A Framework for Scalable Dissemination-Based Systems*

Michael Franklin
University of Maryland
franklin@cs.umd.edu

Stanley Zdonik
Brown University
sbz@cs.brown.edu

Abstract

The dramatic improvements in global interconnectivity due to intranets, extranets, and the Internet has led to an explosion in the number and variety of new data-intensive applications. Along with the proliferation of these new applications have come increased problems of scale. This is demonstrated by frequent delays and service disruptions when accessing networked data sources. Recently, push-based techniques have been proposed as a solution to scalability problems for distributed applications. This paper argues that push indeed has its place, but that it is just one aspect of a much larger design space for distributed information systems. We propose the notion of a Dissemination-Based Information System (DBIS) which integrates a variety of data delivery mechanisms and information broker hierarchies. We discuss the properties of such systems and provide some insight into the architectural imperatives that will influence their design. The DBIS framework can serve as the basis for development of a toolkit for constructing distributed information systems that better match the technology they employ to the characteristics of the applications they are intended to support.

1 Introduction

1.1 The World-Wide Wait

The scenario is all too familiar — a major event, such as a national election, is underway and the latest, up-to-the minute results are being posted on the Web. You want to monitor the results for the important national races and for the races in your state, so you fire up your trusty web browser, point it at the election result web site and wait, and wait, and wait.... What's the problem? It could be any number of technical glitches: a congested network, an overloaded server, or even a crashed server. In a larger sense, however, the problem is one of scalability; the system cannot keep up with the heavy load caused by the (transient) surge in activity that occurs in such situations.

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. An election result server, such as that of the preceding scenario, is an example of a *data dissemination*-oriented application. Data dissemination involves the delivery of data from one or more *sources* to a large set of *consumers*. Many dissemination-oriented applications have data access characteristics that differ significantly from the traditional notion of client-server applications as embodied in navigational web browsing technology. For example, the election result server has the following characteristics: 1) There is a huge population of users (potentially many millions) who want to access the data; 2) There is a tremendous degree of overlap among the interests of the user population; 3) Users who are following the event closely are interested only in new data and changes to the existing data; and, 4) The amount of data that must be sent to most users is fairly small. When looking at these characteristics, it becomes clear that the request-response (i.e., RPC), unicast (i.e., point-to-point) method of data delivery used by HTTP is the wrong approach for this 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

*This work has been partially supported by the NSF under grant IRI-9501353, by Rome Labs Agreement Number F30602-97-2-0241 under ARPA order number F078, by an IBM Cooperative Graduate Fellowship, and by research funding and equipment from Intel Corporation.

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 this example application, where the desires 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.

1.2 Is "Push" the Answer?

The above scenario is well-known to web users and, not surprisingly, an increasing number of products are being introduced to address it. A number of these products have received tremendous media attention lately because they are based on a technology called data *Push*. Using data push, the transmission of data to users is initiated without requiring the users to explicitly request it. Examples of systems that employ some form of push technology include Pointcast, Marimba, BackWeb, and AirMedia. Push has also been added to recent versions of the major Web browsers, and the battle for data push standards is well underway.

Systems that are truly implemented with data push can indeed solve some of the scalability problems attributed above to request-response. Since users do not have to poll servers for new and updated data, the number of client requests that must be handled by a server can be reduced dramatically. Simply changing from a client "Pull" model to a push model, however, does not solve all the problems for an application such as the election result server. In particular, performing push to millions of clients using a unicast communication protocol does little to address network bandwidth problems and still requires the server to perform substantial work for each client it is serving. Compounding the confusion is the fact that many systems that provide a "push" interface to users are actually implemented using a programmed polling mechanism. These systems simply save the user from having to click, but do nothing to solve the scalability problems caused by the request-response approach.

The election result server is an example of just one type of dissemination-oriented application. Other examples include news and entertainment delivery, software distribution, traffic information systems, and navigational web browsing. These applications differ widely in the characteristics of the data involved (e.g., size, consistency constraints, etc.), access patterns, and communication channel properties (e.g., symmetric vs. asymmetric, continuously or intermittently connected, etc.). No one data delivery mechanism can provide adequate support for the wide variety of such applications.

To address this need, we are developing a general framework for describing and ultimately constructing Dissemination-Based Information Systems (DBIS). In this framework, push vs. pull is a choice along just one of several dimensions of the design space for data delivery mechanisms. In this paper, we outline a number of data delivery mechanisms and investigate the tradeoffs among them. The goal is to develop a flexible architecture that is capable of supporting 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.

1.3 Overview of the Approach

We view an integrated DBIS as a distributed system in which the links between the computing elements vary in character: from standard pull-based unicast connections to periodic push over a broadcast channel. 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 its data from a broadcast link as opposed to a request-response protocol should make no difference to clients of that provider.

In our approach, we distinguish between three types of nodes: (1) *data sources* provide the base data for the application; (2) *clients* consume this information; and (3) *information brokers* add value to information and redistribute it. By creating hierarchies of these nodes connected by various data delivery mechanisms, the information flow can be tailored to the needs of many different applications.

We aim to provide a *toolkit* of architectural components that can be used to construct a DBIS. A builder of an information resource would make use 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 remainder of the paper we outline our current ideas on the development of such a toolkit. Section 2 describes several options for data delivery mechanisms (i.e., the "links") and discussesthe tradeoffs among them. Section 3 addresses the various types of nodes in a DBIS. Section 4 uses the DBIS model to describe several existing dissemination-oriented systems. Section 5 outlines issues in the development of a DBIS toolkit. Section 6 lists related work. Finally, Section 7 presents our conclusions.

2 Options for Data Delivery

As stated in the Introduction, a key aspect of the DBIS framework is that it supports a wide variety of links for data delivery between sources and clients. Support for different styles of data delivery allows a DBIS to be optimized for various server, client, network, data, and application properties.

2.1 Three Characteristics

We identify three main characteristics that can be used to compare 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.

2.1.1 Client Pull vs. Server Push

The first distinction we make among data delivery styles is that of "push vs. pull". 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, 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.

2.1.2 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.[1] 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.

2.1.3 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*.

2.2 Classification of Delivery Mechanisms

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

[1] 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.

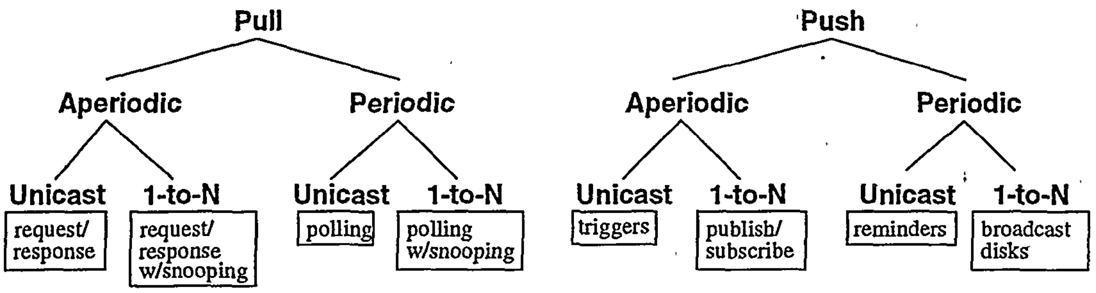

Figure 1: Data Delivery Options

Request/Response - 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.

Polling - 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.

Publish/Subscribe - Publish/subscribe protocols are becoming a popular way to disseminate information in a network [Oki93, Yan95, Glan96]. 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. Such protocols are typically performed in a 1-to-N fashion, but a similar protocol can be used over a unicast channel, as is done for triggers in active database systems.

Broadcast Disks - Periodic push has been used for data dissemination in many systems such as TeleText [Amma85, Wong88], DataCycle [Herm87, Bowe92], Broadcast Disks [Acha95a, Acha95b] and mobile databases [Imie94a]. Clients needing access to a data item that is pushed periodically can wait until the item appears. As with aperiodic push, periodic push can also be used with both unicast and 1-to-N channels, but we believe that 1-to-N is likely to be much more prevalent.

2.3 Some Example Tradeoffs

As can be seen from the preceding discussion, the design space for data delivery mechanisms is quite large. Choosing the proper mechanism (or combination of them) to use for a given link requires an understanding of the tradeoffs among them. In a recent paper, we studied one such set of tradeoffs; namely, those between broadcasting data using periodic push (Broadcast Disks) and aperiodic pull (request-response with snooping) [Acha97]. Here, we briefly discuss some observations from that study.

The tradeoffs between push and pull in general revolve around the costs of initiating the transfer of data. A pull-based approach requires the use of a backchannel for 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 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 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. As mentioned above, *Publish/subscribe* protocols are one popular mechanism for providing such profiles.

In [Acha97] we studied a hybrid push/pull broadcast system. In this system, a broadcast server is responsible for allocating a fixed broadcast bandwidth between data items (pages) that are broadcast according to a fixed schedule (i.e., periodic push) and pages that are broadcast in response to client requests sent over a backchannel (i.e., aperiodic pull). The fundamental performance tradeoff between these two approaches can be seen in in Figure 2, which shows results from [Acha97][2]. The x-axis in the figure models the number of clients (all having identical access rates and distributions) that are accessing data from the broadcast. Thus, at a value of 250, the broadcast is serving 25 times as many clients than at a value of 10. The y-axis indicates the average number of items that a client must watch go by on the broadcast before the item it wants appears.

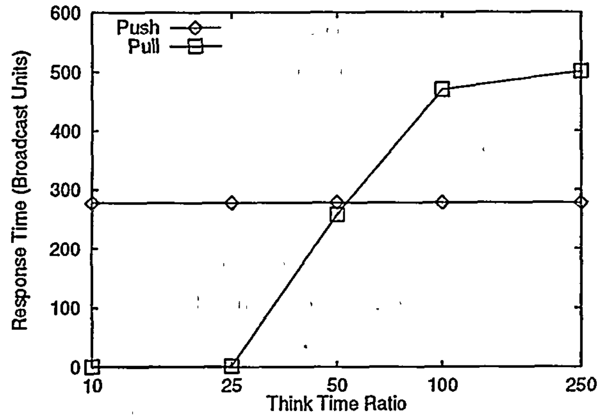

Figure 2: Push vs. Pull for Broadcast

The flat line in the figure (marked by diamonds) indicates the performance of a pure push approach, in which all data is broadcast repeatedly with no requests sent by the clients. This figure was generated using a skewed (Zipfian) access pattern over 1000 items. The broadcast schedule used by the push approach was tailored to support a skewed access pattern through the use of Broadcast Disks which allow the frequency of broadcast for an item to be based on that item's popularity [Acha95a, Acha95b]. As can be seen in the figure, the performance of pure push is *independent* of the number of clients listening to the broadcast here. This is a fundamental property of data broadcast using periodic push — if there is a large overlap in the interests of clients, it provides tremendous scalability in terms of client population.

The other curve in the figure (marked by boxes) shows the performance of a pull-based approach, in which clients submit requests to the server via the backchannel, and the server broadcasts the requested pages in FIFO order.[3] As can be seen in the figure, the pull-based approach exhibits an S-shaped behavior — it provides extremely fast response time for a lightly loaded server, but as the server becomes loaded, its performance degrades, until it ultimately stabilizes (in this case, at a value of 500 items, or half the size of the database being broadcast here).

The behavior of aperiodic pull in this case can be explained as follows. With a lightly loaded system, the server is typically idle so it can respond immediately when a request is received. As the load increases, however, the server saturates and becomes less responsive. Compared to periodic push, it is clear that aperiodic pull demonstrates less scalability in this case. It is, however, important to note that aperiodic pull over a *unicast* channel would be far less scalable — wait time would increase in an unbounded fashion as the server approached saturation. In contrast using broadcast, the performance of aperiodic pull eventually flattens out in this case, because of the overlap in the interests of the client population. Once the server reaches the state where all data items are in the FIFO queue, additional clients receive all of their data by simply "snooping" on the broadcast. In this case the performance of aperiodic pull at saturation is worse than that of periodic push, because the broadcast schedule generated by the FIFO discipline is less well suited to the access pattern than the pre-computed schedule used by periodic push. As discussed in [Acha97], the problems of pull can be exacerbated if the server drops client requests when it becomes overloaded.

The tradeoffs described above give an indication of the kinds of concerns that must be balanced when choosing the proper data delivery mechanism for a given situation. Another set of options arises in the organization of the nodes for a DBIS, as described in the following section.

3 Design Options for Nodes

While the discussion so far has focused on the ways in which data is communicated between computing devices, the nodes in a Dissemination-Based Information System play a crucial role as well: the nodes provide the glue that pastes various data distribution schemes together. A DBIS toolkit should contain classes that model some of the basic features

[2]We briefly summarize these results here, interested readers are referred to [Acha97]for more details

[3]Because a single broadcast of an item satisfies all clients waiting for that item, we do not enqueue a request for an item that is already in the FIFO queue.

of nodes. This section outlines some of those features.

3.1 Classification

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.

Information brokers perform many important functions in our architecture. While the previous discussion focused primarily on different modes of data delivery, the brokers provide the glue that binds these modes together. It is typically the expected usage patterns of the brokers that will drive the selection of which mode of delivery to use. 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.

As we move upstream in the data delivery chain, brokers look like data sources to their clients. Receivers of information cannot detect the details of interconnections any further upstream than their immediate predecessor. This principle of *network transparency* allows data delivery mechanisms to change without having global impact. Suppose that node B is pulling data values from node A on demand. Further, suppose that node C is listening to a cyclic broadcast from node B which includes values that B has pulled from A. Node C will not have to change its data gathering strategy if A begins to push values to B; changes in links are negotiated purely between the two nodes involved.

Of course, nothing is ever simple. In some cases, brokers can also be sources by maintaining their own databases. In this case, the *hybrid* broker can add data of its own to what it receives from its upstream counterparts. The principle of network transparency also protects clients from having to depend on this situation. A data source, be it a pure source, a broker, or a hybrid source, only guarantees that it can provide specific data — independently of where it comes from.

3.2 Caching

While nodes can perform many functions, the most ubiquitous data management facility is caching. Unlike caching in client-server systems, the path from data sources to a client can be of length greater than two. Thus, items might be cached at any of many points along the data path in the network. Thus, caching in this context resembles the kind of proxy caching that one might find in a wide-area network (e.g., the Internet).

While the problems here are very similar to those of any proxy caching scheme, the broad view of data movement available in a DBIS makes the potential solutions much richer. For example, if there are copies of a particular data item in multiple caches, there will always be an issue of how those copies are refreshed when the primary copy is updated. One solution is to send invalidations to each client cache manager. An invalidation message results in the purge of the item from the cache. Alternatively, the new value could be propagated to the client cache managers. For typical client/server systems, invalidation is usually preferable. However, in our broadcast disk studies [Acha96b] we showed that for periodic broadcast, performance can often be improved using propagation.

The decision about how current to keep the cached copies is the same as in other caching mechanisms. Once that has been decided, the means by which it is achieved can vary. In a DBIS, we could propagate (i.e., push) the changes to the clients or wait for the client to request the item again (i.e., pull). In the latter case, if a cache manager cares about keeping items very current, it will have to poll the state of the object often. It is interesting to note that if the data delivery mechanism in a DBIS changes, the means by which updates are propagated (or not) may also need to change.

Deciding which object to evict from the cache when a new candidate arrives is another issue that must be addressed by any cache manager. Many systems use some form of LRU for this purpose. We have shown in previous work [Acha95a] that for some styles of data delivery (e.g., broadcast disks), LRU is not the most effective choice. For cyclic data delivery, in which different items can have different arrival frequencies, a cost-based caching scheme performs significantly better.

In a DBIS, the modes of data delivery might change. In such an environment, the caching policy could change to match the prevailing conditions. We will need heuristics for deciding the appropriate caching policies for a particular configuration of distributed components. As an example, if node B initially pulls data from node A, B might reasonably use LRU as its caching policy. When A creates a broadcast disk which is read by B, B might then change its caching policy to a cost based scheme similar to the one that we propose in [Acha95a].

3.3 Value-Added Nodes

Some nodes may also add value to data as it passes through, by performing specific computations on that data. The computations can be simple or complex, or they can act on single values or sets of values. Other nodes may simply pass values on to other nodes.

As an example, suppose node A pushes stock prices for Fortune 500 companies that are picked up by node B. Node B keeps a database of previous stock prices and when a new price for the day is picked up from node A, it calculates the difference between the most current price and yesterday's close, and pushes this value out to yet another community. Node B is a push-based, value-added server. Of course, it need not be based on push. Other clients could pull stock deviations from B as well.

Another kind of value-added service that a node can perform is *merging* of values from multiple sources. Merging can occur in several ways. The first involves multiple sources that maintain similar information. The merge node can make the most reliable or most current version of a value available. Alternatively, multiple sources may maintain a set of values which the merge node combines to a single value. An example of this might involve nodes that maintain demographic information for towns including their current population. Another node may read these values and consolidate them into a single population figure for the state.

Nodes can also perform the service of *filtering*. A filtering node will receive a large volume of data from another node, only some fraction of which it makes available to its clients. For example, a node could receive all stock prices from the NYSE and provide information about only the Fortune 500 stocks to its clients.

3.4 Recoverable Nodes

Often it will be useful to make guarantees about the reliability of some node. Thus, nodes that implement some degree of recoverability will be a useful component in a DBIS. Consider a node that must guarantee the delivery of the latest version of IBM's stock price. Such a node must not lose its information in the event of a failure. That is, if the information was received, then the node must be able to guarantee that it will eventually be made available to its clients.

Of course, having recoverable brokers is not enough on its own to guarantee that nodes will not miss disseminated information while they are down. In order to address this issue, a scheme like reliable multicasting would have to be used. Reliable multicasting will eventually deliver all messages, but it cannot make real-time guarantees about when an object will arrive.

3.5 The Burden of Push

As mentioned in Section 2.3, any node that provides a push service must do so on the basis of some knowledge of the access patterns of its client base. If the node pushes data that few clients care about, then bandwidth is wasted. The trick is to broadcast items that are of interest to a large segment of the user community. This, of course, is only possible if there is high commonality of interest for at least some data items.

In order to optimize its push schedule, the server must rely on profiles of user needs. Profiles could be learned by servers if clients provide feedback about the effectiveness of the push schedule. Alternatively, a client could communicate a profile to the server at appropriate times, such as when it begins to listen to the push, at regularly scheduled intervals, or whenever the client notices that the current schedule deviates significantly from what it would like to see.

What would such a profile look like? A profile is very much like a continuously executing query [Terr92]. In other words, it is a predicate that indicates the items that the client would like to see. It is continuously executing because the server will push items as long as there are currently valid profiles that match the items.

Profiles can be interpreted to mean that whenever a new item is added to the database that matches a profile, the owner of that profile will receive the new data. On the other hand, the profile could be treated more as a hint to the server indicating interest with no requirement on the server's part to send matching items. In this case, the server may choose to conserve bandwidth and not send a matching item in order to best serve the client community as a whole.

4 Systems Viewed as DBIS

In this section, we describe some existing systems using the concepts of our DBIS framework.

4.1 Pointcast

Pointcast is a dissemination service that has attracted a large population of users. It obtains profiles from users that describe their interests, and then uses these profiles to assemble and update customized "newspapers" from a database of current stories.

The Pointcast system has been touted as one of the first push-based systems. This is not exactly true. Other systems such as Teletex [Amma85], BCS at MIT [Giff90], and Datacycle [Herm87] used push long before Pointcast. However, Pointcast was one of the first push-based systems to achieve wide-spread use. It is instructive, therefore, to see exactly how push is used in Pointcast 1.0 [4].

From the point of view of a DBIS, the use of push within Pointcast is extremely limited. In fact, in terms of the network architecture, push is non-existent; that is, the flow of requests and responses within the global architecture is pull-based. The Pointcast client on a user's workstation generates requests for news stories that match the user's profile. For example, if the user indicates an interest in the computer industry, the Pointcast client polls the Pointcast server for news stories with the keyword "computer industry" whenever the Pointcast screen saver is enabled. All of these requests can generate lots of network traffic.

So, where's the push? If we look at Figure 3, we see that there are essentially two processes in the client machine. One of these processes is responsible for *pulling* the latest news stories down to the user's machine, and the other is responsible for displaying these stories on the user's screen. The push really occurs between these two components. When the pull-based story acquisition module gets a new story, it pushes it to the screen manager. From the user's point of view, this *is* push because things are happening to the screen without any intervention. The use of push as a technique for managing heavy network loads, however, is not part of the design.

[4] Hereafter, referred to as Pointcast.

4.2 Broadcast Disks

Our own work on broadcast disks is based on a model of data delivery that is virtually the direct opposite of that described above for Pointcast (see Figure 4).

In our model, an application process on the client workstation behaves exactly as it would in a traditional pull-based environment. It generates pull requests as it needs data and blocks until that data is received.

The server, however, proactively sends data to the client community in advance of any request (i.e., push). A process on the client listens to the broadcast stream and picks up data items for which the application might be waiting. Thus, the places where pushes and pulls happen have been inverted over the Pointcast case.

It should be noted that in the broadcast disk case, the push is periodic and is scheduled by the server. In the Pointcast case, the pull is also periodic, but the interval is set by the user.

4.3 SIFT

The SIFT [Yan95] system was developed at Stanford University as a way to disseminate documents to a user community. SIFT combines data management ideas from information retrieval with a *publish/subscribe* model for dissemination. We describe the way the publish/subscribe model works in terms of our DBIS architecture.

Looking at Figure 5, we see three active components: the document source, the SIFT server, and a SIFT client (one of potentially many). The connection between the document source and the SIFT server (on the left side of the figure) is push-based, unicast, and aperiodic. The document source could alternatively deliver new documents through a 1-to-n broadcast medium, such as a satellite feed, if there were multiple interested recipients (SIFT servers or otherwise). A backchannel (not shown in the figure), is used only to set up the initial connection. Thereafter, the document source forwards all new documents to the SIFT server. There is no filtering that happens on this link. We could think of the profile held at the document source for the SIFT server as

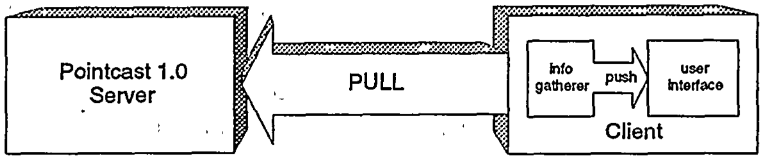

Figure 3: Pointcast 1.0

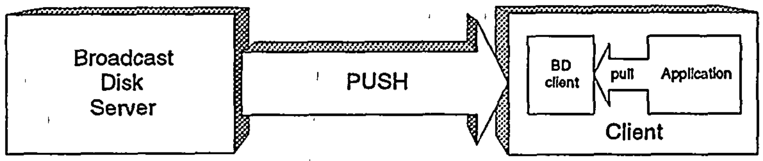

Figure 4: Broadcast Disks

being *send everything.*

The connection between the SIFT server and a given SIFT client (shown on the right side of the figure) is also push-based, unicast, and aperiodic. In this case, though, the client profile that is held at the SIFT server is customized for each client. It consists of a series of keywords and weights that describe documents of interest to that client. The SIFT server provides novel technology for indexing client profiles. Such an index is used for matching profiles against newly arriving documents. This indexing technique allows the server to accommodate a large client population with reasonable performance. The original SIFT prototype disseminated entire articles to clients. With the existence of the web, it becomes possible to send short article descriptions plus the corresponding URLs to conserve bandwidth.

It should be noted that clients get exactly what their profiles specify and nothing more. This is in contrast to a 1-to-n (broadcast) style of delivery in which all clients see the same information stream. It is the server's responsibility to optimize this stream to suit the needs of the largest number of users. It is unlikely that such a stream will be optimal for any one user.

5 Putting it All Together

In the preceding discussion, we described a vision of how distributed information systems should be built in the future. Our framework focused on techniques for delivering data in wide-area network settings in which nodes and links reflect extreme variation in their operating parameters. By adjusting the delivery mechanism to match these characteristics, we believe that we can achieve high performance and scalability without the need to invest in additional hardware. In this section, we briefly discuss our approach to this problem and outline some of the open research questions.

5.1 Toolkit Approach

We intend to realize our solutions to the problems of designing a DBIS through a toolkit that provides the proper components from which any DBIS could be built. This toolkit can be thought of as a set of object classes that support concepts such as *network connections* and *local caches.*

A key part of the toolkit will be a set of classes to allow distributed nodes to negotiate in order to establish a proper connection. This is required at several levels. At the highest level, the nodes must agree on how data is to be transferred. A client node that is relying on data from some server must know whether that server will be using push or accepting requests. There are also handshaking protocols that must occur at lower levels. For example, if a push-based broadcast connection is to be established in an Ethernet, the nodes must agree on which Ethernet address will be used for that broadcast. The parties must also agree on the parameters that will be used to configure that broadcast. For example, if it is a broadcast disk, the frequency of broadcast of each item is of interest to the clients.

The usefulness of a toolkit will rely on the precise definition of the DBIS classes. These classes must be of general utility. Also, as indicated in Section 2.3, the definition of

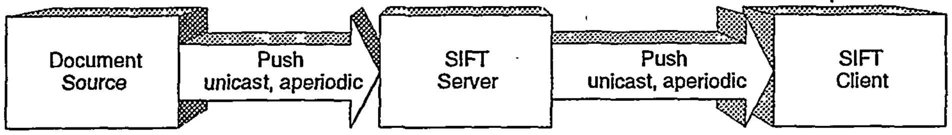

Figure 5: High-Level SIFT Architecture

these classes must be based on a substantial body of experimental results that help to delineate the sometimes subtle tradeoffs.

5.2 Dynamic Reconfiguration

A network can be characterized by prevailing loads on the nodes and the connections. This characterization changes rapidly, and a responsive DBIS must be able to adapt to these changes. Thus, our vision of a fully functional DBIS includes facilities to support the dynamic reconfiguration of the data delivery mechanisms.

A key element of a reconfiguration facility is a statistics gathering component that collects the right performance numbers and that can intelligently select among the available delivery options. This is not a simple matter. Our previous experiments in the area of broadcast disks has shown that the design space here is very complex with many places in which intuitions from more traditional distributed system design often produces poor results.

5.3 Some Design Issues

In addition to the plumbing issues that we have discussed so far, there are some higher-level issues that must be addressed in developing an integrated DBIS. In the following, we briefly outline some of these issues:

- *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 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 communication 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 Related Work

Work on distributed object computing has generated many important standards and systems. CORBA [OMG91] and DCE [OSF94], for example, are two important approaches to system interoperability. This work is not incompatible with the notion of a DBIS. A DBIS can be thought of as infrastructure for such object-oriented middleware.

There is much previous work that relates to the architectural issues of a DBIS. The brief discussion that follows samples some of the work that is most related to the issues presented in this paper.

The management of data in distributed settings has a long history. The preponderance of previous work assumes that data is requested when needed (i.e., pull) and that servers respond to these requests in an orderly fashion. Some of this work has occurred in a client/server database setting [Fran96a] while other work has been done in the distributed file system context [Levy90]. There has been a lot of work on caching in these environments, much of which has focused on the maintenance of cache consistency in the face of updates.

More recently, there has been work on data management issues for wireless environments [Katz94]. Some of work in this area has focused on satellite-based systems [Dao96, Dire96] in which the downstream bandwidth is quite high.

The idea of the publish/subscribe model as a dissemination mechanism has been used in many contexts including SIFT [Yan95] and the Information Bus[Oki93].

There has also been work on broadcasting in Teletex systems [Amma85, Wong88]. [Wong88] presents an overview of some of the analytical studies on one-way, two-way and hybrid broadcast in this framework.

The Datacycle Project [Bowe92, Herm87] at Bellcore investigated the notion of using a repetitive broadcast medium for database storage and query processing. An early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast news articles and information over an FM channel to clients with personal computers specially equipped with radio receivers. Both Datacycle and BCIS used a flat broadcast (i.e., all items have the same frequency). The mobility group at Rutgers [Imie94a, Imie94b] has done significant work on data broadcasting in mobile environments. A main focus of their work has been to investigate novel ways of indexing in order to reduce power consumption at the mobile clients. Some recent applications of dissemination-based systems include information dissemination on the Internet [Yan95, Best96], and Advanced Traveler Information Systems [Shek96].

Our work on Broadcast Disks differs from these in that we consider multi-level disks and their relationship to cache management. In [Acha95a], we proposed an algorithm to generate Broadcast Disk programs and demonstrated the need for cost-based caching in this environment. Recently, [Baru96] gave an algorithm to determine the parameters controlling a broadcast program. In [Acha96a], we showed how opportunistic prefetching by the client can significantly improve performance over demand-driven caching. More recently, in [Acha96b], we studied the influence of volatile data on client performance and showed that the Broadcast Disk environment can be made very robust in the presence of updates. In [Acha97], we explored the tradeoff between cyclic broadcast and pull.

7 Conclusions

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. A key attribute of many such applications is their huge scale. 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 types of information brokers. We described some of the unique aspects of such systems and discussed how several existing dissemination-based architectures fit in to the DBIS model.

The ideas presented in this paper have grown out of our previous work on the Broadcast Disks paradigm for data delivery. A key lesson from that work was the importance of applying a data management perspective to distributed systems architecture issues. We are currently completing a prototype that combines the push-based Broadcast Disks with a pull-based broadcast model. We view that prototype as the first step in the development of a generic DBIS toolkit that will support the creation of a variety of large-scale dissemination-based applications across several different communication media.

Acknowledgments

We would like to thank Swarup Acharya for his contributions to these ideas through the development of the Broadcast Disks paradigm and Demet Aksoy who has provided us with important insights into the properties of broadcast scheduling.

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.

[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 Conf.*, Bombay, India, September, 1996.

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

[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.

[Best96] A. Bestavros, C. Cunha, "Server-initiated Document Dissemination for the WWW", *IEEE Data Engineering Bulletin*, 19(3), September, 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.

[Care91] M. Carey, M. Franklin, M. Livny, E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures", *Proc. ACM SIGMOD Conf.*, Denver, June, 1991.

[Dao96] S. Dao, B. Perry, "Information Dissemination in Hybrid Satellite/Terrestrial Networks", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Dire96] Hughes Network Systems, DirecPC Home Page, `http://www.direcpc.com/`, Oct, 1996.

[Erik94] H. Erikson, "MBONE: The Multicast Backbone", *CACM*, 37(8), August, 1994.

[Fran96a] M. Franklin, *Client Data Caching: A Foundation for High Performance Object Database Systems*, Kluwer Academic Publishers, Boston, MA, February, 1996.

[Fran96b] M. Franklin, S. Zdonik, "Dissemination-Based Information Systems", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[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.

[Imie94a] T. Imielinski, B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management", *CACM*, 37(10), October, 1994.

[Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *Proc. ACM SIGMOD Conf.*, Minneapolis, MN, May, 1994.

[Katz94] R. Katz, "Adaption and Mobility in Wireless Information Systems", *IEEE Personnal Comm.*, 1st Quarter, 1994.

[Levy90] Levy, E., Silbershatz, A., "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, 22(4), December, 1990.

[OMG91] Object Management Group and X/Open, "Common Object Request Broker: Architecture and Specification", *Reference OMG 91.12.1*, 1991.

[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.

[OSF94] Open Software Foundation, "Introduction to OSF DCE", *Prentice Hall*, Englewood Cliffs, NJ, 1994.

[Shek96] S. Shekhar, A. Fetterer, D. Liu, "Genesis: An Approach to Data Dissemination in Advanced Traveller Information Systems", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Terr92] D. Terry, D. Goldberg, D. Nichols, "Continuous Queries Over Append-Only Databases", *Proc. ACM SIGMOD Conf.*, San Diego, CA, June, 1992.

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

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