Scalable Dissemination: What's Hot and What's Not

Jonathan Beaver* Nicholas Morsillo* Kirk Pruhs* Panos K. Chrysanthis*
Vincenzo Liberatore†

ABSTRACT

A major problem in web database applications and on the Internet in general is the scalable delivery of data. One proposed solution for this problem is a hybrid system that uses multicast push to scalably deliver the most popular data, and reserves traditional unicast pull for delivery of less popular data. However, such a hybrid scheme introduces a variety of data management problems at the server. In this paper we examine three of these problems: the push popularity problem, the document classification problem and the bandwidth division problem. The push popularity problem is to estimate the popularity of the documents in the web site. The document classification problem is to determine which documents should be pushed and which documents must be pulled. The bandwidth division problem is to determine how much of the server bandwidth to devote to pushed documents and how much of the server bandwidth should be reserved for pulled documents. We propose simple and elegant solutions for these problems. We report on experiments with our system that validate our algorithms.

1. INTRODUCTION

The Web has brought massive amounts of information to everyone's fingertips, changing forever the way we learn the news, perform research, do business, and deal with disasters. At the same time, widespread interconnectivity has resulted in Internet hot spots and flash crowds, and poses a pressing scalability problem. The scalability of data dissemination is particularly important exactly at the time when scalable data delivery is most important. Examples in the popular press include: news about the terrorist attacks at `msnbc.com` during 9/11/2001, virus patches from `mcafee.com` during the recent Slammer virus, and weather reports at the Federal Emergency Management Agency `fema.gov` during hurricane Isabel around September 18, 2003.

*Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. {`beaver, nwm1, kirk, panos,`}`@cs.pitt.edu` Supported in part by NSF grant CCR-0098752, NSF grant ANI-0123705, and NSF grant ANI-0325353
†Division of Computer Science, Case Western Reserve University, 10900 Euclid Ave., Cleveland, Ohio 44106-7071. `vincenzo.liberatore@case.edu` Supported in part by NSF grant ANI-0123929.

Scalable data delivery can be ensured with *multicast push*, which employs point-to-multipoint communication (*multicast*) and sends documents from the server to clients in the absence of explicit client requests (*push*) [1, 2, 3, 4, 12, 25, 26, 27]. Multicast push is scalable in that the addition of new clients does not change the server workload and the client-perceived response time. Multicast push can be combined with traditional unicast pull in a *hybrid data dissemination* scheme [3, 7, 25]. In the hybrid method, the document set is partitioned into two groups: the multicast push documents and the unicast pull documents. The former are cyclically and repeatedly transmitted on the multicast push channel, the latter are delivered on the unicast pull channel upon client requests.

Hybrid data dissemination can be evaluated along various performance metrics, and much work has focused on the average server-side delay before a document is transmitted (e.g., [25]). Client-side latency can be minimized by assigning multicast push to deliver the most popular (*hot*) data and unicast pull to deliver less popular (*cool*) data [25]. The resulting hybrid scheme strives for the scalability of multicast push and avoids clogging a multicast channel with cool data items [1, 13]. However, the hybrid scheme introduces three inter-related data management problems at the server, and the primary contribution of this paper is to propose an integrated solution for these problems.

In the hybrid scheme, the server must dynamically assign each documents either to the unicast pull channel or to the multicast push channel (*document classification*) [25]. Furthermore, the server must also partition dynamically its bandwidth between unicast pull and multicast push (*bandwidth division*). Document classification and bandwidth division are inter-related issues because a given bandwidth division determines the performance of a document classification choice and, conversely, a given document classification determines a bandwidth split that optimizes performance. In turn, both document classification and bandwidth division depend on the popularity of data items because download latency is smaller when hot items are assigned to multicast push, cool items to unicast pull, and the bandwidth is divided appropriately between the two channels. Therefore, the server must estimate the document popularity (*push popularity* problem). The estimation of document popularity is complicated by the fact that no requests are made by clients for multicast push documents. In particular, if the popularity wanes for a specific document and that document was on the multicast push, the shift in client interests is not reflected in request logs. In turn, the server would not know that it is time to demote a document to the unicast pull channel.

In this paper, we give an integrated algorithm for solving simultaneously and to near-optimality the bandwidth division and document classification problems. Our algorithm is evaluated through emulations on a comprehensive middleware platform for scalable data dissemination [23]. The algorithm exhibited lower average latency than previous schemes. The underlying reason is that if document selection is addressed separately from bandwidth division, a certain bandwidth split can be fixed to a level that is suboptimal for a certain assignment of documents to channels. More generally, the performance trade-offs differ quantitatively and qualitatively under the combined scheme. For example, the assignment led to multicast push latency that is significantly faster than pull latency due to the higher relative popularity of multicast items over unicast documents. Furthermore, the paper also presents and quantifies analytically the accuracy of a simple document popularity algorithm that is based on random sampling.

The remainder of the paper is structured as follows: In section 2 we discuss our proposed algorithms. In Section 3, we report on the experimental validation of our algorithms. In section 4 we survey some work related to our own and section 5 summarizes our observations and contributions.

2. ALGORITHMS AND ANALYSIS

2.1 Document Classification and Bandwidth Division

To understand the algorithm for document classification and bandwidth division, it is first necessary to understand the different nature of average latency for the multicast push and for the unicast pull channel. The average latency for documents on the multicast push channel is roughly linear in the number of documents on this channel. Specifically, the average latency for a document on the multicast push channel is half of the period that it takes to cycle through all the pushed documents. However, the delays for pulled documents are radically different from those of pushed documents. If document i is assigned to unicast pull, a client request for i is queued at the server for transmission. Let S_i be the size of document i. Basic queuing theory tells us that the corresponding queuing delay is either $O(S_i)$ or unbounded, depending on whether the server load is less than 1 or not. Thus, to minimize average latency, the server should require as many documents as possible be pulled, as long as the load for the pulled documents is bounded by a constant less than 1.

Our solution to document classification and bandwidth division is to use an integrated algorithm that minimizes average latency. The starting point is a method suggested by [6] that minimizes the bandwidth B to achieve a target latency L. The known method is not directly applicable to document classification and bandwidth division because our goal, on the contrary, is to minimize the latency L given a fixed amount of available server bandwidth B. However, the previous method will be useful as a component of the final algorithm.

For the purpose of analysis, client requests follow a Poisson process in which document i is requested with probability p_i and the aggregate request rate is λ. The popularity profile p_i and the rate λ can be determined by the push popularity algorithm described in Section 2.2. Algorithm 1 uses a target average latency L. Its goal is, as in [6], to partition the documents between unicast pull and multicast push and to split the system bandwidth so as to minimize the system bandwidth B required to achieve latency L. If document i is assigned to the pull channel, it will use bandwidth $\lambda p_i S_i$. If document i is assigned to the push channel, it will use bandwidth S_i/L, which is also the rate at which the document must be broadcast to give *worst-case* response time L. It was then stated [6] that a document should be pushed if

$$\lambda p_i S_i > \frac{S_i}{L} \ . \tag{1}$$

While Algorithm 1 follows this approach, we also need to make the following modification in (1). The *average* latency of a pushed document is the appropriate measure, rather than the worst-case latency. The reason is that the unicast pull term $\lambda p_i S_i$ in (1) is an average latency and thus the multicast push term also needs to be an average delay for the comparison to be meaningful. In this case, the bandwidth used by this document on the push channel is $S_i/(2L)$, and a document should be pushed if $\lambda p_i S_i > S_i/(2L)$. The resulting subroutine is shown below as Algorithm 1 and its parameters as summarized in Table 1.

Parameter	Description
n	number of documents
λ	observed request rate λ
α	pull over-provisioning factor
B	total available system bandwidth
S	array of document sizes S_i
p	array of document probabilities p_i
ϵ	tolerance factor

Table 1: Parameters for Algorithms 1 and 2.

Algorithm 1 tryLatency

Require: n, λ, L, p as defined in Table 1
Ensure: Returns the number k of items pushed given that average latency of L is required

while max $-$ min > 1 **do**
  $k \leftarrow (\max + \min)/2$
  if $(p_k \lambda L) > 1/2$ **then**
    min $\leftarrow k$
  else
    max $\leftarrow k$
  end if
end while
Return k

The bandwidth for the push channel is now $\sum_{i:p_i>1/(\lambda L)} S_i/(2L)$ and the bandwidth for the unicast requests is $\sum_{i:p_i \le 1/(\lambda L)} \lambda p_i S_i$. In particular, if L increases, while all other variables remain fixed, then more documents are pushed, an observation that will be used to derive the final algorithm.

The second and more substantial modification to the previous argument is due to the mismatch between the objectives of Algorithm 1 and the desired objectives for bandwidth division and document selection. Algorithm 1 minimizes the amount of required bandwidth to achieve a fixed L, whereas our goal is to minimize L given a fixed deployed bandwidth B. In some sense, our problem is the dual of the one that Algorithm 1 solves.

Algorithm 2 solves the bandwidth division and document selection problems, and uses Algorithm 1 as a subroutine. The algorithm employs a parameter $\alpha > 1$ that measures the target level of over-provisioning for the pull channel. More precisely, the actual bandwidth we reserve for pull is α times what an idealized estimate predicts. Queuing theory asserts that $\alpha > 1$ guarantees bounded queuing delays, whereas $\alpha \le 1$ leads to infinite queuing delays. As such, the parameter α can also be thought of as a safety margin for the pull channel. The algorithm also uses a parameter $\epsilon > 0$, which is an arbitrarily small positive number, and finds a solution that has latency within ϵ of the optimum for the given bandwidth and popularities.

Algorithm 2 Bandwidth Division and Document Classification

Require: $n, \lambda, \alpha, B, S, p$, and ϵ as defined in Table 1, and $p_i \geq p_{i+1}$ $(1 \leq i < n)$

Ensure: k is the optimal number of documents on the push channel, pullBW is the optimal pull bandwidth, pushBW is the optimal push bandwidth

1: **for** $i = 1, \ldots, n$ **do**
2: $\quad$ $\text{rspt}_i \leftarrow \text{rspt}_{i-1} + p_i S_i \lambda$
3: $\quad$ $\text{sizeTotal}_i = \text{sizeTotal}_{i-1} + S_i$
4: **end for**
5: $\text{lMax} \leftarrow \text{sizeTotal}_n / B$
6: $\text{lMin} \leftarrow 0$
7: **while** $\text{lMax} - \text{lMin} > \epsilon$ **do**
8: $\quad$ $L \leftarrow (\text{lMax} + \text{lMin})/2$
9: $\quad$ $k \leftarrow \text{tryLatency}(L, p, \lambda, n)$
10: $\quad$ $\text{pullBW} \leftarrow \alpha(\text{rspt}_n - \text{rspt}_k)$
11: $\quad$ $\text{pushBW} \leftarrow B - \text{pullBW}$
12: $\quad$ **if** $\text{pushBW} \geq (\text{sizeTotal}_k/(2L))$ **then**
13: $\quad\quad$ $\text{lMax} \leftarrow L$
14: $\quad$ **else**
15: $\quad\quad$ $\text{lMin} \leftarrow L$
16: $\quad$ **end if**
17: **end while**

Algorithm 1 assumes that documents have been sorted in non-increasing order of popularity, i.e., $p_i \geq p_{i+1}$ $(1 \leq i < n)$. It can be easily seen that if i is pushed, then $j < i$ should be pushed as well. Then, the problem becomes that of finding a value of k such that the multicast push set $\{1, 2, \ldots, k\}$ minimizes the latency L given a certain bandwidth B and pull over-provisioning factor α. The optimal value k^* can be found by trying all possible values of L, computing the document k that achieves L with Algorithm 1, and checking that this value of k satisfies the bandwidth requirements. The pull bandwidth requirement is $\alpha \sum_{i=k+1}^{n} \lambda p_i S_i$, which leaves $\text{pushBW} = B - \alpha \sum_{k+1}^{n} \lambda p_i S_i$ for the push channel, and average latency for the pushed documents of $\sum_{i=1}^{k} S_i / 2\text{pushBW}$. If this computed average latency for the pushed documents is greater than L, then L needs to be increased, otherwise L needs to be decreased.

Algorithm 2 follows this approach but with two optimizations. In the first place, the algorithm performs a binary search over all possible values of L and stops when the interval for L is bounded by the tolerance ϵ. Moreover, the algorithm pre-computes the sums $\sum_{i=1}^{k} \lambda p_i S_i$ and $\sum_{i=1}^{k} S_i$ in the arrays rspt and the sizeTotal respectively (Lines 1–4). The purpose of these computations is to use the totals in the place of the sums in the bandwidth computations. Because of this optimization, the algorithm runs in linear time. The algorithm could also be implemented in logarithmic time using standard augmented binary tree techniques, where logarithmic time per query is taken to update the data structures [8].

2.2 Report Probabilities

Document selection and bandwidth division rely on estimates p of document popularity. The values of p can be estimated by sampling the client population as follows. The server publishes a report probability s_i for each pushed document i. Then, if a client wishes to access document i, it submits an explicit request for that document with probability s_i. In principle, clients would not need to submit any request for push documents, but if they do send requests with probability s_i, the server can use those requests to estimate p_i. At the same time, the report probability s_i should be small enough that server is almost surely not going to be overwhelmed with requests for pushed documents. In particular, we consider the objective of minimizing the maximum relative inaccuracy observed in the estimated popularities of the pushed documents. In this case, we show analytically that each report probability should be set inversely proportional to the predicted access probability for that document.

First, the server calculates the rate β of incoming reports that it can tolerate. Presumably, β is approximately equal to the rate that the server can accept TCP connections minus the rate of connection arrivals for pulled documents. Therefore, the value of β can be estimated from the access probabilities and the current request rate, all scaled down by a safety factor to give the server a little leeway for error. Then, the s_i's have to be set such that $\sum_{i=1}^{k} \lambda p_i s_i \leq \beta$, where documents $1, \ldots k$ are on the push channel. The expected number of reports μ_i that the server can expect to see for i over a unit time period is $\lambda p_i s_i$. Using standard Chernoff bounds, the probability that number of reports is more than $(1+\delta)\mu_i$ is roughly $e^{\frac{-\mu_i \delta^2}{4}}$, and that the probability that number of reports is less than $(1-\delta)\mu_i$ is roughly $e^{\frac{-\mu_i \delta^2}{2}}$. If the goal is to minimize the expected maximum relative inaccuracy of the reports, all of the upper tail bounds should be equal and all of the lower tail bounds should be equal. That is, all μ_i should be equal, or equivalently it should be the case that for all i, $1 \leq i \leq k$, $s_i = \frac{\beta}{\lambda p_i k}$. Hence, each document should have a report percentage inversely proportional to its access probability.

3. EVALUATION

The objective of the experiments is to validate the algorithms introduced in Section 2. In particular, the development of Algorithm 2 made several idealized assumptions about the environment and these assumptions can be investigated experimentally. The choice of α is a major parameter in the following experiments. We also wish to verify that lower delays are achieved by an integrated algorithm that does both document classification and bandwidth division. Finally, the scalability of various popularity estimation algorithms remains to be verified.

3.1 Methodology

The experimental analysis leverages on an existing prototype middleware. The middleware supports the hybrid dissemination scheme utilizing multicast push and unicast pull. It acts as a reverse-proxy to a Web server for the delivery of documents that are materialized views [23]. A simulated client uses the middleware and generates Poisson requests for documents with a Zipf probability distribution. In this paper, we report on the case in which the size of the documents is fixed to 0.5KB, and we have additional evidence suggesting that results are fundamentally the same with variable sized documents.

An objective of this evaluation was to isolate algorithm performance from network factors, such as network congestion or routing transients. On the other hand, scalability is asserted when requests are generated by a large number of clients. Our solution was to run both the client and server on the same machine so that network effects would not be visible. (Although the emulation runs on a single machine, the middleware is capable of running on a distributed environment [23].) Aggregate requests from multiple clients was simulated by a background request filler. The filler simulates a specified number of clients, and sends requests to the server. The requests by the filler are treated identically to those made by another distinguished client, except that we record latency only for the requests from the distinguished client. All experiments were run for 10000 requests and figures reflect the average statistics from these runs. The computer used in these simulations was a 2.0Ghz dual processor machine with 1.2GB of RAM and running Linux Redhat Version 8.0. JRMS was used for multicasting [24].

Parameter	Value	Default
Document Size	0.5K bytes	0.5K bytes
Zipf parameter θ	1.1 - 2.0	1.5
System Bandwidth	100000 bytes/sec	100000 bytes/sec
Request rate λ	250 / sec	250/sec
Total items n	100 - 10000	1000
Total Requests Made	10000	10000
α	1.1 - 4.2	2
ϵ	.005	.005

Table 2: Simulation Parameters

Table 2 describes the parameters used in the experiments. Although we explored the algorithm sensitivity to parameter values within the stated range, we report for compactness only on the default values unless otherwise noted.

3.2 Document Classification and Bandwidth Division

Figure 1 shows the effects of various values of α on the average latency of Algorithm 2. The curve in Figure 1 is jagged because an infinitesimal change in α can have a discrete effect in the number of items pushed. Figure 1 shows that the value of α that minimizes average latency is between 2.0 to 3.0. We adopt $\alpha = 2.0$ in the rest of the paper — although this is not the actual minimum, any value in the range produces similarly good results.

Figure 2 can be interpreted as a brute force search for a good bandwidth split and document classification by trying several closely spaced values of k and pushBW. In the chart legend, the first number in the bandwidth split refers to pull. In addition to the points plotted in the figure, we verified that if less than half of the bandwidth was devoted to pull, the latency was suboptimal. In this scenario, Algorithm 2 assigns the most popular 7 documents on the push channel, and allocates 63% percent of the bandwidth to push. The figure shows the algorithm's outcome with a circular point and an arrow pointing to it. The solution produced by our algorithm is better than any other point in the diagram. More specifically, our algorithm chose a split of 63/37 and the closest brute force curve in the figure is the 65/35 curve. The 65/35 line was also the lowest in the graph. Algorithm 2 chose $k = 7$ point as the number of push documents, which is also the minimum point on the 65/35 curve. Thus, Algorithm 2 chose a better bandwidth split than the brute force approach and a document classification that was just as good.

Let $G(k)$ be the average latency if the k most popular documents are placed on the push channel. The function $G(k)$ is a weighted average of the average latency for pushed documents and the average latency for pulled documents. A graph showing an idealized $G(k)$ from [25] is shown in Figure 3. The function $G(k)$ has a unique local minimum, which can be be found by local search [25]. Figure 3 shows that the minimum of $G(k)$ is to the right of the intersection of the push and pull curves. In this case, pulled documents would have lower latency than pushed documents. The actual curve that we obtained from our experiments is shown in Figure 4. Notice that the minimum of $G(k)$ is to the left of the intersection of the push and pull curves, and thus pushed documents have lower latencies than pulled documents. Further, the minimum of $G(k)$ occurs at a relatively small value of k, and thus complicated hierarchical schemes for the push channel may not be useful

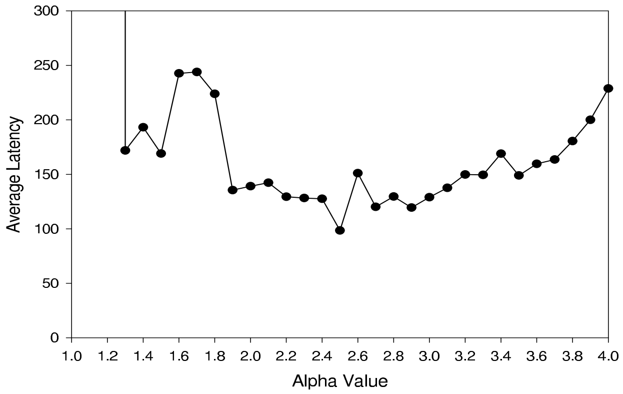

Figure 1: Effects of various α values on average latency

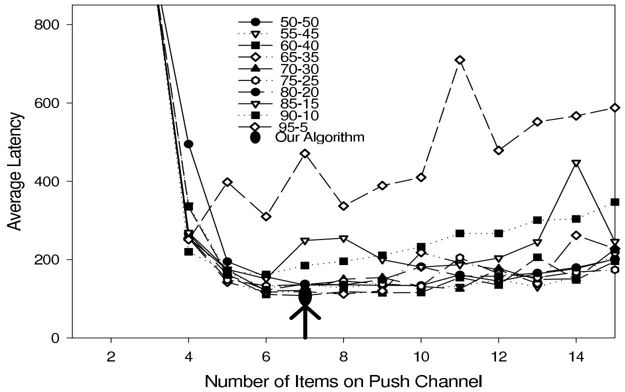

Figure 2: Demonstrating the optimality of Algorithm 2 for document classification and bandwidth division. The arrow points to the single point found by the algorithm.

in this setting. The location of the minimum is due to two complementary reasons. First, the most popular items are chosen for push and are also those to which a Zipf (or Zipf-like) distribution gives substantially more weight. Therefore, if a solution favors multicast push, it will also have the largest impact on the globally average delays. Second, the unicast pull curve levels off and, from that point on, the exact choice of k has little impact on pull delay. In other words, pull delays are practically minimized at the point $\bar{k}$ where the pull curve flattens out. However, $\bar{k}$ precedes the intersection of the pull curve with the push curve, and so the overall minimum occurs before that intersection.

In conclusion, Algorithm 2 was shown to be better than the best value returned by a brute force search. Furthermore, the integrated algorithm led to a behavior of the push and pull curves that differs qualitatively and quantitatively from previously published work, e.g., in terms of the relative behavior of push and pull delays.

3.3 Report Probabilities

In order to determine the usefulness of our proposed push popularity scheme, we compare it to a solution found in a comparable work to our own. The solution for the push popularity problem proposed in [25] was to occasionally drop each pushed document i off of push channel so that clients would have to make explicit requests to i. However, there is a danger that these explicit requests for i could overload the server. Thus, in [25] it was recommended that i should be dropped as short of a period of time as possible. The shortest possible time the the document can be dropped is one

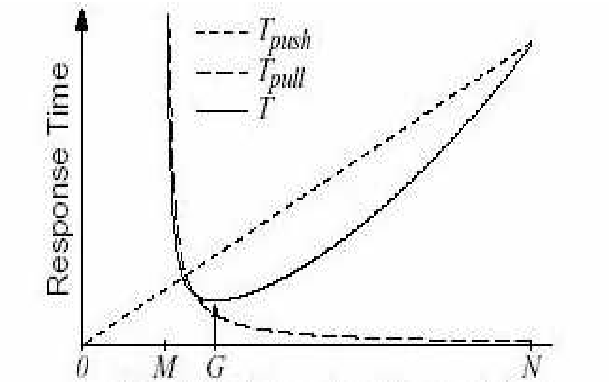

Figure 3: Relation of Push and Pull latencies as number of items pushed is changed, according to Stathatos *et al.*

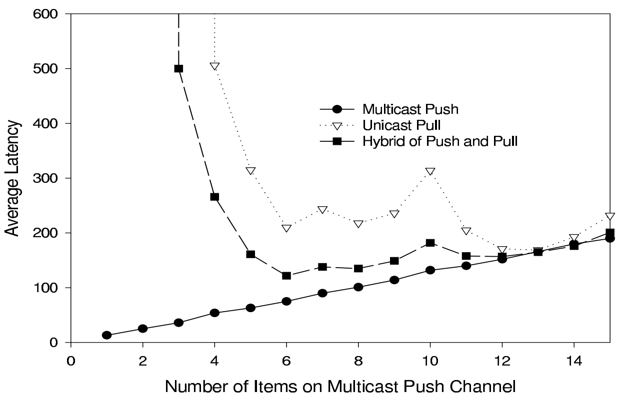

Figure 4: Relation of Push and Pull latencies as number of items pushed changes according to our experiments

broadcast cycle. However, we show here that even such a short drop disrupts the server. While our proposed method does not suffer disruptions.

Figure 5 shows the average latencies around the broadcast cycle T when the most popular item is dropped from the push channel. The figure shows a performance degradation for about 5 broadcast cycles. Basically, looking at the graph shows that before the drop occurs, the system is in a steady state of response times. However, once the item is dropped down the clients are no longer getting requests off the push channel. Instead, they must make requests directly to the server. Based on the zipf distribution, as mentioned earlier, the bulk of requests were for items that were on the push channel. Therefore, dropping an item down causes a brief but substantial influx of requests to the server. This brief surge causes response times for requests during the given broadcast cycle and a few subsequent cycles to suffer while the server recovers and returns to its steady state.

Figure 6 shows the average latency over the next 5 broadcast cycles when the i^{th} most popular document is dropped from the push channel for one broadcast cycle. The flat line represents the average response time using our method for push popularity. If the most popular document is dropped, then we see a 35% increase in average latency over the next 5 broadcast cycles. If the 6th most popular document is dropped, we see an 8% increase in average latency over the next 5 broadcast cycles. This increase is in comparison to using the simple yet effective scheme we proposed of simply including a popularity estimator with the broadcast index.

In fairness, our proposed method has the disadvantages that it requires extra space in the broadcast index and it slightly increases the request rate at the server during all broadcast cycles.

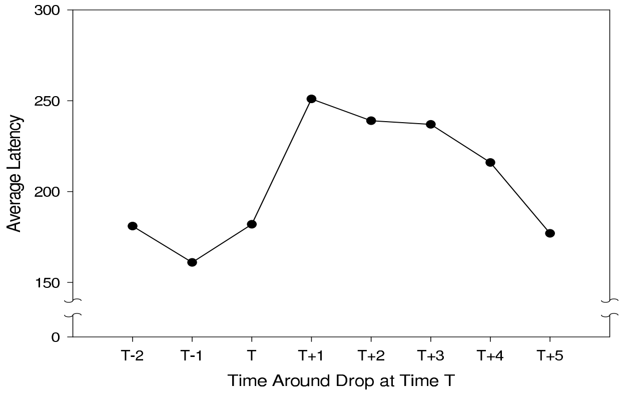

Figure 5: Effect on latency of demoting an item to test its popularity.

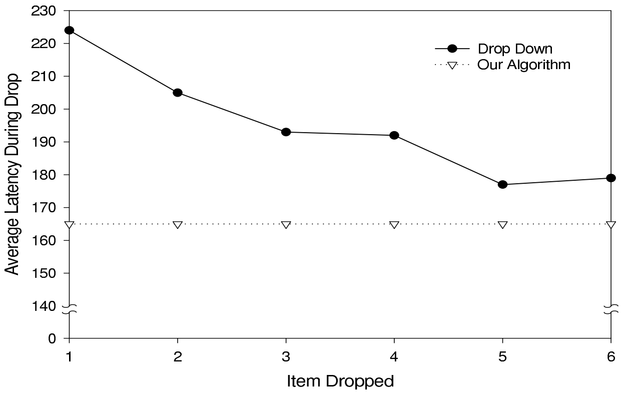

Figure 6: Comparison of drop down method to our probability method.

4. RELATED WORK

Scalable data delivery has often been approached through data caching and replication such as, for example, in client or proxy caches [11, 16], server-side caches [10], and content-delivery networks [22]. Moreover, back-end methods are deployed between a web server and a back-end database server and include web server cache plug-in mechanisms and asynchronous caches [5, 17, 18, 19]. These approaches follow the traditional *unicast pull* paradigm, whereby data is delivered from the server to each client individually on demand. In turn, the unicast pull approach severely limits the inherent scalability of data delivery.

The document classification problem was introduced in [25]. In addition to directly related work, some other work has been done addressing the issue of hot and cold documents and of bandwidth division, though not in the context we are describing. In [1, 14, 2, 26] the issue of mixing pull and push documents together on a single broadcast channel is examined. The idea is that popular documents are similarly considered hot, and are continuously broadcast while all other documents are cold. These documents are request through a back channel and scheduled for broadcast.

Similarly, in [1] the authors discuss how to divide the broadcast channel bandwidth between hot and cold documents. The main difference between previous work and ours is previous work deals with a broadcast environment with a single channel and focuses on scheduling items, not how to divide them into hot and cold. We are looking into the division of both documents and bandwidth to minimize latency.

The hybrid scheme relies on estimates of the popularity of documents in the web site because popularity determines the assignment of documents to dissemination modes. Popularity estimation can be approached separately for pulled and for pushed documents. Pull popularity can be solved in sub-linear space by monitoring the client request stream [9]. As for push popularity, the problem is complicated by the absence of a client request stream. One solution is to occasionally drop each pushed document from the push channel, thus forcing clients to send explicit requests. Such requests can then be counted and the document popularity estimated [25]. A related problem is multicast group estimation [20], which can be specialized as follows in our context: remove a document from the multicast push channel and re-insert it as soon as the first request for that document is received. The document popularity can be estimated by the length of time it takes for the first client request to reach the server.

5. CONCLUSION

In this paper we examined three data management problems that arise at the server in a hybrid data dissemination scheme. We argued that the document classification problem and bandwidth division problem should be solved in an integrated manner. We then presented a simple, yet essentially optimal, algorithm for the integrated problem. We validated the optimality of our algorithm experimentally. We proposed solving the push popularity problem by having each client request a hot document D_i with some probability s_i, which the server sets in the push index. We looked at the difference in using our push popularity scheme versus using a scheme which simply drops an item off the push channel in order to test its popularity. We showed that dropping an item off the hot channel for one broadcast cycle can appreciably increase the average latency for approximately five broadcast cycles. Our proposed scheme does not suffer from such disruptions.

6. REFERENCES

[1] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull data broadcast. In *ACM SIGMOD*, May 1997.

[2] D. Aksoy and M. Franklin. Rxw: A scheduling approach for large-scale on-demand data broadcast. *ACM/IEEE Transactions on Networking*, 7(6):846–860, December 1999.

[3] K. C. Almeroth, M. H. Ammar, and Z. Fei. Scalable delivery of Web pages using cyclic best-effort (UDP) multicast. In *INFOCOM*, 1998.

[4] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. Dbis toolkit: Adaptable middleware for large scale data delivery. In *ACM SIGMOD*, jun 1999.

[5] M. Altinel, Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. G. Lindsay, H. Woo, and L. Brown. Dbcache: Database caching for web application servers. In *ACM SIGMOD*, June 2002.

[6] Y. Azar, M. Feder, E. Lubetzky, D. Rajwan, and N. Shulman. The multicast bandwidth advantage in serving a web site. In *3rd NGC*, pages 88–99, 2001.

[7] P. Chrysanthis, K. Pruhs, and V. Liberatore. Middleware support for multicast-based data dissemination: a working reality. In *WORDS*, January 2003.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[9] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking frequent items dynamically. In *Proceedings of Principles of Database Systems*, 2003.

[10] A. Datta, K. Dutta, K. Ramamritham, H. M. Thomas, and D. E. Vandermeer. Dynamic con-tent acceleration: A caching solution to enable scalable dynamic web page generation. In *ACM SIGMOD*, 2001.

[11] A. Datta, K. Dutta, H. M. Thomas, D. E. VanderMeer, Suresha, and K. Ramamritham. Proxy-based acceleration of dynamically generated content on the world wide web: an approach and implementation. In *ACM SIGMOD*, pages 97–108, 2001.

[12] M. Franklin and S. Zdonik. " data in your face ": Push technology in perspective. In *ACM SIGMOD*, June 1998.

[13] Y. Guo, M. Pinotti, and S. Das. A new hybrid scheduling algorithm for asymmetric communication systems. *ACM SIGMobile Computing and Communications Review*, 5(3), July 2001.

[14] A. Hall and H. Taubig. Comparing push- and pull-based broadcasting or: Would "microsoft watches" profit from a transmitter? *Lecture Notes in Computer Science*, 2647, January 2003.

[15] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *OSDI*, pages 197–212, October 2000.

[16] S. Jin and A. Bestavros. Temporal locality in Web request streams: sources, characteristics, and caching implications. In *SIGMETRICS*, pages 110–111, 2000.

[17] A. Labrinidis and N. Roussopoulos. Webview materialization. In *ACM SIGMOD*, May 2000.

[18] A. Labrinidis and N. Roussopoulos. Webview balancing performance and data freshness in web database servers. In *VLDB*, 2003.

[19] Q. Luo, S. Krishnamurthy, C. Mohan, H. Woo, H. Pirahesh, B. G. Lindsay, and J. F. Naughton. Middle-tier database caching for e-business. In *ACM SIGMOD*, June 2002.

[20] J. Nonnenmacher and E. W. Biersack. Scalable feedback for large groups. *IEEE/ACM Trans. Netw.*, 7(3):375–386, 1999.

[21] V. Padmanabhan and L. Qiu. The context and access dynamics of a busy web site: Findings and implications. In *ACM SIGCOMM'00*, 2000.

[22] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the Web: An open proxy's view. In *HotNets-II*, 2004.

[23] V. Penkrot, J. Beaver, M. Sharaf, S. Roychowdhury, W. Li, W. Zhang, P. Chrysanthis, K. Pruhs, and V. Liberatore. An optimized multicast-based data dissemination middleware: A demonstration. In *ICDE 2003*, March 2003.

[24] P. Rosenzweig, M. Kadansky, and S. Hanna. The java reliable multicast service: A reliable multicast library. SMLI TR-98-68, Sun Microsystems, 1998.

[25] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *VLDB*, 1997.

[26] P. Triantafillou, R. Harpantidou, and M. Paterakis. High performance data broadcasting systems. *Mobile Networks and Applications*, 7:279–290, 2002.

[27] W. Zhang, W. Li, and V. Liberatore. Application-perceived multicast push performance. In *IPDPS*, 2004.