retaining a stale copy of a data item is preferable to having no copy of that item.

Two different consistency models have been studied — immediate and periodic (i.e., based on the period of the broadcast). Our studies have shown that under both models, for low to moderate update rates it is possible to approach the performance of the case in which no updates occur. In other words, the broadcast mechanism can be made quite robust in the face of updates. One reason for this is that when invalidated items are automatically prefetched by clients, the broadcast program itself can act as a propagation medium. Explicitly propagating an item, therefore, is useful only if clients are likely to need to access that item long before it would naturally appear in the broadcast; otherwise, the propagation wastes bandwidth.

5 Towards an Integrated DBIS

One of the key lessons of the Broadcast Disks work has been the importance considering server-side, client-side, and network issues when developing a data dissemination technique. As described in Section 3, the periodic push approach used by Broadcast Disks is just one potential technique for delivering data in a DBIS. Our current work is focused on developing a more general dissemination-based information system architecture that is capable of supporting a wide range of applications across many varied environments (e.g., mobile networks, satellites, wide-area networks). The key to achieving this flexibility is to combine the various data delivery techniques in a way that achieves the most efficient use of the communication channels, while providing clients with responsive access to data.

We view an integrated DBIS as a system in which the links between the computing elements vary in character: from standard pull-based connections to periodic broadcast-based links. The choice of the data delivery mechanism to use between two given nodes will be made based on the properties of the available communications channels and applications as well as economic concerns. These factors may be stable or may change over time, requiring the DBIS to adapt dynamically. A key point is that the character of a link should be of concern only to the nodes on either end. For example, the fact that an information provider receives data from a broadcast disk as opposed to a request/response protocol should make no difference to clients of that provider.

In an integrated DBIS, there will be three types of nodes: (1) *data sources*, which provide the base data that is to be disseminated; (2) *clients*, which are net consumers of information; and (3) *information brokers*, that acquire information from other sources, add value to that information (e.g., some additional computation or organizational structure) and then distribute this information to other consumers. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

The goal is to provide the architectural components that can be used to construct a DBIS. A builder of an information resource would make use of a library of these components to construct the interfaces to their service. Example components include a broadcast generator, a set of dissemination services, a client cache manager, a client prefetcher, a backchannel monitor, etc. In the following, we briefly outline some of the fundamental issues in the design of an integrated DBIS:

- *Bandwidth Allocation* - For a given link, policies are needed for allocating bandwidth among the various data delivery mechanisms.

- *Push Scheduling* - For the push-based approaches, intelligent scheduling is necessary in order to obtain the maximal benefit from the available bandwidth. Scheduling must also take into account the likelihood and distribution of transmission errors. Also, for periodic push, the broadcast should include index and/or schedule information that describes the objects that are to appear in the upcoming broadcast. Such information allows clients to minimize the amount of time and/or processing that they devote to monitoring the broadcast and can aid in storage management decisions.

- *Client Storage Management* - Clients must allocate their storage resources among the data obtained through the various delivery mechanisms. Furthermore, as stated earlier, different methods of data delivery impose differing demands on the policies for client caching and prefetching. Furthermore, in some cases (e.g., mobility), storage management must also take into account the likelihood of disconnection and of data becoming stale due to updates or expiration.

- *User Profiles and Feedback* - Profiles of client needs are key for making allocation, scheduling and other policy decisions at both clients and servers. The form of the profiles will be important to achieve the most effective use of the medium. For example, access probabilities are one specific representation of the client needs. The server must also have effective models for combining client profiles. The integration of a *backchannel* from clients to servers is needed to allow for updating profiles and making additional requests.

- *Security Issues* - Another set of important issues that must be addressed revolves around the security and privacy concerns that arise in any distributed information system. The emphasis on one-to-N commuinca-tion in a DBIS, however, increases the significance of such issues.

- *Consistency Issues* - The final issue we list here is the maintenance of data consistency, particularly in the face of possibly intermittent connection. Two types of consistency must be considered. First, guarantees on the timeliness of individual data items must be provided if required by the clients. Second, *mutual consistency* across multiple items will be required in some instances. All types of consistency must be provided in a flexible manner, so that tradeoffs between consistency and responsiveness can be made on a case-by-case basis.

6 Summary

The increasing ability to interconnect computers through internetworking, mobile and wireless networks, and high-bandwidth content delivery to the home, has resulted in the proliferation of information delivery applications. A key attribute of many such applications is their inherent asymmetry. These applications present new challenges for data management throughout all components of a distributed information system. We have proposed the notion of a dissemination-based information system that integrates many different data delivery mechanisms and described some of the unique aspects of such systems. We also described the Broadcast Disks data dissemination paradigm and showed how the choice of data delivery mechanism effects both clients and servers. Our current work is aimed at integrating Broadcast Disks with a pull-based request/response mechanism. We view this work as a step towards the development of a more general DBIS architecture.

References

[Acha95a] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", *Proc. ACM SIGMOD Conference*, San Jose, CA, May, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", *IEEE Personal Communications*, 2(6), December, 1995.

[Acha96a] S. Acharya, M. Franklin, S. Zdonik, "Prefetching from a Broadcast Disk", *12th International Conference on Data Engineering*, New Orleans, LA, February, 1996.

[Acha96b] S. Acharya, M. Franklin, S. Zdonik, "Disseminating Updates on Broadcast Disks", *Proc. 22nd VLDB Conference*, Mumbai (Bombay), India, September, 1996.

[Amma85] M. Ammar, J. Wong, "The Design of Teletext Broadcast Cycles", *Perf. Evaluation*, 5 (1985).

[Baru96] S. Baruah and A. Bestavros, "Pinwheel Scheduling for Fault-tolerant Broadcast Disks in Real-time Database Systems",Technical Report TR-96-023, Boston University, August, 1996.

[Bowe92] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, A. Weinrib, "The Datacycle Architecture", *CACM*, 35(12), December, 1992.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", CACM, 33(2), February, 1990.

[Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May, 1987.

[Imie94] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *Proc. ACM SIGMOD Conference*, Minneapolis, MN, May, 1994.

[Oki93] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus - An Architecture for Extensible Distributed Systems", *Proc. 14th SOSP Conference*, Ashville, NC, December, 1993.

[Wong88] J. Wong, "Broadcast Delivery", *Proceedings of the IEEE*, 76(12), December, 1988.

[Zdon94] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky?", *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December, 1994.