Multicast Scheduling for List Requests

Vincenzo Liberatore

***Abstract*— Advances in wireless and optical communication, as well as in Internet multicast protocols, make broadcast and multicast methods an effective solution to disseminate data. In particular, repetitive server-initiated broadcast is an effective technique in wireless systems and is a scalable solution to relieve Internet hot spots. A critical issue for the performance of multicast data dissemination is the multicast schedule. Previous work focused on a model where each data item is requested by clients with a certain probability that is independent of past accesses. In this paper, we consider the more complex scenario where a client accesses pages in blocks (e.g., a HTML file and all its embedded images), thereby introducing dependencies in the pattern of accesses to data. We present a sequence of heuristics that exploit page access dependencies. We measured the resulting client-perceived delay on multiple Web server traces, and observed an average speed-up over previous methods ranging from 8% to 91%. We conclude that scheduling for multi-item requests is a critical factor for the performance of repetitive broadcast.**

***Index Terms*— Multicast, Scheduling, Web performance, Network Applications, Wireless networks.**

I. Introduction

SEVERAL emerging technologies and applications naturally lead to the adoption of broadcast or multicast as the primary method for data dissemination. Broadcast is the primary mode of operation of the physical layer in media such as satellites and optical networks. As a result, it is natural to develop broadcast applications for those media. Broadcast can also be used in networks other than wireless and optical as a method to solve scalability problems. For example, multicast methods can relieve the scalability problems of Web hot spots [1] and can support the operations of a content delivery network [2]. Multicast methods can be combined with other performance enhancing techniques, such as caching [3], [4]. Broadcast and multicast techniques have spawned research (e.g., [1], [5]) and commercial ventures [6], [7], [8] that aim at higher scalability.

A common data dissemination method is to use *repetitive server-initiated multicast* [9], [5], [3], whereby a server cyclically multicasts (or broadcasts) data to a large client population. As a general data management technique, repetitive broadcast can be used in both wired [5] and wireless [10] networks to disseminate a variety of resource types, including Web contents [1] and database records [11]. A critical issue for broadcast performance is its organization. Some broadcast items will be more popular than others, so it is natural to broadcast the hot items more frequently. Page popularity has been modeled in the literature in terms of the probability p_i that page i is requested by clients. For the model where the probabilities p_i are stationary and independent of past accesses, several algorithms have been proposed [12], [13], [14]. Extensions include the cases when broadcast pages have different sizes [15], [16] and when client objectives are described by polynomial utility functions [17].

In general, clients are seldom interested in individual data items, and attempt to download multiple items. For example, Web clients are seldom interested in only one HTML resource, but access almost always the HTML document along with all its embedded images [18]. Analogously, database clients often access multiple items to complete a read transaction [19]. In this paper, we will examine novel scheduling strategies that keep into account dependencies in the client accesses to resources. The objective is to reduce client-perceived latency when she downloads multi-item objects.

The presence of multi-item requests complicates the scheduling of the broadcast, as shown by the following examples.

Example 1: Suppose that E is an image embedded in page A. Consider a schedule that broadcasts E immediately after A. A request $\{A, E\}$ takes only slightly more than the time needed to retrieve A only. If E and A were broadcast in an arbitrary order that takes into account only their access frequencies, it is possible that one document is broadcast a long time after the other, thereby delaying the request completion time.

Example 2: Suppose that E is embedded in A as well as in another page B. Consider now a request for $\{B, E\}$. By the same token, E should be transmitted immediately after B. However, if E is broadcast after A *and* after B, the repeated transmission of E lengthens the broadcast cycles and could delay other pending requests. A different method is to send the three documents in the order $\ldots A, B, E \ldots$, which could potentially have better performance than repeating E.

Example 3: Consider a request for $\{A_1, A_2, \ldots, A_k\}$, where $A_1, A_2, \ldots, A_{k-1}$ are broadcast fairly often and A_k is seldom broadcast. The completion time of this request is tied up to the low transmission rate of A_k. In other words, frequent broadcast of hot items does not help the completion time of multi-document requests involving colder items.

In general, multi-item requests create complex dependencies in the document access pattern and can complicate the broadcast schedule. The paper will propose and analyze effective heuristics for the problem of multicast scheduling under dependencies in the request sequence. Algorithms are evaluated on multiple server logs of Internet hot spots.

The paper is organized as follows. In section II, we give background information on broadcast data management techniques and on broadcast scheduling. In sections III and V, we present a sequence of algorithms for broadcast scheduling and provide evidence of the limited applicability of known broadcast strategies. In section IV, we describe our experimental

Electrical Engineering and Computer Science Department, Case Western Reserve University, 10900 Euclid Avenue, Cleveland, Ohio 44106-7071. E-mail: vl@eecs.cwru.edu. URL: http://vorlon.cwru.edu/∼vxl11/.
Research supported in part under NSF grant ANI-0123929.

set-up. In section VI, we validate our algorithms on two more traces that we did not use to tune parameters. In section VII, we summarize work related to ours, and in section VIII we draw the conclusions of our investigation.

II. Background

Broadcast Environment: *Cyclical server-initiated broadcast and multicast* [9], [5], [3] can be used to execute data dissemination and are well suited for wireless and mobile environments, as well as for relieving Internet hot spots. A set of n pages is cyclically broadcast by a server to a large client population. The broadcast is initiated without client requests, i.e., it follows a "push" style of data dissemination. Furthermore, the broadcast is repetitive, that is, the server continuously cycles through its set of broadcast data. Examples of broadcast programs are illustrated in figure 1 and 2. When a client needs to read the contents of page i, it waits for the data source to broadcast i. The client does not need to listen continuously on the broadcast for page i to be sent if an appropriate index is broadcast as well; such index also allows the client to determine which contents are present in the broadcast data set [20], [21], [22], [23]. In its purest implementation, the server accepts no input from clients and simply cycles through its broadcast. In more complex schemes, the server accepts update transactions from clients [24], [19] or uses broadcast as a complement to other dissemination methods [25]. For example, a server can use broadcast to propagate hot documents, while it uses other methods for colder items [1]. A critical performance metrics for broadcast data dissemination is the amount of time that elapses between a client request and the time when the client has downloaded all requested pages.

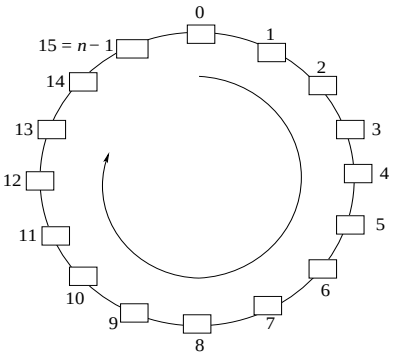


Fig. 1. An example of a flat broadcast program. Pages are numbered from 0 to $n-1$, and are cyclically transmitted by the server in that order.

The scope of the paper is to investigate algorithms to schedule the broadcast at the server site so as to reduce client-perceived latency. To focus on the scheduling problem, we make the following assumptions:

- The broadcast schedule is fixed by the server, and is known by clients.
- Pages are reliably received by the clients in the same order as they are broadcast.
- Pages are read-only, and cannot be updated by either the server or the clients.
- Clients receive pages only from a unique server broadcast over a single broadcast layer.

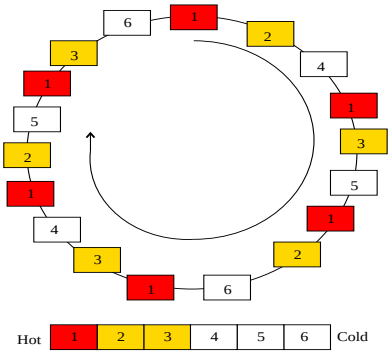


Fig. 2. A skewed broadcast schedule that is obtained by multiplexing on the same physical channel the logical channels $\{1\}, \{2,3\}, \{4,5,6\}$.

Term	Definition	Similar terms
Page	Broadcast transmission unit	-
Document	Data object that a client can identify by an id	Resource, ADU
Object	Collection of resources target of client requests	-

Dependency	Definition
Internal	between pages belonging to the same document
External	between documents

TABLE I
SUMMARY OF DEFINITIONS USED IN THE PAPER.

- The set of broadcast pages does not change.
- Data is broadcast at a constant rate.

All these restrictions can be removed in an actual implementation, as will be explained in the next sections. However, a more complex scenario would obscure the analysis of scheduling strategies, and so we do not consider it in the rest of the paper.

Broadcast Scheduling: The data source can schedule pages for broadcast according to a variety of strategies. The simplest broadcast strategy is to adopt a *flat broadcast schedule*, whereby each page is transmitted once every n ticks. A flat schedule is exemplified in figure 1. There are $(n-1)!$ distinct flat schedules, and section III will demonstrate that certain flat schedules have substantially better performance than others. Non-flat schemes are desirable when some pages are more popular than other, in which case hot pages should be devoted a larger fraction of available bandwidth. A simple way to differentiate pages is through *frequency multiplexing*: the data source partitions the data set across several physical channels according to their popularity. Differentiated treatment arises from aggregating a smaller amount of hot data on one channel and a larger amount of colder data on another channel. Since channel bandwidth is the same, the fewer hotter pages receive a proportionally larger amount of bandwidth than colder pages. Frequency multiplexing can be effective if multiple channels are available, but is unfeasible otherwise. An alternative is

time-division multiplexing, whereby pages are partitioned into a set of *logical channels*, and the logical channels alternate over the same physical channel [9]. The broadcast schedule is flat within a single logical channel, but hotter channels contain less pages or are scheduled for broadcast more frequently than colder channels. Thus, hot pages are transmitted more often than colder ones. Figure 2 gives an example of a broadcast schedule that is the time-multiplexed combination of three logical channels, each containing a different number of pages. Time-division multiplexing is potentially more flexible than frequency multiplexing in that it allows for a finer bandwidth partition. In particular, a logical channel can contain only one page, which results in a fine per-page transmission schedule. When the broadcast is scheduled on a per-page basis, pages are broadcast on the same physical channel with frequency proportional to their popularity.

A family of scheduling algorithm for time-multiplexed broadcast assumes that the data source has estimates of the probabilities with which clients need pages. The square-root law asserts that page i should be scheduled with frequency proportional to $\sqrt{p_i}$ [13], where p_i is the probability that i is requested by clients. A simple and practical 2-approximation algorithm is expressed by the *MAD* (Mean Aggregate Delay) rule [12], [13]. The MAD algorithm maintains a value s_i associated with each page i. The quantity s_i is the time since the last time page i was broadcast. The MAD algorithm broadcasts a page i with the maximum value of $(s_i + 1)^2 p_i$. MAD guarantees a cyclical schedule, and, in particular when all p_i's are equal, MAD generates a flat broadcast. The access probabilities p_i do not express dependencies between data items. Consider the following elementary example. Pages A and B are not accessed very frequently, but when A is accessed, page B is almost certainly accessed as well. In this scenario, the access probability p_B of page B is small, but the value of p_B is not fully expressive of the true access pattern to B.

We classify dependencies among pages in internal and external (table I). An *external dependency* arises when there is a dependency between the original resources in the client access pattern, as for example when the access probability of B is conditional to the previous occurrence of a request for A. An *internal dependency* arises when the underlying transport forces long documents to be broken in smaller portions. For example, IP-based data dissemination, such as [1], uses IP multicast to propagate data to a large client population. Since the broadcast can cross Internet links with different MTU's and reach clients with different reassembly buffer sizes, a document is fragmented at the source into *pages* of approximately equal size so that pages fit within the IP size limits [26], [27]. We make the distinction between *documents*, which can be identified and requested by clients through a resource identifier, and *pages* which are broadcast units of roughly equal size and in which original resources are partitioned. Documents can also be variously referred to as *resources* or *Application Data Units (ADU)*. We will call the dependency among pages from the same document an *internal dependency*. A common definition in the literature is that of an *object*, which is a collection of resources that are the target of a client request. With this terminology, documents within the same object show an external dependency, while pages within the same document have an internal dependency. Pages belong to only one document whereas documents can belong to an arbitrary number of other objects. As a result, internal dependencies are likely to create a simpler scenario than external dependencies. In this paper, we consider both internal and external dependencies. Client-perceived delays will be partitioned into two components: the *seek time* is the time that clients wait to receive the first page of an object, and the *transfer time* is the time that clients wait to receive the rest of the object.

III. Circular Arrangement

In this section, we examine whether transfer time can be reduced within the context of flat broadcast schedules. Skewed (i.e., non-flat) schedules will be examined later in section V. Although flat schedules do not transmit a page any more frequently than any other page, flat schedules performance can vary significantly when there are dependencies between page accesses. For example, suppose that page B is always requested after page A. A random flat schedule takes about n broadcast ticks in the expectation to retrieve the object $\{A, B\}$: $n/2$ ticks to retrieve A followed by $n/2$ ticks to retrieve B. A flat schedule that arranges B immediately after A takes only $n/2 + 1$ ticks to retrieve the same object $\{A, B\}$. Although the second schedule did not reduce the seek time (i.e., the time to retrieve A), it was extremely effective at reducing transfer time (i.e., the time to retrieve B after A has been downloaded). In general, a page can appear in multiple objects (e.g., a resource embedded in several other documents), and so object pages cannot always be grouped in a contiguous broadcast interval. A possible solution would be to replicate the page once for each containing object, but such approach could unnecessarily lengthen the broadcast schedule and result in longer seek times. In section V, we will explore a limited replication mechanisms that duplicates the hottest pages for a controllable number of times. First, however, we examine in which ways and to what extent transfer times can be reduced solely in the context of flat schedules.

Problem Model: We model the problem of reducing transfer time as the following graph optimization problem. We associate a node to each page and insert an arc (i, j) from page i to page j when there is a dependency between i and j that makes j more likely to be accessed after page i. We will also associate a weight to the arc (i, j) proportional to the strength of the dependency. We call such graph the *dependency graph* of a trace because its arcs express dependencies between pages and arc weight express the strength of the dependency. We then seek to arrange pages around a broadcast cycle so that the weighted arc length is minimized. More precisely, we define the *Minimum Circular Arrangement (MCA)* problem as

Instance: A directed graph $G = (N, A)$ and non-negative arc weights $w(e) \in \mathbf{N}$ for each $e \in A$.

Question: Find a one-to-one function $f : N \to \{0, 1, \ldots, n-1\}$ that minimizes

$$\sum_{e=(u,v)\in A} w(e)\ell(e) \tag{1}$$

where $n = |N|$ and $\ell(e) = ((f(v) - f(u)) \bmod n)$.

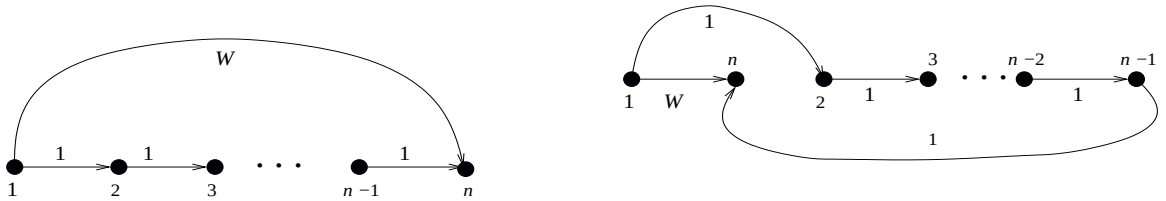


Fig. 3. A directed linear arrangement (left) whose cost is a $\Omega(n)$ factor away from the cost of the optimum circular arrangement (right). The value of h for the arc of weight W is $n-1$ in the topological arrangement and is only 1 in the circular arrangement. The value of h for the back arc in the circular arrangement is 2, and was 1 in the linear arrangement. Thus, the cost for the heavy arc can be substantially reduced with minimal changes in the cost of other arcs.

MST algorithm
Given a dependency graph $G = (N, A)$
Let P be a partition of the nodes of the graph G, initialized to n singleton sets.
(The algorithm maintains an ordering of each set in P)
for all arcs $e = (u, v)$ of G in non-increasing order of weight:
  Let P_u be the component of P that contains u and P_v be the component that contains v
  if $P_u \neq P_v$
    Insert e in the spanning tree T
    Unite P_v and P_u and append the ordering of P_v after the ordering of P_u
Concatenate all orderings of sets in P and
return such ordering.

Fig. 4. The MST algorithm.

The quantity $\ell(e)$ is the distance between arc endpoints in the circular arrangement f and will be said to be the *length* of arc e in f. In the MCA model, there is some degree of latitude in the choice of the objective function. We chose a linear objective (1) because it forces related pages to be clustered next to each other while the objective function remains relatively easy to analyze.

A question related to MCA is the *linear arrangement problem*, where the graph nodes are to be arranged along a line (instead of a circle) and the graph is assumed to be acyclic. The minimum linear arrangement has been extensively studied: it is NP-hard [28] and several approximation algorithms have been proposed [29], [30], [31], [32]. A simple example shows that the linear and circular arrangement problems are intrinsically and radically different, so that approximation algorithms for linear arrangement are most likely irrelevant in the context of circular arrangements. Specifically, we demonstrate that the optimum linear arrangement can cost $\Omega(n)$ times as much as a circular arrangement even when the underlying graph G is acyclic. Consider the graph in figure 3, and observe that it has a unique topological ordering $1, 2, \ldots, n$ at a cost of $(n-1)(W+1)$, whereas the circular arrangement $1, n, 2, \ldots, n-1$ has a cost of $W + n + 1$. We then take $W = \Omega(n)$ to make the cost ratio $\Omega(n)$. On the other hand, if we consider any circular arrangement, the cost due to an arc is at most $n-1$ times the cost that the optimum pays for the same arc. As a result, an $O(n)$-approximation algorithm is trivial. We conclude that circular and linear arrangement problems are in general unrelated.

Hardness: We consider the theoretical solvability for MCA, and to this end we introduce the following decision version of the optimum circular arrangement problem, which we call the *Circular Arrangement Problem* (CA):

Instance: A directed graph $G = (N, A)$, non-negative arc weights $w(e) \in \mathbf{N}$ for each $e \in A$, and a positive integer K.

Question: Is there a one-to-one function $f : N \to \{0, 1, \ldots, n-1\}$ such that

$$\sum_{e=(u,v)\in A} w(e)\ell(e) \leq K?$$

where $n = |N|$ and $\ell(e) = ((f(v) - f(u)) \bmod n)$.

Proposition 1: The Circular Arrangement Problem (CA) is NP-complete.

Proof: [Sketch] The proof is a reduction from the direct linear arrangement problem. ■

MST Heuristic: Although no polynomial-time algorithm is likely to solve MCA optimally, a reasonably good solution can be obtained through a heuristic applied to the dependency graph. Our procedure is based on a topological ordering of a maximum spanning tree (MST) of the dependency graph. Specifically, the algorithm maintains a partition of the node set and an ordering for the nodes within each partition. Initially, the node partition consists of n singleton, one for each node of the original graph. Then, our procedure computes the maximum spanning tree of the dependency graph with Kruskal's algorithm [33], and, when the algorithm combines two node sets, the heuristics also combines the two component orderings. On the whole, the algorithm is in figure 4.

The algorithm is greedy, in that it arranges nodes as close as possible if there is an arc with a large weight between them. At the beginning, the algorithm arranges the nodes (i.e., pages) u and v next to each other if the arc (u, v) has maximum weight (i.e., if v appears in the same object as u for the maximum number of times). As the algorithm progresses, the algorithm combines the ordering of P_u and P_v if the arc (u, v) has maximum weight among all remaining arcs (i.e., the page orderings are combined if there is a page v that appears in the same object as u for the maximum number of times). As a result, the algorithm can be viewed as producing a sequence of page clusters P and combining two clusters on the basis of dependencies between two pages; as clusters are combined, their orderings are

trace	begin (coord. univ. time)	end (coord. univ. time)	length		Broadcast trace				
			log	sanitized	n	clients	m	objreq	sngobj
1	[30/Jun/1998:15:00:00]	[01/Jul/1998:00:00:00]	57639163	90.36%	225	127256	80046572	7626200	28.40%
2	[01/Jul/1998:15:00:00]	[02/Jul/1998:00:00:00]	9079767	94.29%	174	67261	9967240	896297	25.12%
3	[08/Jul/1998:20:00:00]	[09/Jul/1998:00:30:00]	15500700	90.95%	206	59479	20473599	2083034	28.98%
4	[09/Jul/1998:20:00:00]	[10/Jul/1998:00:00:00]	1986787	94.36%	200	17563	2449844	199169	21.29%

TABLE II
CHARACTERISTICS OF CLIENT WEB TRACES COLLECTED FROM THE WORLD CUP 98 SERVER TRACE.

concatenated as well. Therefore, the algorithm gives as a by-product a page clustering that depends on the frequency with which pages belong to the same object. The MST algorithm takes $O(n^2 \log n)$ time in the worst case and, on our simulations (section IV), it always ran in less than 600 ms on a Ultra 60 workstation with a 450Mhz CPU, 4MB L2 cache, 512 MB of memory, Solaris 8, `g++` compiler, and LEDA data structures libraries [34].

IV. EVALUATION

Methodology: We limit our empirical analysis to Web logs of Internet of hot spots. Although the concepts in this paper should be applicable to both wired and wireless networks and to several applications, the restriction allows us to obtain more exhaustive results and to use several publicly available traces. Experiments were executed with traces that were extracted from the log of the HTTP servers for the Soccer World Cup 98. The World Cup trace includes more than one billion requests over a period of 1 1/2 month and is one of the largest trace analyzed to date [35]. Furthermore, the World Cup servers received up to 10 million requests per hour. As a result, the World Cup site is one of the most busy recorded so far, which makes it an ideal testbed for multicast data dissemination. Additional traces will be considered in section VI to execute a blind validation of algorithms.

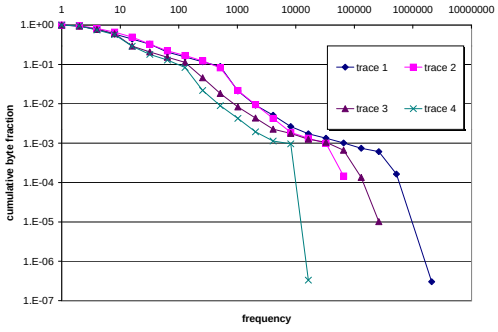


Fig. 5. Cumulative percentage of bytes requested with at least the given frequency.

The server logs report only requests that percolate to the origin server, and in particular, the logs do not report requests that are satisfied by intermediate caches. We extracted from the complete server logs a set of four subtraces that correspond to the four most active periods (table II). We kept requests that fell in the target busy interval, that are GET or HEAD methods for HTML, image, Java, or compressed resources, and that

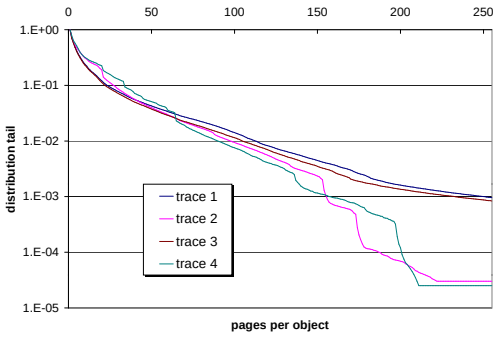
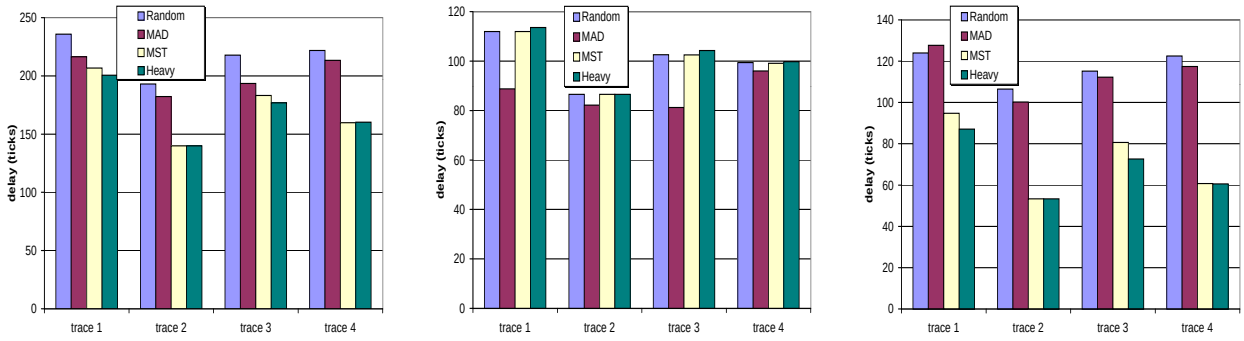


Fig. 6. Tail of the distribution of the number of pages per object. The vertical axis gives the frequency with which an object had at least the number of pages on the horizontal axis.

gave rise to 200 (ok) and 304 (not modified) response codes. Some document sizes changed during the course of the trace. Size change is due either to interrupted transfers or to document contents updates. Although updates can be incorporated in broadcast environments [19], [24], our logs do not allow us to determine the origin or nature of size changes, and so we restrict this study to fixed-size documents. After documents with changing sizes were eliminated, the resulting subtraces contain more than 90% of the transfers made during the chosen intervals (table II, column "sanitized").

In Internet data delivery, documents that are referenced sporadically are not usually multicast [1], [36]. Figure 5 gives the cumulative size (as a fraction) of resources that were requested with at least a certain frequency. The distribution has a knee in correspondence of $\chi = 7 \cdot 10^{-4}$. In other words, if the hottest χ fraction of bytes is broadcast, the broadcast will contain extremely popular items, but a further increase of the broadcast size n would quickly result in significantly colder items occupying the broadcast schedule. Consequently, we inserted in the broadcast only the resources corresponding to the hottest χ fraction of bytes. An alternative choice is based on a dynamic assessment of document popularity [36]. We did not use any dynamic schemes in this paper because we were interested in isolating the performance of scheduling algorithms from that of other methods. We envisage that a real implementation would need to support both scheduling and dynamic document selection. As in [1], broadcast documents were divided into 512B pages. A GET method translates into a request for all document pages, a HEAD request for the appropriate number of pages at the beginning of the document, and a non-modified reply into a request for the first page of that document. In practice, clients



(a) Total latency (b) Seek (c) Transfer

Fig. 7. Average latency expressed as number of broadcast ticks. *Seek* latency is the number of ticks required to retrieve the first document page. *Transfer* latency is the number of ticks to retrieve the remaining document pages.

are often interested in a collection of related documents, such as an HTML file and its embedded images. The HTTP protocol does not aggregate document requests into object requests. In these cases, a heuristic is that if two documents are requested within one second of each other in the trace, they belong to the same logical object [37]. In our experiments, the client triggers requests for all other pages in the object upon reception of the first object page; such scheme can be implemented through prefetching hints embedded in the first object page [38]. As cached resources do not appear in the server logs, the client requests are akin to GETLIST method invocations [18]. Table II gives the number n of pages in the simulated server broadcast, the number of unique IP addresses generating requests for broadcast pages, the number m of page requests, the number *objreq* of object requests, and the percentage *sngobj* of single page object requests. Some objects are big, but, depending on the trace, 21% to 29% of objects contained only one page. Figure 6 plots the distribution of the number of pages within objects.

Another design choice is the speed at which data is broadcast. Previous work suggests an optimal rate of 256 Kbps for IP multicast [1]. For comparison, if a single unicast-based server had been connected at the same rate, it would not have scaled to satisfy logged requests for broadcast resources in the World Cup traces. We simulated several rates ranging from 48 Kbps (to support most modems) to 1.544 Mbps (a T1 line). Most of these rates are also within the capacity of short-range or 3G wireless technology. A *broadcast tick* is the time needed to transmit a page. The duration of a broadcast tick clearly depends on the broadcast rate. Latencies were measured both in seconds and as number of ticks; in the latter case (latency as number of ticks), delays did not significantly depend on the broadcast rate. The simulations will employ only static values of available bandwidth. The more general problem of dynamically adjusting the transmission rate to the available bandwidth is explored in [39], [5], [40].

Results: Figure 7 compares a random flat broadcast, a MAD broadcast [12], [13], which is based solely on stationary access probabilities, and the MST heuristic. The figure shows the average delays expressed as number of broadcast ticks. For comparison, at the given broadcast rate, a unicast server would not have been able to satisfy the requests for broadcast

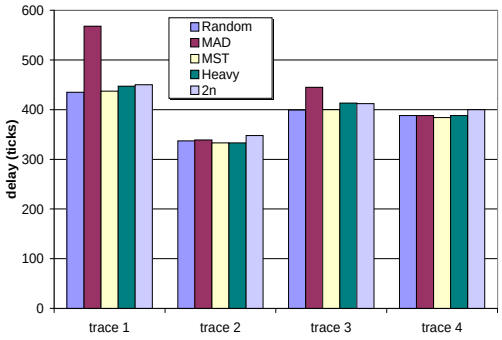


Fig. 8. Total latency in the 99th percentile expressed as number of broadcast ticks.

resources. The reported delays are for requests to the origin server, and so any performance improvement is in addition to those due to caching. In these experiments, MAD schedules lead to improvements over flat schedules of the order of 6% to 13%. MAD schedules were particularly effective at reducing seek time (from 4% to 26%), but did not provide a clear advantage in terms of transfer time (from -3% to 6%). The MST algorithms outperformed MAD by 5% to 34%. Not surprisingly, MST seek time was comparable to that of a random schedule and it was roughly $n/2$. However, MST resulted in a substantial reduction of transfer times, which in two traces was half as much as MAD's. The waiting time reduction is more marked for trace 2 and 4; we believe that the more pronounced improvement is due to the smaller percentage of single objects request in those traces (table II).

The distribution of the delays was collected as well and figure 8 shows the 99th percentile of the delays. Flat schedules, and MST in particular, never take more than $2n$ ticks to download an object, whereas no worst-case bound holds for MAD. Correspondigly, figure 8 shows that the delay of MST and random is always below $2n$, whereas MAD exceed $2n$ on both trace 1 and 3. It can be also noticed that if in a trace MAD did better on the average, it did more poorly in the 99th percentile. Intuitively, MAD attempts to optimize for the average case at the expenses of the tail whenever possible.

In summary:

Random flat broadcast is not a particularly good strategy on

Heavy algorithm
Construct an MST schedule f and determine all θ-heavy arcs
for each page v that is the head of a heavy arc
  Locate v's position in the MST ordering and, starting from this position,
  do
    Scan the MST ordering in the reverse direction of the broadcast order until the tail u of a heavy arc is (u, v) is found
    if u is more than ℓ ticks away from the next time v will be broadcast,
      then Insert v in the broadcast after u
  until the scan returns to v's original position
return the resulting schedule.

Fig. 9. The Heavy algorithm.

the average, but it provides a worst-case bound ($2n$).

MAD improved on a random flat broadcast by 6% to 13% on the average, but it provides no worst-case guarantees.

MST improved over MAD by 5% to 34% on the average by exploiting access pattern dependencies, and, since it follows a flat broadcast, it has the same worst-case bound ($2n$) as the random flat schedule.

V. Beyond Flat Schedules

We have dealt so far with two types of scheduling strategies: skewed schedules that exploit stationary access probabilities (i.e., MAD) and flat schedules that exploit page dependencies (i.e., MST). Up to this point, we viewed the two approaches as antithetic. In this section, we examine ways to combine the two paradigms to reduce client-perceived latencies. The resulting schedule should be skewed to favor hotter pages over colder ones and should be arranged in such a way as to reduce user-perceived latency.

Heavy Arcs: An MCA solution f can contain arcs that have a large value of $w(e)\ell(e)$ and that consequently contribute to a large fraction of the CA objective value.

Definition V.1: An arc is *θ-heavy* if $w(e)\ell(e) \geq \theta$ for a threshold value θ. If the value of the threshold θ is clear from the context, we will simply designate such arcs as *heavy*.

A flat schedule is inherently limited in its ability to eliminate heavy arcs. For example, suppose that the dependency graph has an arc from every node u to a designated vertex v and that the weights of the arcs $e = (u, v)$ are large, e.g., $w(e) > 2\theta/n$. Then, there are $\Omega(n)$ arcs with $w(e)\ell(e) \geq w(e)n/2 \geq \theta$.

Definition V.2—[33]: Let $G = (N, A)$ be a directed graph. The *head* of an arc (u, v) is node v and the *tail* is node u.

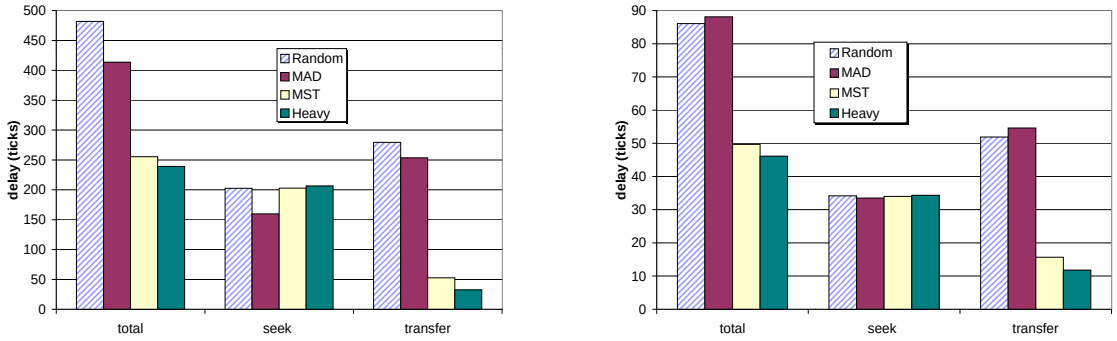
A method to reduce the impact of heavy arcs is to broadcast the page corresponding to the arc head soon after the tail. As a result, the arc length $\ell(e)$ is reduced and so is its contribution $w(e)\ell(e)$ to the CA objective value. The drawback is that the schedule contains more pages, and so the length of other arcs can increase. Therefore, a schedule should not replicate an excessive number of arc heads for an excessive number of times. The degree of replication can be controlled by tuning two parameters. The first parameter is the threshold θ for an arc to be considered heavy. The second parameter is the maximum length $\bar{\ell}$ of a heavy arc in the new schedule. A larger value of $\bar{\ell}$ allows a heavy arc to be longer in the new schedule, and thus the arc head v to be replicated a smaller number of times. To simplify the implementation, we ignored outgoing arcs from a replicated node. A simple greedy strategy calculates the smallest number of times a page is replicated along a schedule so that heavy arc length is no more than $\bar{\ell}$. The resulting algorithm is in figure 9.

We found that good parameter values are $\theta = 1$ and $\bar{\ell} = 32$ across all traces; such algorithms will be denoted as the *heavy* algorithm. The heavy algorithm is fast (no more than 640 ms on the same machine and traces as in section III). Figure 7 gives the latency for the resulting strategy for the four traces; MAD was outperformed from 8% to 33% by the heavy algorithm.

Square-Root Scheduling: We have attempted several methods to integrate MST and heavy scheduling with other methods based on independent probabilities and on the square root law, but we were not able to achieve any appreciable performance improvement over the heavy algorithm. We believe that this is due to the page access distribution in our traces. Figure 10 plots page access frequency as a function of page rank. The most popular page is assigned a rank of 1 and the least popular page a rank of n. In a log-log plot, a Zipf distribution $\Pr[i] \propto i^{-\alpha}$ would appear as a line with slope $-\alpha$. Page popularity can be explained by a Zipf distribution, although with rather low confidence ($0.65 \leq R^2 \leq 0.76$). Furthermore, the fitted Zipf distribution is only mildly skewed, with $0.65 \leq \alpha \leq 0.76$. Such result is consistent with the analysis in [35], where lack of skewness was attributed to factors such as the use of the most popular embedded images across most of the site objects and the presence of caches interposed between clients and servers. At any rate, low values of α lead to an almost flat square-root broadcast and so they hamper the potential for improvement of schedules based on independent probabilities. We conclude that square-root schedules are inherently limited in their ability to address satisfactorily this type of workloads.

VI. Validation

Up to this point, we have used the same traces both to measure scheduling performance and to tune parameters, such as χ, θ, and $\bar{\ell}$. We then validated our methods on two additional traces with *no* further algorithm modification or parameter tuning. Our objective was to perform a "blind" test of our methods. We collected the trace of HTTP requests to the main Web server of the Computer Science department at Rutgers University between 12 p.m. and 4 p.m. on December 18, 1999. The other



(a) cs.edu (b) NASA

Fig. 11. Latency on the cs.edu and NASA traces expressed as number of broadcast ticks.

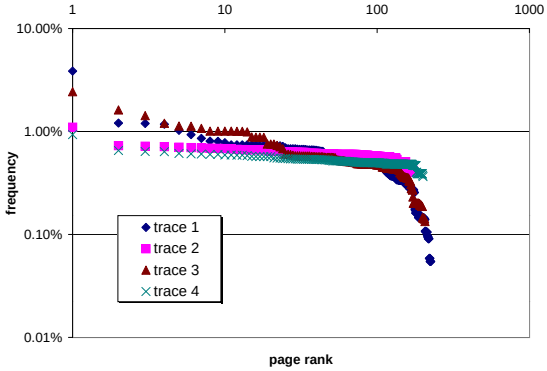


Fig. 10. Concentration of references (reference count plotted against page rank).

trace is the NASA server trace collected between 12 p.m. and 4 p.m. on August 3, 1995. Similarly to the preprocessing of the World Cup traces, we eliminated all but the $\chi = 0.07\%$ of the hottest bytes and simulated a cyclical broadcast at the rate of 256 Kbps. User-perceived latency in term of broadcast ticks is shown in Figure 11. The heavy algorithm outperformed MAD by 73% to 91%. The heavy and MST methods had seek time comparable to that of a random schedule, but reduced transfer time by a factor ranging from 4 to 7.

VII. Related Work

Broadcast scheduling has been the subject of extensive investigation, mostly under the hypothesis that pages have stationary access probabilities [12], [13], [14]. Furthermore, Ammar gives analytical expressions to estimate client waiting time in the presence of conditional access probabilities and Poisson OFF periods [38]. To the best of our knowledge, no previous work examines the case where a client requests simultaneously multiple resources. Broadcast with non-uniform page sizes has been considered as well [15]. Variable-size pages cannot be directly identified with objects in that multiple objects can have common pages. Alternative scheduling methods use a pyramid scheme that is particularly suited to streaming media [41]. Clustering through spanning trees is a well-known technique [33], which we adapt to broadcast scheduling in the presence of access pattern dependencies. Additional background was summarized in Section II and IV.

VIII. Conclusions

Discussion: Broadcast is the primary mode of operation for several wireless and optical media and leads naturally to transport and application solutions. Analogously, multicast has the potential of effectively relieving Internet hot spots, thereby leading to scalable applications. Multicast can also be used in CDN backbones and it can be employed in conjunction with caching. A critical issue in broadcast management is the organization of the broadcast. Typical broadcast schedules, such as MAD, transmit hot pages more often than colder ones. Meanwhile, clients are often interested in downloading lists of pages from the server. This scenario leads to strong dependencies in the access pattern, and if such dependencies are taken into account, substantial performance improvements are possible. We have a proposed a simple greedy algorithm (MST) for flat schedules and a refinement (Heavy) that leads to a (slightly) skewed schedule. The algorithms are fast in theory and in practice and it is easy to update their schedule if underlying dependencies change. Moreover, the algorithms give as a by-product a clustering of pages according to their access dependencies.

The algorithms were extensively analyzed on multiple Web traces. Furthermore, an additional set of two more traces was considered in order to perform a "blind" validation of algorithms, i.e., an algorithm validation without any further parameter tuning. Our final algorithm (Heavy) leads to improvements in client-perceived latency ranging from 8% to 33% on the World Cup traces, and up to 91% on the blind validations. The MST algorithm generates a flat schedule and so, in addition to improving average access time, it offers a worst-case bound on the acces time. We conclude that substantial performance improvements can be efficiently obtained by considering dependencies in the access pattern.

Methodological Implications: From a methodological perspective, we observe that most broadcast data management research has been conducted under the *independent reference* assumption: page i is requested at time t according to a stationary probability p_i that is independent of past accesses. Much research in broadcast scheduling and caching assumes independent references with stationary probabilities. We believe that the independent reference assumption is in most cases a good first order approximation that leads to valuable algorithms and to a first conceptual clarification of the problem at hand. We

also venture that substantial performance improvements and a better understanding can be derived from bringing to light the complex dependencies in data access patterns.

Future Work: The techniques in this paper have been validated in the context of multicast dissemination of Web contents. It is natural to conjecture that access pattern dependencies exist in other contexts as well, and so the algorithms in this paper should be applicable to a variety of other scenarios, as for example, wireless information stations or satellite-supported CDN's. We are actively working at extending the scope of our measurements. Furthermore, we plan to implement a platform to support data management issues for Internet data dissemination [42], which would, among other objectives, allow us to obtain more direct measurements for the performance of our algorithms and their interaction with document selection, caching, congestion control, and layering.

REFERENCES

[1] K. C. Almeroth, M. H. Ammar, and Z. Fei, "Scalable delivery of Web pages using cyclic best-effort (UDP) multicast," in *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[2] Michael Rabinovich, "Resource management issues in content delivery networks (CDNs)," in *DIMACS Workshop on Resource Management and Scheduling in Next Generation Networks*, 2001.

[3] Sanjeev Khanna and Vincenzo Liberatore, "On broadcast disk paging," *SIAM Journal on Computing*, vol. 29, no. 5, pp. 1683–1702, 2000.

[4] Vincenzo Liberatore, "Caching and scheduling for broadcast disk systems," in *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX 00)*, 2000, pp. 15–28.

[5] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proc. Sigcomm*, 1998.

[6] "http://www.digitalfountain.com/," .

[7] "http://www.hns.com/," .

[8] "http://www.panamsat.com/," .

[9] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proceedings of the 1995 ACM SIGMOD Conference International Conference on Management of Data*, 1995, pp. 199–210.

[10] Qinglong Hu, Wang-Chien Lee, and Dik Lun Lee, "Performance evaluation of a wireless hierarchical data dissemination system," in *Proc. MobiCom*, 1999.

[11] Gary Herman, Gita Gopal, K. C. Lee, and Abel Weinrib, "The datacycle architecture for very high throughput database systems," in *Proceedings of the 1987 ACM SIGMOD Conference International Conference on Management of Data*, 1987, pp. 97–103.

[12] C. J. Su and L. Tassiulas, "Broadcast scheduling for information distribution," in *Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1997)*, 1997.

[13] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber, "Minimizing service and operation costs of periodic scheduling," in *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 11–20.

[14] Claire Kenyon, Nicolas Schabanel, and Neal Young, "Polynomial-time approximation scheme for data broadcast," in *Proceedings of the Thirtisecond ACM Symposium on the Theory of Computing*, 2000.

[15] Claire Kenyon and Nicolas Schabanel, "The data broadcast problem with non-uniform transmission times," in *Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 547–556.

[16] Nicolas Schabanel, "The data broadcast problem with preemption," in *LNCS 1770 Proc. of the 17th International Symposium on Theoretical Aspects of Computer Science (STACS 2000)*, 2000, pp. 181–192.

[17] Amotz Bar-Noy, Boaz Patt-Shamir, and Igor Ziper, "Broadcast disks with polynomial cost functions," in *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, 2000.

[18] Balachander Krishnamurthy and Jennifer Rexford, *Web Protocols and Practice*, Addison-Wesley, Boston, 2001.

[19] Jayavel Shanmugasundaram, Arvind Nithrakashyap, Rajendran Sivasankaran, and Krithi Ramamritham, "Efficient concurrency control for broadcast environments," in *ACM SIGMOD International Conference on Management of Data*, 1999.

[20] R. Agrawal and P. K. Chrysanthis, "Efficient data dissemination to mobile clients in e-commerce applications," in *Proceedings of the Third IEEE Int'l Workshop on Electronic Commerce and Web-based Information Systems*, June 2001.

[21] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Energy efficient indexing on air," in *Proc. of the SIGMOD Conference*, 1994, pp. 25–36.

[22] S. Viswanathan T. Imielinski and B.R. Badrinath, "Data on air: Organization and access," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 3, pp. 353–372, 1997.

[23] Sanjeev Khanna and Shiyu Zhou, "On indexed data broadcast," in *Proceedings of the Thirtieth ACM Symposium on the Theory of Computing*, 1998, pp. 463–472.

[24] E. Pitoura and P. K. Chrysanthis, "Exploiting versions for handling updates in broadcast disks," in *Proc. of the 25th Int'l Conference on Very Large Data Bases*, Sept. 1999, pp. 114–125.

[25] Swarup Acharya, Michael Franklin, and Stanley Zdonik, "Balancing push and pull for data broadcast," in *ACM SIGMOD International Conference on Management of Data*, 1997.

[26] W. Richard Stevens, *Unix Network Programming*, PTR PH, 1998.

[27] S. Deering and R. Hinden, "Internet Protocol, version 6 (IPv6) specification," RFC 2460, 1998.

[28] Michael R. Garey and David S. Johnson, *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif., 1979, A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[29] Mark D. Hansen, "Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 604–609.

[30] R. Ravi, Ajit Agrawal, and Philip Klein, "Ordering problems approximated: single-processor scheduling and interval graph completion," in *Automata, languages and programming (Madrid, 1991)*, pp. 751–762. Springer, Berlin, 1991.

[31] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber, "Divide-and-conquer approximation algorithms via spreading metrics," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, Oct. 1995, pp. 62–71.

[32] Satish Rao and Andréa W. Richa, "New approximation techniques for some ordering problems," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998)*, New York, 1998, pp. 211–218, ACM.

[33] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, *Network flows*, Prentice Hall Inc., Englewood Cliffs, NJ, 1993, Theory, algorithms, and applications.

[34] Kurt Mehlhorn and Stefan Näher, *LEDA*, Cambridge University Press, Cambridge, 1999, A platform for combinatorial and geometric computing.

[35] Martin Arlitt and Tai Jin, "Workload characterization of the 1998 World Cup web site," Tech. Rep. HPL-1999-35R1, HP Labs, 1999.

[36] K. Stathatos, N. Roussopoulos, and J. S. Baras, "Adaptive data broadcast in hybrid networks," in *Proc. 23rd International Conference on Very Large DataBases*, 1997, pp. 326–335.

[37] Paul Barford and Mark Crovella, "A performance evaluation of hypertext transfer protocols," in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1999, pp. 188–197.

[38] M. H. Ammar, "Response time in a teletext system: An individual user's perspective," *IEEE Transactions on Communication*, vol. COM-35, no. 11, pp. 1159–1170, Nov. 1987.

[39] S. Battacharyya, J. F. Kurose, D. Towsley, and R. Nagarajan, "Efficient rate controlled bulk data transfer using multiple multicast groups," in *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[40] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like congestion control for layered multicast data transfer," in *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[41] S. Vishwanath and T. Imielinski, "Pyramid broadcasting for video on demand service," Tech. Rep. DCS-TR-311, Rutgers, 1994.

[42] Panos K. Chrysanthis, Vincenzo Liberatore, and Kirk Pruhs, "Middleware support for multicast-based data dissemination: A working reality," White paper, 2001.