Research in Data Broadcast and Dissemination

Demet Aksoy[2], Mehmet Altinel[2], Rahul Bose[1], Ugur Cetintemel[2], Michael Franklin[2], Jane Wang[1] and Stan Zdonik[1]

[1] Department of Computer Science, Brown University, Providence, RI 02912
[2] Department of Computer Science, University of Maryland, College Park, MD 20742

1 Introduction

The proliferation of the Internet and intranets, the development of wireless and satellite networks, and the availability of asymmetric, high-bandwidth links to the home, have fueled the development of a wide range of new "dissemination-based" applications. These applications involve the timely distribution of data to a large set of consumers, and include stock and sports tickers, traffic information systems, electronic personalized newspapers, and entertainment delivery. Dissemination-oriented applications have special characteristics that render traditional client-server data management approaches ineffective. These include:

- tremendous scale.
- a high-degree of overlap in user data needs.
- asymmetric data flow from sources to consumers.

For example, consider a dissemination-oriented application such as an election result server. Typically, such applications are implemented by simply posting information and updates on a World Wide Web server. Such servers, however, can and often do become overloaded, resulting in the inability for users to access the information in a timely fashion. We argue that such scalability problems are the result of a mismatch between the data access characteristics of the application and the technology (in this case, HTTP) used to implement the application. HTTP is based on a request-response or RPC, unicast (i.e., point-to-point) method of data delivery, which is simply the wrong approach for this type of application.

Using request-response, each user sends requests for data to the server. The large audience for a popular event can generate huge spikes in the load at servers, resulting in long delays and server crashes. Compounding the situation is that users must continually *poll* the server to obtain the most current data, resulting in multiple requests for the same data items from each user. In an application such as an election server, where the interests of a large part of the population are known *a priori*, most of these requests are unnecessary.

The use of unicast data delivery likewise causes problems in the opposite direction (from servers to clients). With unicast the server is required to respond individually to each request, often transmitting identical data. For an application with many users, the costs of this repetition in terms of network bandwidth and server cycles can be devastating.

To address the particular needs of dissemination-based applications, we are developing a general framework for describing and constructing Dissemination-Based Information Systems (DBIS). The framework incorporates a number of data delivery mechanisms and an architecture for deploying them in a networked environment. The goal is to support a wide range of applications across many varied environments, such as mobile networks, satellite-based systems, and wide-area networks. By combining the various data delivery techniques in a way that matches the characteristics of the application and achieves the most efficient use of the available server and communication resources, the scalability and performance of dissemination-oriented applications can be greatly enhanced.

In this paper, we provide an overview of the current status of our DBIS research efforts. We first explain the framework and then describe our initial prototype of a DBIS toolkit. We then focus on several research results that have arisen from this effort.

2 The DBIS Framework

There are two major aspects of the DBIS framework.[3] First, the framework incorporates a number of different options for data delivery. A taxonomy of these options is presented in Section 2.1 and the methods are further discussed in Section 2.2. Secondly, the framework exploits the notion of *network transparency*, which allows data delivery mechanisms to be mixed-and-matched within a single application. This latter aspect of the framework is described in Section 2.3.

2.1 Options for Data Delivery

We identify three main characteristics that can be used to describe data delivery mechanisms: (1) push vs. pull; (2) periodic vs. aperiodic; and (3) unicast vs. 1-to-N. Figure 1 shows these characteristics and how several common mechanisms relate to them.

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 support clients that explicitly send requests for data items when they require them. 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, push-based data delivery involves sending information to a client population in advance of any specific request. With push-based delivery, the server initiates the transfer.

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

[3] Parts of this section have been adapted from an earlier paper, which appeared in the 1997 ACM OOPSLA Conference [Fran97].

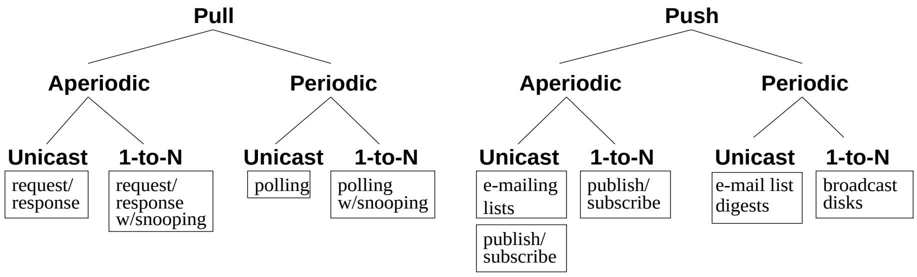

Fig. 1. Data Delivery Characteristics

each request. Furthermore, as described in the Introduction, 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 those clients 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. One solution to this problem is to allow the clients to provide a *profile* of their interests to the servers. *Publish/subscribe* protocols are one popular mechanism for providing such profiles.

Aperiodic vs. Periodic - Both push and pull can be performed in either an aperiodic or periodic fashion. Aperiodic delivery is *event-driven* — a data request (for pull) or transmission (for push) is triggered by an event such as a user action (for pull) or data update (for push). In contrast, periodic delivery is performed according to some pre-arranged schedule. This schedule may be fixed, or may be generated with some degree of randomness.[4] 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.

[4] For the purposes of this discussion, we do not distinguish between fixed and randomized schedules. Such a distinction is important in certain applications. For example, algorithms for conserving energy in mobile environments proposed by Imielinski et al. [Imie94b] depend on a strict schedule to allow mobile clients to "doze" during periods when no data of interest to them will be broadcast.

Unicast vs. 1-to-N - The third characteristic of data delivery mechanisms we identify is whether they are based on unicast or 1-to-N communication. With unicast communication, data items are sent from a data source (e.g., a single server) to one other machine, while 1-to-N communication allows multiple machines to receive the data sent by a data source. Two types of 1-to-N data delivery can be distinguished: multicast and broadcast. With multicast, data is sent to a specific subset of clients. In some systems multicast is 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*.

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.

2.2 Classification of Delivery Mechanisms

It is possible to classify many existing data delivery mechanisms using the characteristics described above. Such a classification is shown in Figure 1. We discuss several of the mechanisms below.

Aperiodic Pull - Traditional request/response mechanisms use aperiodic pull over a unicast connection. If instead, a 1-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 (e.g, see [Acha97, Akso98]).

Periodic Pull - In some applications, such as remote sensing, a system may periodically send requests to other sites to obtain status information or to detect changed values. If the information is returned over a 1-to-N link, then as with request/response, other clients can snoop to obtain data items as they go by. Most existing Web or Internet-based "push" systems are actually implemented using Periodic Pull between the client machines and the data source(s) [Fran98].

Aperiodic Push - Publish/subscribe protocols are becoming a popular way to disseminate information in a network [Oki93, Yan95, Glan96]. In a publish/subscribe system, users provide information (sometimes in the form of a profile) indicating the types of information they wish to receive. Publish/subscribe is push-based; data flow is initiated by the data sources, and is aperiodic, as there is no predefined schedule for sending data. Publish/subscribe protocols are inherently 1-to-N in nature, but due to limitations in current Internet technology, they are often implemented using individual unicast messages to multiple clients. Examples of such systems include Internet e-mail lists and some existing "push" systems on the Internet. True 1-to-N delivery is possible through technologies such as IP-Multicast, but such solutions are not universally available across the Internet.

Periodic Push - Periodic push has been used for data dissemination in many systems. An example of Periodic Push using unicast is Internet mailing lists that send out "digests" on a regular schedule. For example, the Majordomo system allows a list manager to set up a schedule (e.g., weekly) for sending digests. Such digests allow users to follow a mailing list without being continually interrupted by individual messages. There have also been many systems that use Periodic Push over a broadcast or multicast link. These include TeleText [Amma85, Wong88], DataCycle [Herm87], Broadcast Disks [Acha95a, Acha95b] and mobile databases [Imie94b].

2.3 Network Transparency

The previous discussion has focused primarily on different modes of data delivery. The second aspect of the DBIS framework addresses how those delivery modes are used to facilitate the efficient transfer of data through the nodes of a DBIS network. The DBIS framework defines three types of nodes:

1. *Data Sources*, which provide the base data to be disseminated.
2. *Clients*, which are net consumers of information.
3. *Information Brokers*, (or agents, mediators, etc.), which acquire information from other sources, possibly add value to that information (e.g., some additional computation or organizational structure), and then distribute this information to other consumers.

Brokers are the glue that bind the DBIS together. Brokers are middlemen; a broker acts as a client to some number of data sources, collects and possibly repackages the data it obtains, and then functions as a data source to other nodes of the system. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

The ability of brokers to function as both clients and data sources provides the basis for the notion of Network Transparency. Receivers of information cannot detect the details of interconnections any further upstream than their immediate predecessor. Because of this transparency, the data delivery mechanism used between two or more nodes can be changed without requiring changes to the data delivery mechanisms used for other communication in the DBIS. For example, suppose that node B is pulling data values from node A on demand. Further, suppose that node C is listening to a periodic broadcast from node B which includes values that B has pulled from A. Node C will not have to change it's data gathering strategy if A begins to push values to B. Changes in links are of interest only to the nodes that are directly involved. Likewise, this transparency allows the "appearance" of the data delivery at any node to differ from the way the data is actually delivered earlier in the network. This in turn, allows the data delivery mechanisms to be tailored for a given set of nodes. For example, a broker that typically is very heavily loaded with requests could be an excellent candidate for a push-based delivery mechanism to its clients.

Current Internet "push" technology, such as that provided by PointCast [Rama98] provide an excellent example of network transparency in action. To the user sitting at the screen, the system gives the impression of using aperiodic push over a broadcast channel. Due to current limitations of the Internet, however, that data is actually brought over to the client machine using a stream of periodic pull requests, delivered in a unicast fashion. Thus, the data delivery between the client and the PointCast server is actually the exact opposite of the view that is presented to the user *in all three dimensions* of the hierarchy of Figure 1. This situation is not unique to PointCast; in fact, it is true for virtually all of the Internet-based push solutions, and stems from the fact that current IP and HTTP protocols do not adequately support push or 1-to-N communication.

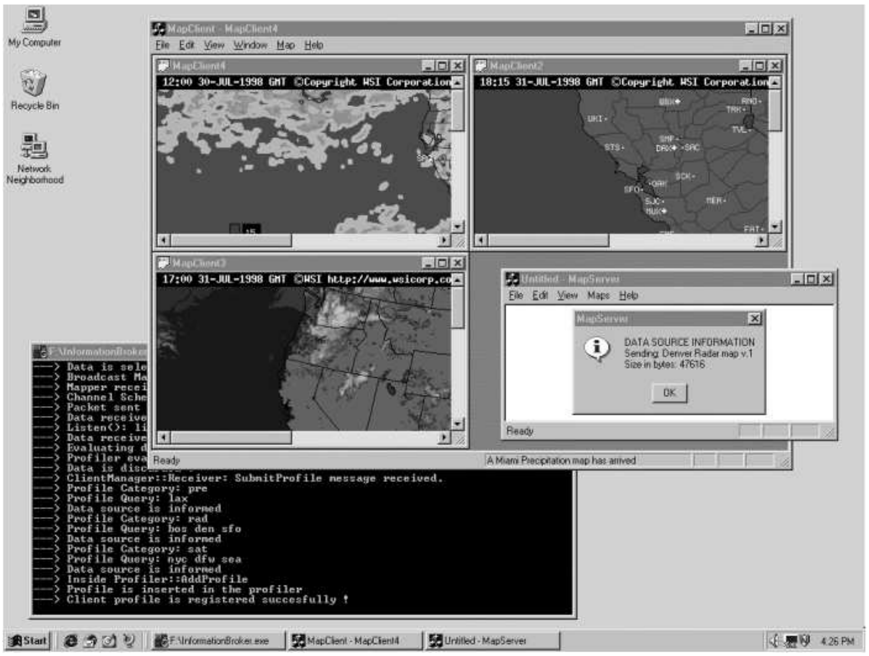

Fig. 2. The Map Dissemination Application

3 An Initial Prototype

As stated in the introduction, our ultimate goal is to build a toolkit of components that can be used to create a DBIS tailored to support a particular set of dissemination-based applications. In order to better understand the requirements and desired properties of such a toolkit, we have constructed an initial prototype toolkit and have used it to implement a weather map dissemination application.

Figure 2 shows an example screen from this application. In this application one or more "map servers" sends out updated maps of different types (i.e., radar, satellite image, etc.) for different regions of the United States. Clients can subscribe to updates for specific types of maps for specific regions. They can also pose queries to obtain the most recent versions of specific maps. The DBIS components route such queries to the appropriate server(s). In the current prototype, all maps are multicast to all clients — the clients perform additional filtering to avoid displaying unrequested results to the user. In the remainder of this section, we briefly describe the implementation of the prototype toolkit.

3.1 Toolkit Description

Figure 3 shows an example instantiation of a DBIS using the current toolkit. The toolkit consists of four main components. These are shown as lightly-shaded items in the figure. The darker shaded items are software that is not part of the DBIS toolkit, namely, the data sources and clients themselves. The components of the current prototype are:

1. **Data Source (DS) Library -** a wrapper for data sources that encapsulates network communication and provides conversion functions for data.
2. **Client Library -** a wrapper for client programs that encapsulates network communication and provides conversion functions for queries and user profiles. The client library is also responsible for monitoring broadcast and multicast channels and filtering out the data items of local interest that appear on those channels.
3. **Information Broker (IB) -** the main component of the DBIS toolkit. The IB contains communication, buffering, scheduling, and catalog management components and is described in more detail below.
4. **Information Broker Master -** The IB Master is responsible for managing global catalog information about data and about the topology of the DBIS. All IBs must register with the IB Master and all catalog updates must be sent to the IB Master. The presence of the IB Master is one of the major limitations of this initial prototype, as it is obviously a potential scalability bottleneck for the system. A large part of the design effort for the next version of the prototype is aimed at distributing the functions of the IB Master.

3.2 Data Modeling Considerations

The DBIS prototype currently uses a simple data model: the catalog consists of a set of category definitions. Categories are application-specific, that is, each

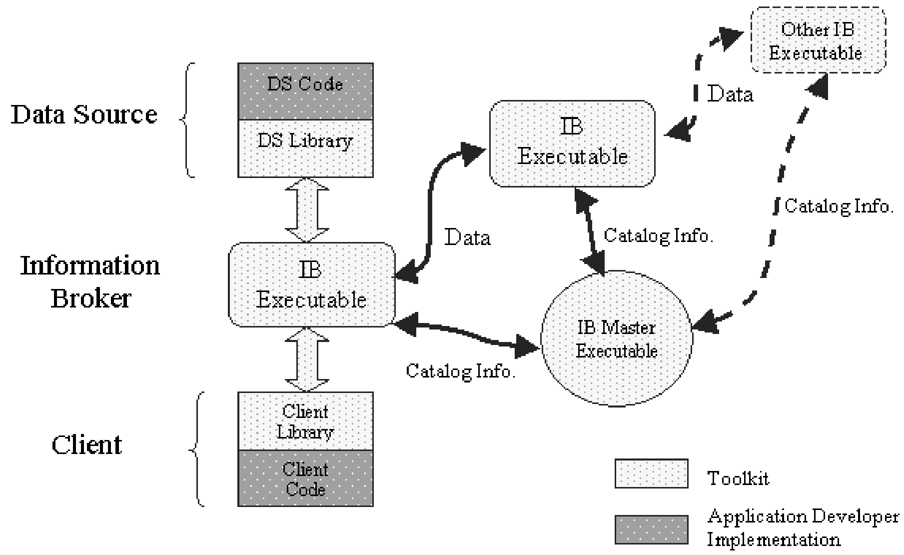

Fig. 3. An Instantiation of a DBIS

application provides its own set of category definitions. Each data item is associated with a single category. In addition, a set of *keywords* can be associated with each data item. Categories and keywords are used in the specification of *queries* and *profiles*. Queries are *pull* requests that are transmitted from a client to a data source. Queries consist of a category and optional keywords. Queries are processed at a data source (or an IB); all data items that match the category (and at least one of the keywords if specified) are sent to the client from which the query originated. In contrast, profiles are used to support *push*-based delivery. When a new data item arrives at an IB, its category and keywords are compared with the user profiles registered at that IB and the item is sent to any clients whose profile indicates an interest in the item. Thus, profiles can be viewed as a form of continually executing query.

Clearly, this simple approach to data modeling must be extended to support more sophisticated applications. We are currently exploring database and WWW-based (e.g., XML) approaches for semi-structured data modeling for use in subsequent versions of the toolkit.

3.3 Information Broker Architecture

As stated above, the Information Broker module contains most of the functionality of the DBIS toolkit. The architecture of an IB is illustrated in Figure 4. Basic components of the IB are the following:

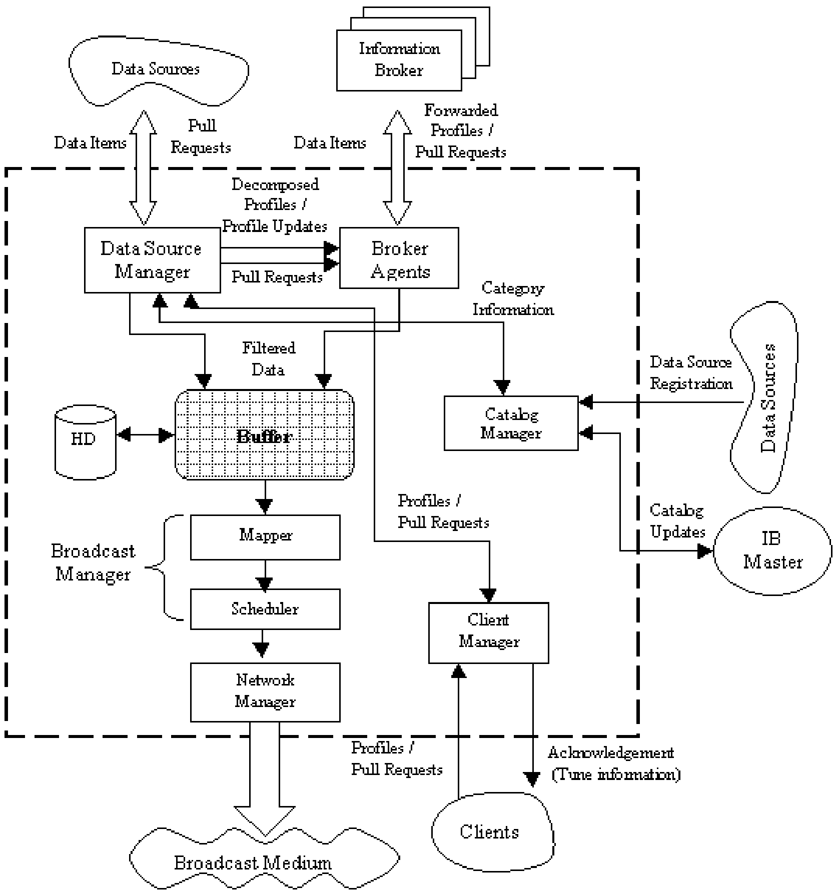

Fig. 4. Information Broker (IB) Architecture

- **Catalog Manager -** This component manages local copies of catalog information for use by the processes running at the broker. Recall that the primary copy of the catalog is managed by the IB Master. All requested changes to the catalog information are sent to the IB Master, which then propagates them to the catalog managers of all other IBs.
- **Data Source Manager -** This component is in charge of receiving and filtering data items obtained from the data sources. It manages a separate listener thread for each data source directly connected to the IB.
- **Broker Agent -** This component is responsible for IB-to-IB interaction, that is, when an IB receives data from another IB rather than directly from a data source.

– **Broadcast Manager -** Once data has been filtered through the data source manager or the broker agent, it is passed to the Broadcast Manager, which has two main components. The *Mapper* assigns the data item to one or more physical communication channels. The *Scheduler* makes decisions about the order in which data items should be placed on those channels.
– **Network Manager -** This is the lowest level of the communication component of the IB. It sends data packets to the network according to the information provided by the broadcast manager.
– **Client Manager -** This module handles all requests that arrive from the clients of the IB. It forwards these requests to the proper modules within the IB and maintains communication sessions with the clients.

4 Example Research Topics

Having described our general approach to building Dissemination-Based Information Systems, we now focus on two examples of the many research issues that arise in the development of such systems.

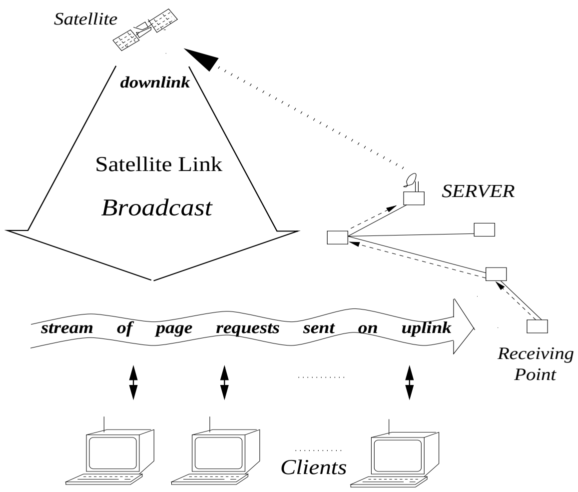

Fig. 5. Example Data Broadcasting Scenario

4.1 Topic 1: On Demand Broadcast Scheduling

As described in Section 2.1, one of the many possible mechanisms for data dissemination uses on-demand (i.e., aperiodic pull) broadcast of data. An example scenario using such data delivery is shown in Figure 5. In this scenario, two independent networks are used: a terrestrial network for sending pull requests to the server, and a "listen only" satellite downlink over which the server broadcasts data to all of the clients. When a client needs a data item (e.g., a web page or database object) that it cannot find locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among these requests, broadcasts it over the satellite link, and removes the associated request(s) from the queue. Clients monitor the broadcast and receive the item(s) that they require.

In a large-scale implementation of such a system, an important consideration is the scheduling algorithm that the server uses to choose which request to service from its queue of waiting requests. We have developed a novel on-demand broadcast scheduling algorithm, called *RxW* [Akso98], which is a practical, low-overhead and scalable approach that provides excellent performance across a range of scenarios.

The intuition behind the *RxW* scheduling algorithm is to provide a balanced performance for hot (popular) and cold (not so popular) pages. This intuition is based on our observations of previously proposed algorithms. We have observed that two low overhead algorithms, Most Requests First (MRF) and First Come First Served (FCFS) [Dyke86, Wong88], have poor average case performance because they favor the broadcasting of hot or cold pages respectively. A third algorithm, Longest Wait First (LWF) [Dyke86, Wong88] was shown to provide fairer treatment of hot and cold pages, and therefore, good average case performance. LWF, however, suffers from high overhead, making it impractical for a large system.

Based on these observations, we set out to combine the two low-overhead approaches (MRF and FCFS) in a way that would balance their strengths and weaknesses. The *RxW* algorithm schedules the page with the maximal $R \times W$ value where R is the number of outstanding requests for that page and W is the amount time that the oldest of those requests has been waiting for the page. Thus, *RxW* schedules a page either because has many outstanding requests or because there is at least one request that has waited for a long time.

The algorithm works by maintaining two sorted lists (one ordered by R values and the other ordered by W values) threaded through the service queue, which has a single entry for any requested page of the database. Maintaining these sorted lists is fairly inexpensive since they only need to be updated when a new request arrives at the server[5]. These two sorted lists are used by a pruning technique in order to avoid an exhaustive search of the service queue to find the maximal $R \times W$ value. This technique is depicted in Figure 6

The search starts with the pages at the top of the R list. The corresponding W value for that page is then used to compute a limit for possible W values. That is, after reading the top page in the R list, it is known that the maximum *RxW*-valued page cannot have a W value below this limit. Next, the entry for the page at the top of the W list is accessed and used to place a limit on the R value. The algorithm alternates between the two queues and stops when the limit is reached on one of them. This technique prunes the search space while

[5] In contrast, for LWF the ordering can change over time, even in the absence of new requests.

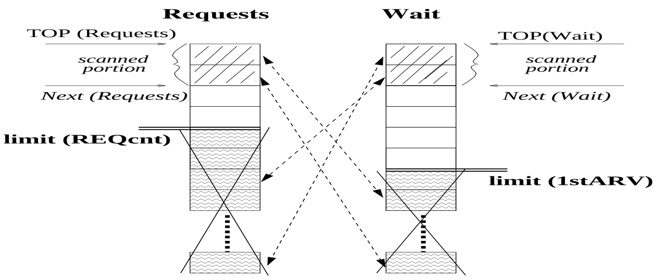

Fig. 6. Pruning the Search Space

still guaranteeing that the search will return the page with the maximum *RxW* value.

In our experiments [Akso98], the pruning technique was shown to indeed be effective – reducing the number of entries searched by 72%. While such a substantial savings is helpful, it is probably not sufficient to keep the scheduling overhead from ultimately becoming a limiting factor as the system is scaled to the huge applications that will be enabled by the national and global broadcasting systems currently being deployed.

In order to achieve even greater reductions in the search space we developed an approximation-based version of the algorithm. By varying a single parameter α, this algorithm can be tuned from having the same behavior as the *RxW* algorithm described so far, to being a *constant time* approach. The approximate algorithm selects the *first* page it encounters whose *RxW* value is greater than or equal to $\alpha \times$ *threshold*, where *threshold* is the running average of the *RxW* value of the last page that was broadcast and the *threshold* at that time.

The setting of α determines the performance tradeoffs between average waiting time, worst case waiting time, and scheduling overhead. The smaller the value of the parameter, the fewer entries are likely to be scanned. At an extreme value of 0, the algorithm simply compares the top entry from both the *R* list and the *W* list and chooses the one with the highest *RxW* value. In this case, the complexity of making a scheduling decision is reduced to *O(1)*, ensuring that broadcast *scheduling* will not become a bottleneck regardless of the broadcast bandwidth, database size, or workload intensity. We demonstrated the performance, scalability, and robustness of the different *RxW* variants through an extensive set of performance experiments described in [Akso98].

4.2 Topic 2: Learning User Profiles

User profiles, which encode the data needs and interests of users, are key components of push-based systems. From the user's viewpoint, a profile provides a means of *passively* retrieving relevant information. A user can submit a profile to a push-based system once, and then continuously receive data that are (supposedly) relevant to him or her in a timely fashion without the need for submitting the same query over and over again. This automatic flow of relevant information helps the user keep pace with the ever-increasing rate of information generation. From the system point of view, profiles fulfill a role similar to that of queries in database or information retrieval systems. In fact, profiles are a form of continuously executing query. In a large publish subscribe system, the storage and access of user profiles can be be resource-intensive. Additionally, given the fact that user interests are changing over time, the profiles must be updated accordingly to reflect up to date information needs.

We have developed an algorithm called *Multi-Modal* (MM), for incrementally constructing and maintaining user profiles for filtering text-based data items [Ceti98]. MM can be tuned to tradeoff effectiveness (i.e., accuracy of the filtered data items), and efficiency of profile management. The algorithm receives relevance feedback information from the users about the documents that they have seen (i.e., a binary indication of whether or not the document was considered useful), and uses this information to improve the current profile. One important aspect of MM is that it represents a user profile as multiple keyword vectors whose size and elements change dynamically based on user feedback.

In fact, it is this *multi-modal* representation of profiles which allows MM to tradeoff effectiveness and efficiency. More specifically, the algorithm can be tuned using a threshold parameter to produce profiles with different sizes. Let us consider the two boundary values of this threshold parameter to illustrate this tradeoff: When the threshold is set to 0, a user profile is represented by a single keyword vector, achieving an extremely low overhead for profile management, but seriously limiting the effectiveness of the profile. At the other extreme, if the threshold is set to 1, we achieve an extremely fine granularity user model, however the profile size equals the number of relevant documents observed by the user, making it impractical to store and maintain profiles. Therefore, it is more desirable to consider intermediate threshold values which will provide an optimal effectiveness/efficiency tradeoff for a given application.

We evaluated the utility of MM by experimentally investigating its ability to categorize pages from the World Wide Web. We used non-interpolated average precision as our primary effectiveness metric and focused on the profile size for quantifying the efficiency of our approach. We demonstrated that we can achieve significantly higher precision values with modest increase in profile sizes. Additionally, we were able to achieve precision values with small profiles that were comparable to, or in some cases even better than those obtained with maximum-sized profiles. The details of the algorithm, experimental setting, and the results are discussed in [Ceti98].

5 Summary

The increasing ability to interconnect computers through internetworking, mobile and wireless networks, and high-bandwidth content delivery to the home, has resulted in a proliferation of dissemination-oriented applications. 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 (DBIS) that integrates many different data delivery mechanisms and described some of the unique aspects of such systems. We described our initial prototype of a DBIS Toolkit, which provides a platform for experimenting with different implementations of the DBIS Components. Finally we described our work on two of the many research issues that arise in the design of DBIS architectures.

Data Dissemination and data broadcasting are very fertile and important areas for continued research and development. In fact, we see a migration of data management concerns from the traditional disk-oriented architectures of existing database systems, to the more general notion of *Network Data Management*, in which the movement of data throughout a complex and heterogeneous distributed environment is of paramount concern. Our ongoing research efforts are aimed at better understanding the challenges and tradeoffs that arise in the development of such systems.

References

[Acha95a] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conf.*, 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.

[Acha97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Broadcast Data", *Proc. ACM SIGMOD Conf.*, Tucson, AZ, May, 1997.

[Akso98] D. Aksoy, M. Franklin "Scheduling for Large-Scale On-Demand Data Broadcasting" *IEEE INFOCOM '98*, San Francisco, March, 1998.

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

[Ceti98] U. Cetintemel, M. Franklin, and C. Giles, "Constructing User Web Access Profiles Incrementally: A Multi-Modal Approach", *In Preparation*, October, 1998.

[Dyke86] H.D. Dykeman, M. Ammar, J.W. Wong, "Scheduling Algorithms for Videotex Systems Under Broadcast Delivery", IEEE International Conference on Communications, Toronto, Canada, 1986.

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", Proc. ACM OOPSLA Conference, Atlanta, October, 1997.

[Fran98] M. Franklin, S. Zdonik. "Data in Your Face: Push Technology in Perspective", *Proc. ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 98)*, Seattle, WA, June, 1998, pp 516-519.

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

[Glan96] D. Glance, "Multicast Support for Data Dissemination in OrbixTalk", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[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.

[Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *Proc. ACM SIGMOD Conf.*, 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*, Ashville, NC, December, 1993.

[Rama98] S. Ramakrishnan, V. Dayal, "The PointCast Network" *Proc. ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 98)*, Seattle, WA, June, 1998, p 520.

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

[Vish94] S. Viswanathan, "Publishing in Wireless and Wireline Environments", *Ph.D Thesis*, Rutgers Univ. Tech. Report, November, 1994.

[Yan95] T. Yan, H. Garcia-Molina, "SIFT – A Tool for Wide-area Information Dissemination", *Proc. 1995 USENIX Technical Conference*, 1995.