Mobile Networks and Applications 7, 279–290, 2002

High Performance Data Broadcasting Systems *

PETER TRIANTAFILLOU, R. HARPANTIDOU and M. PATERAKIS
Department of Electronic and Computer Engineering, Technical University of Crete, Chania, 73100, Greece

Abstract. Data broadcasting as a means of efficient data dissemination is a key technology facilitating ubiquitous computing. For this reason, broadcast scheduling algorithms have received a lot of attention. However, all existing algorithms make the core assumption that the data items to be broadcast are immediately available in the transmitter's queue, ignoring the key role that the disk subsystem and the cache management play in the overall broadcast system performance. With this paper we contribute a comprehensive system's perspective towards the development of high performance broadcast systems, taking into account how broadcast scheduling, disk scheduling, and cache management algorithms affect the overall performance. We contribute novel techniques that ensure an efficient interplay between broadcast scheduling, cache management, and disk scheduling. We study comprehensively the performance of the broadcast server, as it consists of the broadcast scheduling, the disk scheduling, the cache management algorithms, and the transmitter. Our results show that the contributed algorithms yield considerably higher performance. Furthermore, one of our algorithms is shown to enjoy considerably higher performance, under all values of the problem and system parameters. A key contribution is the result that broadcast scheduling algorithms have only a small effect on the overall system performance, which necessitates the definition of different focal points for efforts towards high performance data broadcasting.

Keywords: data dissemination, broadcast scheduling, disk scheduling, cache management, system performance

1. Introduction

Mobile computing and wireless networks are quickly evolving technologies that are making ubiquitous computing a reality. As the population of portable wireless computers increases, mechanisms for the efficient dissemination of information to such wireless clients are of significant interest. Such mechanisms could be used by a satellite or a base station to disseminate information of common interest. Many emerging applications involve the dissemination of data to large populations of clients. Examples of such dissemination-oriented applications include information dispersal systems for volatile time-sensitive information such as stock prices and weather conditions, news distribution systems, traffic information systems, electronic newsletters, software distribution, hospital information systems, public safety applications, and entertainment delivery.

Many of the dissemination-oriented applications we mention above, have data access characteristics that differ significantly from the traditional notion of client–server applications as embodied in navigational web browsing technology. A fairly limited amount of data is distributed from a small number of sources to a huge client population (potentially many millions) that have overlapping interests, meaning that any particular data item is likely to be distributed to many clients. Data broadcasting is considered to be an efficient way, in terms of bandwidth and energy, for the distribution of such information for both wireless and wired communication environments and has been extensively studied (see [1,9,18,22]). Furthermore, broadcast transmission compared to traditional unicast can be much more efficient for disseminating information, because unicast by having to transmit every data item, often identical, at least once for every client who requests it, creates scalability problems as the client population increases. Thus, a key advantage of broadcast delivery is its scalability: it is independent of the number of clients the system is serving. Much of the communication technology that has enabled large-scale dissemination supports broadcast, and in some cases, is primarily intended for broadcast use. For instance, direct broadcast satellite providers, and cable television companies (through the use of high-bandwidth cable modems) are now, or will soon be, capable of supporting multi-megabit per second data broadcast. Intel has also been broadcasting data along with normal TV signals [13].

1.1. Related work

Broadcasting as a mean of efficient data dissemination is receiving increasingly higher attention [8,11,17]. The problem of determining an efficient broadcast schedule for information distribution systems, in particular, has been extensively studied in the past [1–4,9,18–21]. In [4] the authors propose the RxW scheduling algorithm which calculates the product of the number of outstanding Requests (R), times the Wait time (W) of the oldest outstanding request for all data items corresponding to the requests pending in the broadcast server queue. The data item with the highest product value is chosen for broadcast. Therefore, RxW broadcasts a data item either because it is very popular (high R value) or because it has at least one long-outstanding request. It provides a balanced treatment of requests for both popular (hot) and not-so-popular (cold) items.

* An abridged, earlier version of this paper, based only on preliminary descriptions of the algorithms and results of sections 2 and 3 appeared in the 2nd Mobile Data Management Conference, MDM01.

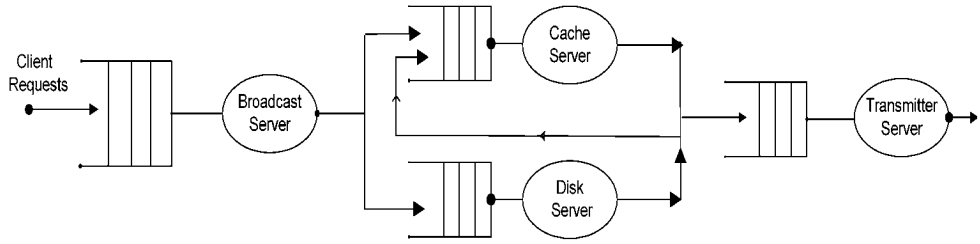


Figure 1. Broadcast server architecture.

In [3], Acharya and Muthukrishman study the scheduling problem arising in on-demand environments for applications with data requests of varying sizes and they introduce an alternative to the response time of a request metric- the stretch of a request, which seems better suited to variable-sized data items. They present an algorithm called MAX based on the criteria of optimizing the worst case stretch of individual requests.

Other efforts, such as the ones in [6,20] assume that cache memory is available at each user. The management of this memory was considered in order to reduce the mismatch between the push-based broadcast schedule and the user's access pattern. In [19], the authors consider the problem of scheduling the data broadcast such that the access latency experienced by the users is minimized. Push-based and pull-based systems are considered. Vaidya and Hammed [21] propose algorithms for broadcast schedules in asymmetric environments that minimize wait time. Variations of those algorithms for environments subject to errors, and systems where different clients may listen to different number of channels are also considered.

1.2. Problem motivation, formulation, and system model

The abundance of dissemination-based applications caused the rapid development of scheduling algorithms for data broadcast. All such algorithms attempt to select which item to broadcast in order to improve performance. The root implicit or explicit assumption of all existing broadcast scheduling algorithms is that the data items to be broadcast are immediately available to the transmitter (see, for instance, [2–4,19,21, and the references therein][1]). This assumption ignores the fact that in many cases these data items must be retrieved from secondary storage before they can be broadcasted. They also typically ignore the existence of the broadcast server's cache and the related cache management issues. By ignoring these issues, such scheduling algorithms when used in real systems can cause significant degradation of the broadcast efficiency, or at the very least the reported results, regarding the efficiency of proposed broadcast scheduling algorithms, are misleading. In our paper, we take into account the fact that broadcast scheduling, disk scheduling, and cache management algorithms affect the performance of each other and the overall performance of the broadcast server.

With this paper we put forward a comprehensive study from a systems' viewpoint of the problem of pull-based broadcast scheduling. We consider a broadcast server with the architecture shown in figure 1. All client requests enter into the broadcast server queue. The requests may need service from the disk server or alternatively, when cache memory exists, the data items may be found in the cache (i.e., they had been retrieved earlier from disk) and they are forwarded directly to the transmitters' queue. From the transmitters' queue all data items are transmitted through the communication channel, reaching the clients that had made the corresponding requests. When a data item is broadcasted, all requests for the particular data item are satisfied simultaneously regardless of the time of their arrival.

The system we study consists of a large and possibly time varying client population that requests data items from an information source equipped with a data broadcasting capability. Clients use two independent channels for communicating with the server: an uplink channel for sending requests to the server, and a "listen only" downlink channel for receiving data. When a client needs a data item (e.g., a database object) that cannot be found locally, it sends a request to the server. Client requests are queued up at the broadcast server upon arrival. Requests that correspond to the same item are grouped together forming a multi-request. In the remainder of the paper, we refer to such multi-requests as requests.

We make the following assumption. First, we assume, for simplicity reasons only, that data items are of fixed-length (e.g., database objects). Second, we assume that clients continuously monitor the broadcast channel after they send a request to the server and we do not consider the effects of transmission errors, so that all clients waiting for a data item receive it when it is broadcasted by the server. We ignore the delay for sending requests via the client-to-server uplink, which we expect to be small compared to the latency of obtaining broadcasted items from moderately or heavily loaded servers.

1.3. Overview of contributions

In our system model users may place requests about information items directly to the broadcast server, and the broadcast scheduler works together with the disk scheduler and the cache manager to decide which of the requests will be serviced next. The study of the interplay of these components,

[1] We have recently become aware [10] that the authors of the RxW technique are studying issues extending RxW with cache management and disk accesses, with a focus to further improve the performance of RxW.

which constitute the necessary infrastructure of a broadcast server is, to the best of our knowledge, a novel contribution. Another novel contribution of the work in the paper is the study of the end-to-end performance of a system that is comprised of a broadcast scheduler, a cache manager, a disk scheduler, and a transmission scheduler with a transmission channel.

Our study is "comprehensive" in that it considers the interplay between the broadcast scheduling algorithm, the disk scheduling algorithm, and the cache management scheme. The high level contributions of this work are:

- We propose mechanisms that ensure the required interplay of the above algorithms in order to ensure high performance. These mechanisms consist of four novel scheduling algorithms and a cache management scheme.
- We show that without such mechanisms the algorithms for broadcast scheduling found in the literature can be of little practical use.
- We conduct a detailed performance study of the broadcast system: we quantify the expected performance under different values of the problem parameters and we identify the critical mechanisms that limit performance under different configurations.

1.4. The remainder of the paper

The remainder of the paper is organized as follows. In section 2, we describe the scheduling algorithms ADoRe, FLUSH, OWeiST, RxW/S that we propose. In section 3, we present the simulation model, the performance metrics and the performance behavior of these algorithms. In section 4, we focus on cache management issues, we discuss cache replacement algorithms, and we present performance results for the algorithms of section 2 when used in conjunction with the cache replacement algorithms. We also present results regarding the efficiency of the cache replacement algorithms. In section 5, our focus includes the transmitter and we present the relevant performance results. Finally, we conclude this paper in section 6.

2. Integrating broadcast and disk scheduling

The broadcast scheduling algorithm that we have chosen from the related literature is the exhaustive RxW algorithm, which appears to be a practical, low-overhead, scalable scheme that requires no *a priori* advanced knowledge (such as the access probabilities of items, etc.) [4]. Our group has performed performance studies comparing RxW with other algorithms (e.g., the algorithms in [3,21]), and we have found it to have the best performance. These are the reasons we have chosen it as the basic broadcast scheduling algorithm.

For disk scheduling we selected the C-LOOK algorithm, which has been found to have very good performance [23], unless stated otherwise. The C-LOOK algorithm sorts data items to be retrieved from the disk in ascending order of their cylinder position on the disk. The read–write head is only moved as far as the last request in each direction. It services requests from the service queue as it reaches each cylinder, until there are no requests in the current direction. Then, it immediately returns to the first requested item of the other end, without servicing any requests on the return trip, and repeats the process.

In this section, we disregard the existence of a cache and we assume that each requested data item must be retrieved from the secondary storage. This is done for two reasons: first in order to measure the impact of the disk subsystem in the performance of the server, and second, because in applications where the data items will be very large and/or the distribution of the requests to data items will not be skewed, the cache will have little impact. Furthermore, in some system configurations the cache size might be quite small. Consider, as an example, a broadcast server running on a network-attached disk, which has a cache size (nowadays 8–16 MB) that is a negligible percentage of the database size. In later sections we will study the impact of the cache.

The mechanisms we propose are classified in two categories: (i) those that combine separate, "off-the-shelve" broadcast and disk scheduling algorithms, and (ii) those that are based on a single scheduling criterion that takes into account information that is important to both broadcast and disk scheduling algorithms.

2.1. Combining separate broadcast and disk scheduling algorithms

The first of the algorithms we present below extend and combine the RxW broadcast scheduling algorithm with a disk scheduling algorithm through various mechanisms. The second uses FCFS, as broadcast scheduling algorithm. Both algorithms depend on the C-LOOK disk scheduling algorithm.

2.1.1. The ADoRe algorithm: Active disk on requests

A straightforward implementation of the RxW algorithm in our setting would imply that each time it is called the broadcast scheduling algorithm forwards a single request to the disk system and once the item is retrieved and broadcasted, RxW is called again to pick the next item, and so on. This, obviously, results in poor disk subsystem performance, since in essence the items are retrieved from disk in a random order. The ADoRe algorithm attempts to avoid the performance shortcomings of RxW. It reduces the average disk service time by forwarding a group of requests to the disk system, facilitating the disk scheduling algorithm's optimization to produce better results. For example, running RxW once to pick $K = 10$ items and invoking a C-LOOK sweep with $K = 10$ requests will take less time than running the RxW algorithm 10 times, each time picking one data item and waiting until it is retrieved from the disk. At the same time, ADoRe keeps the disk system highly utilized, by forwarding requests to it when it becomes idle, even if fewer than K requests exist in the broadcast queue.

ADoRe is a fairly simple algorithm, that is based on a generalization of RxW. It works exactly as follows: when the disk becomes idle, K requests are directed from the broadcast server queue to the disk scheduler queue to be served. If there are more than K outstanding requests in the broadcast server queue, the RxW algorithm is applied and a group of K requests with the highest RxW values are selected and handed to the disk. If there are fewer than K requests in the broadcast queue, then all requests are passed to the disk.

Therefore, if K equals 1 the ADoRe algorithm becomes a straightforward implementation of the RxW algorithm since it directs one request from the broadcast server queue, by applying the RxW algorithm, to the disk server queue every time the broadcasting of the previously selected data item is completed.

2.1.2. The FLUSH algorithm

The performance gains of ADoRe come from higher disk utilization and higher disk scheduling optimizations, while still trying to exploit the performance benefits of RxW, particularly its fairness achieved through the W component in the scheduling criterion. The key idea for the development of FLUSH is to pursue more aggressively higher disk utilizations and higher disk scheduling optimizations, at the expense of any benefits contributed by RxW. The performance evaluation of FLUSH will help us weigh the benefits of RxW against those due to higher disk subsystem performance.

The FLUSH algorithm uses also the C-LOOK disk scheduling algorithm, however, FLUSH manipulates differently the requests on the broadcast server. Every time the disk finishes the service of a single request, all the requests in the broadcast queue are flushed to the disk system and incorporated into the C-LOOK scan lists. Thus, FLUSH in essence employs no scheduling algorithm at the broadcast queue. The end result is that longer queues (scan lists) are formed at the disk, which results in even higher disk utilizations and possible disk scheduling optimizations.

2.2. Amalgamating broadcast and disk scheduling algorithms

The amalgamated algorithms combine information available at the broadcast server and at the disk server, producing a single scheduling criterion that takes into account factors that have been found to improve the performance of broadcast and disk scheduling. The obvious motivation for studying such algorithms is to verify whether the single scheduling criterion they propose can lead to higher performance, compared to that of mechanisms that simply effectively combine separate disk and broadcast scheduling algorithms.

2.2.1. The OWeiST algorithm: Optimal Weighted Service Time

The OWeiST algorithm attempts to improve performance in two ways. First, it exploits information available at the broadcast server and at the disk server. Second, it employs a different disk scheduling algorithm, which for larger groups of requests, introduces further optimizations. The additional motivation is to study whether more sophisticated disk scheduling algorithms will result in higher performance.

According to the OWeiST algorithm whenever the disk becomes idle, K or fewer requests as in the ADoRe algorithm, are being selected from the broadcast queue and forwarded to the disk queue. The service of the requests is being carried out in such order that the sum of the products R times Disk Service Time of the requests is kept minimum. The operation of the algorithm is as follows. We maintain a graph of K requests. The edge connecting two requests r_i and r_j has a label $R_j \times S_j$ where R_j is the number of requests in the broadcast queue for item j and S_j represents the disk access cost to access item j, given that the previously retrieved item from the disk was item i. The disk access cost contains the seek time from the cylinder of item i to the cylinder of item j, plus the rotational delay necessary to access item j once the disk head is positioned on j's cylinder, plus the time to retrieve item j from the disk. The algorithm computes all possible permutations for the K requests and selects the optimal permutation that gives the smallest total weighted cost. Therefore, OWeiST minimizes the quantity

$$\sum_{i=1}^{K}(R_i \times S_i).$$

Note that, obviously, this is analogous to computing a solution to the Traveling Salesman Problem (TSP). However, by bounding the value of the parameter K we can control the overhead involved in computing the optimal service schedule. Notice that, when K equals 1, OWeiST is identical to ADoRe with K equal to 1.

2.2.2. The RxW/S algorithm

We also propose the amalgamated algorithm RxW/S, where S is the disk service time, which takes into account information of the broadcast scheduling algorithm (i.e., R and W for every requested data item) and the disk service overhead. RxW/S is similar to the "Relief" scheduling algorithm proposed in the literature [12] in the context of scheduling video tape accesses in tertiary storage tape libraries. It is a one-step algorithm, which is being activated whenever the disk becomes idle and selects a request from the broadcast queue to be serviced from the disk. The selected request is the one that has the higher value of the expression $(R \times W)/S$. In this algorithm there is no grouping of requests because requests are directed to the disk one at a time. The selection of the data item to be broadcasted, favors items with high $R \times W$ values (as the RxW algorithm suggests) and low disk access times (as our comprehensive view of the problem requires).

3. Performance study

The performance of the algorithms has been studied through simulation. The simulations were executed on typical Pentium III PCs. Each run simulates the transmission of one million data items. We observed, that by simulating 1,000,000

Table 1
Disk characteristics.

Cylinders	6,900
Surfaces	12
Sector size	512
Revolution speed	10,000 rpm
Number of zones	20
Average transfer rate	12 MBps

transmitted data items, we were able to estimate with accuracy the steady-state algorithms performance.

Table 1 shows the parameter setting for the simulated disk system. We assume that the database consists of 10,000 16 KB (or alternatively 200 KB) data items.

3.1. Simulation model

We developed a model, as depicted in figure 1. We used a Request Generator, which generates a stream of requests according to a Poisson arrival process. The request arrival rates we used in our simulations vary between 10 and 500 requests per second.

Requests are generated from the Request Generator and then they enter the broadcast server queue. By the application of the algorithms in section 3, requests are directed to the disk server queue where the corresponding data items are retrieved from disk and are then forwarded to the transmitters' queue (see section 3.3). When a cache exists, all the data that are forwarded to the transmitters' queue are first moved from disk to the cache (provided that they are not already located in the cache). The transmitter conveys all the data items to the clients through the channel link. Finally, the statistics collector records all relevant statistics in order to measure the performance.

In our simulation, it is assumed that the request probabilities of all data items follow a Zipf distribution. The Zipf distribution may be expressed as follows:

$$p_i = c\left(\frac{1}{i}\right)^{\theta}, \quad 1 \leqslant i \leqslant M,$$

where $c = 1/\sum_{i=1}^{M}(1/i)^{\theta}$ is a normalizing factor, and θ is a parameter referred to as the access skew coefficient. The distribution becomes increasingly "skewed" as θ increases [15,21]. We will report results for two values for θ; $\theta = 0$ (uniform distribution), and $\theta = 1.17$ (highly skewed access distribution).

3.2. Performance metrics

In client–server information systems, the user response time, namely the time between the arrival of the request at the broadcast server and its service, is one of the most important factors for evaluating the systems' performance. A metric that gives an overall view of the response time of all clients in the system is the mean response time. As is remarked in [15], it is natural in the real world some users' demand patterns to completely differ from the overall demand pattern and their own response time may be much worse than the overall mean. In this paper, we address this problem by adopting as a performance metric an *index of response time fairness* that always lies between 0 and 1 [14]. We have chosen this metric over the square coefficient of variation of the response time, since the fairness index is a further normalization giving a number between 0 and 1. This boundedness aids intuitive understanding of the fairness index. For example, an algorithm with a fairness index of 0.10 means that it is unfair to 90% of the users, and an algorithm with a fairness index of 0.90 means that is fair to 90% of the users. The fairness index, if n contending users are in the system such that the response time of the request of the ith user is denoted by x_i, is defined as follows:

$$f(x) = \frac{[E[x]]^2}{E[x^2]} = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n\sum_{i=1}^{n} x_i^2}, \quad x_i \geqslant 0,$$

where x is the random variable denoting the response time of a client's request. We have also considered the ratio of the standard deviation to the mean response time of a request (i.e., the square root of the square coefficient of variation), as an additional fairness indicator.

3.3. Performance results

The results we present below are a representative sample of the results we have obtained. We conducted a number of experiments under different combinations of the arrival rate, the grouping parameter K (for algorithms ADoRe and OWeiST), and the access skew coefficient θ. The two primary performance metrics, the mean response time and the fairness index, are plotted versus the request arrival rate λ, for different values of the parameter K. The CPU overhead for the RxW algorithm and for calculating the optimal permutation of the OWeiST algorithm is not included in the mean response time result. However, the CPU time (overhead) of the application of the RxW algorithm on the broadcast server queue was estimated to be less than 1 ms and the corresponding time for the disk scheduling mechanism of the OWeiST algorithm for figuring out the optimal permutations when K equals 5 was estimated to be approximately 1 ms. Hence, these costs are considered negligible.

Figures 2–5 present the results we obtain under the assumptions of no cache memory available and of infinite broadcast channel speed. As a result of these assumptions, the response time of a client's request corresponds to the time between the arrival of the request at the server and the retrieval of the corresponding data item from the disk.

Figures 2–4 present the mean response time of the proposed scheduling algorithms, ADoRe, FLUSH, OWeiST, and RxW/S. As we mentioned in section 2, the ADoRe algorithm with K equal to 1 resembles the RxW algorithm. Figure 2 presents the mean response time (in ms) versus the arrival rate λ, for $\theta = 1.17$, 16 KB data item size, and $K = 1$ (which means that each group of the requests contains 1 element). We observe that as the aggregate request arrival rate increases beyond 50 ($\lambda > 50$), the mean response time of the

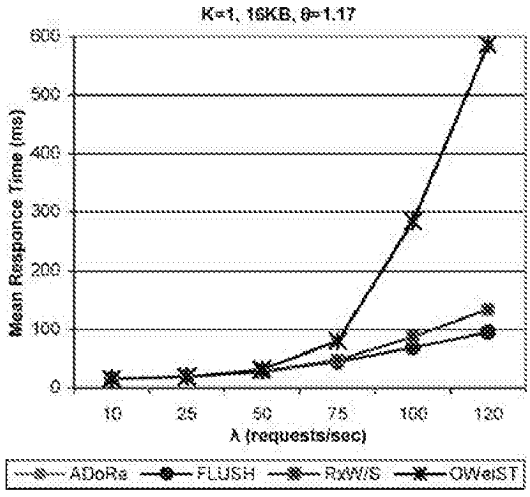


Figure 2.

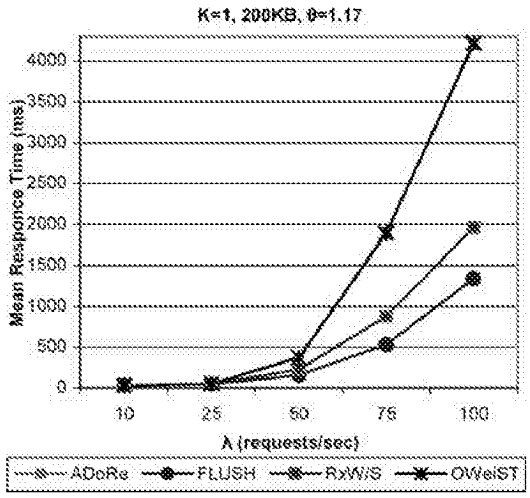


Figure 3.

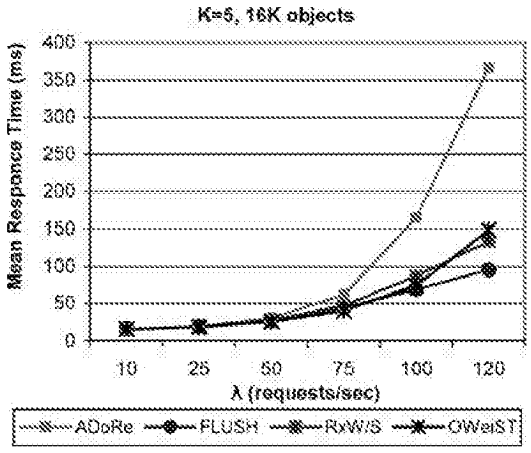


Figure 4.

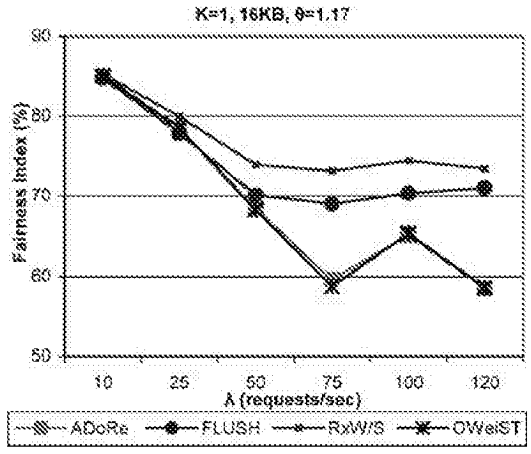


Figure 5.

ADoRe, and the OWeiST is sharply increased, reaching the value of 590 ms for arrival rate $\lambda = 120$ requests/s. On the contrary, the mean response time is maintained low for the FLUSH, and the RxW/S algorithms, with mean response time values less than 130 ms for the RxW/S and less than 100 ms for the FLUSH, with arrival rate $\lambda = 120$ requests/s. This demonstrates that the latter two algorithms perform considerably better. The same trend is observed for 200 KB data item size, as is shown in figure 3. The difference in the mean response time is due to the retrieval time from the disk of the 200 KB data item, which is a multiple of the corresponding time of the 16 KB data items.

In figure 4 the value of the parameter K increases to 5 and we notice similar behavior. Notice that K refers only to ADoRe and OWeiST algorithms, while the curves of FLUSH and RxW/S are the same as in figure 2. OWeiST performs much better than before, and the mean response time is roughly similar to that of the FLUSH and the RxW/S for all λ values examined. This performance improvement of the OWeiST was expected, since its optimization (i.e., the calculation of the shortest path for visiting all K data items on the disk in accordance to their popularity) introduces obviously greater benefits as K increases. ADoRe with K equal to 5 performs slightly better than with K equal to 1, because the RxW algorithm forwards a group of requests to the disk and not just one at a time.

The improved performance of FLUSH over the other three algorithms, as λ increases, shown in figures 2–4 was expected since FLUSH forwards requests to the disk server as they arrive, even before the disk system has finished the previous group of requests, increasing the number of requests that are waiting to be serviced in the disk queue. This gives the disk scheduling algorithm (C-LOOK) the chance to further optimize the disk access time. The OWeiST algorithm cannot do that since given the NP-Completeness of calculating the optimal schedule, the disk system queue must be relatively small. The critical observation is that the broadcast server must keep the disk server as busy as possible and this outweighs in importance any improvement from the other disk scheduling algorithms.

Figure 5 shows the fairness index plotted versus the request arrival rate λ for the case of 16 KB data item size, and K equal to 1. We observe that FLUSH and RxW/S maintain the fairness index above 70% even for increased arrival rates, while the fairness indices of ADoRe and OWeiST drop

abruptly. Even for 200 KB data item sizes (for space reasons these results are not shown), FLUSH maintains the fairness index around 70% when the fairness indices of the remaining algorithms drop well below 50%.

As an additional fairness metric we have examined the ratio of the standard deviation to the mean response time. FLUSH maintains the value of this ratio below 0.7 for all values of the arrival rate, and data item sizes we examined. For completeness purposes, we also simulated the demand for data items using the Uniform distribution ($\theta = 0$). As expected, we observed that the mean response time of all algorithms for $\theta = 0$, increases rapidly as the arrival rate increases (compared to the corresponding results in figures 2, 3, and 4). This is due to the decreasing possibility of serving more than one client by a single broadcast. FLUSH, however, continues to perform best and maintains a fairness index above 65%.

4. Studying the impact of the cache and the cache replacement algorithms

In more skewed access distributions, some data items are more popular and thus it is reasonable to keep them in cache memory, since we can thus reduce the average cost for fetching data items from secondary storage. We present a cache replacement mechanism, which combines both the Least Recently Used (LRU) and the Least Frequently Used (LFU) policies. The LRU and the LFU cache replacement policies are popular due to their simplicity and efficiency. The combination of those two policies results in a policy that is superior to them as well as to other more efficient replacement policies that have been previously suggested (such as the LRU-k algorithm [16]).

4.1. The cache replacement algorithms

The LF-LRU algorithm. LF-LRU operates as follows. It uses a cache buffer, with a given capacity in data items. The algorithm operates using two sorted lists, an LRU and an LFU list. The data items are entering the buffer being placed at the top of the LRU list. For every item that enters the buffer, the algorithm maintains a counter of the number of its references during the time it stays in the buffer. Upon a reference of a data item, the algorithm first checks whether the requested item is in the cache memory. If it is, the reference counter is incremented and the data item is moved from its present position in the LRU list to the top of the LRU list. In the case that the requested data item is not in the cache memory, the missed item is fetched from disk, placed at the top of the LRU list and its reference counter is initialized to 1.

LF-LRU maintains also the LFU list; a sorted list of all cached items in non-decreasing order of reference counter values. The replacement policy of LF-LRU dictates that the newly fetched item must replace in the cache memory the item that has the smallest reference counter. Among the items that tie for the smallest counter value, the item that is closest to the bottom of the LRU list is removed. Every time a data item leaves the cache memory the corresponding counter is erased. This makes the mechanism adaptable to abrupt changes in the interests of the client population.

The LRU-k algorithm. The motivation behind this algorithm [16] is to overcome the well-known shortcoming of LRU in that it does not incorporate any knowledge of the history of accesses to data items, which would lead to the ability to identify the truly hot objects. LRU-k reduces to the well-known LRU method when $k = 1$. The basic idea in LRU-k is to keep track of the times of the last k references to memory data items, and to use this statistical information to rank-order the data items as to their expected future behavior. More formally, given a reference string known up to time t, $r_1, r_2, \ldots, r_t$, the "backward k-distance" denoted $b_t(p, k)$ is the distance backward to the kth most recent reference of data item p:

$$B_t(p,k) = \begin{cases} x, & \text{if } r_{t-x} \text{ has the value } p \\ & \text{and there have been exactly} \\ & k-1 \text{ other values } i \\ & \text{with } t - x < i \leqslant t, \text{ where } r_i = p, \\ \infty, & \text{if } p \text{ does not appear at least } k \text{ times} \\ & \text{in } r_1, r_2, \ldots, r_t. \end{cases}$$

The data item p to be dropped is the one whose backward k-distance, $b_t(p, k)$, is the maximum of all data items in the buffer. It is obvious that the larger the k-value is, the greater the efficiency of the algorithm will be; but also the greater will be the administration cost for maintaining all these references. The designers of LRU-k suggested that a value of $k = 2$ (i.e., LRU-2) will probably be a very good compromise for most environments.

4.2. Performance results

The system including cache memory was simulated as depicted in figure 1. We have studied the performance of the algorithms in section 2 as it is modified by the existence of the cache. We have examined both the LF-LRU algorithm as well as the LRU-2, and LRU-10 algorithms. We obtained results for many cache memory sizes in the range of 0.03% to 20% of the database size, but we only present some representative results for space reasons. We used the Zipf distribution for the simulation of the demands for various data items by the clients, with a variety of different values for the access skew coefficient θ, ranging from $\theta = 0$ to $\theta = 1.17$, but we only present the case of $\theta = 1.17$. The data item sizes that we had studied are 16 KB and 200 KB. We assume that a request for a data item that is located in the cache memory is immediately serviced with zero response time, since the time to read an item from the cache memory is negligible compared to the time of the retrieval of the same item from the disk.

Figure 6, shows the mean response time behavior of ADoRe ($K = 1$), FLUSH, and RxW/S algorithms when a cache memory of size 5% of the database exists, in the case of 16 KB data item size, and $\theta = 1.17$ for different arrival rates λ. As we notice, the algorithms perform almost the same

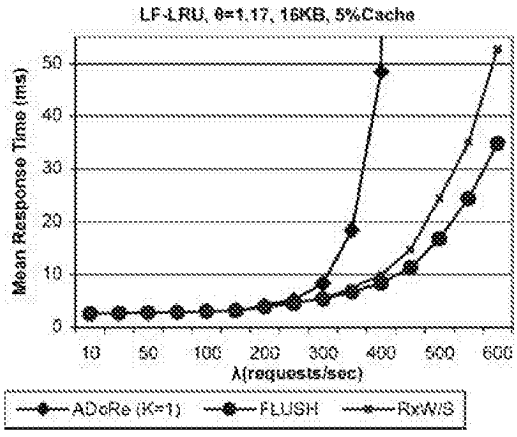


Figure 6.

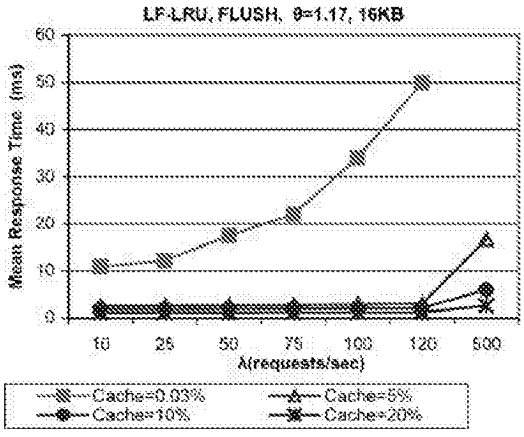


Figure 7.

for arrival rates less than or equal to 300 requests/s. When $\lambda = 350$ requests/s, ADoRe's mean response time increases more rapidly than the other two algorithms. As λ increases to 400 requests/s the mean response time of ADoRe is almost 50 ms, while FLUSH and RxW/S have response times less than 10 ms. As λ increases further, the mean response time of ADoRe exceeds 1.7 s, while that of FLUSH is around 35 ms and of RxW/S around 53 s. Since, FLUSH performs better than the other two algorithms in the remaining of the paper we use this algorithm.

Figures 7 and 8 present the mean response time, and fairness behavior of the FLUSH algorithm, in the case of 16 KB data item size and $\theta = 1.17$ for several different cache sizes. Figure 7 presents the mean response time (in ms) versus the request arrival rate λ for cache sizes equal to 0.03%, 5%, 10% and 20% of the database. We observe that even in the case of the smallest cache memory of size equal to 0.03% of the database, the mean response time decreases by almost 50% compared to the corresponding results in figure 2. As the cache size increases to 5% of the database or higher, the mean response time is reduced drastically. For arrival rates less than 120 requests/s the mean response time is kept below 3 ms, and as the load increases it maintains values less than 20 ms for cache size equal to 5% of the database and below 10 ms for larger cache sizes.

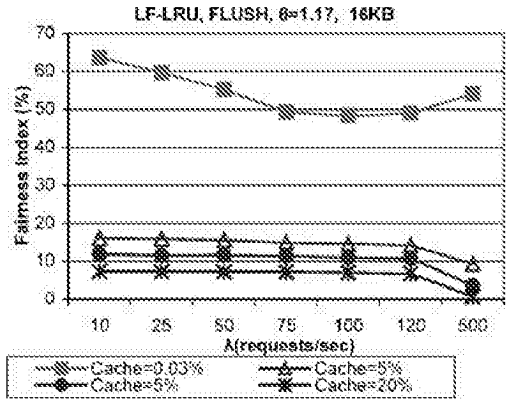


Figure 8.

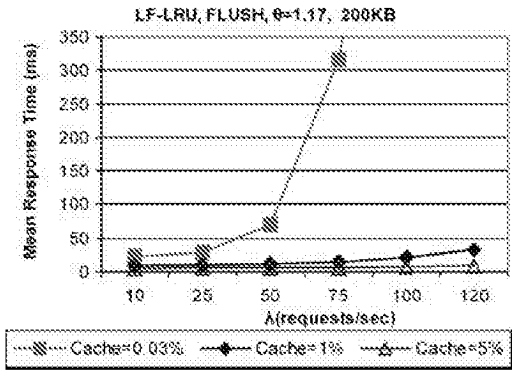


Figure 9.

Figure 8 shows the fairness index of FLUSH versus the arrival rate λ for four different cache sizes. We observe that only when the cache size is less than or equal to 0.03% of the database the algorithm maintains a high degree of fairness. For larger cache sizes the fairness of the algorithm drops. This is expected, due to the fact that there are now fewer data items which continue to be retrieved from the disk in order to be transmitted, in contrast to the increased number of data items that are found in the cache and they are immediately transmitted from there. This creates a larger variance in the response time, which is reflected in the fairness index. Note that this is a fundamental performance characteristic of larger caches and affects all algorithms, not just FLUSH.

Figure 9 shows the behavior of FLUSH with 200 KB data item size, and cache memory. We observe that in comparison to figure 3, the response time of the requests has been reduced more than 35%, even for the small cache size of 0.03% of the database. As it is expected, fairness is maintained only when the cache is small and the algorithm is not fair when the cache size becomes larger (the obtained results are similar to those in figure 8, and are thus not shown). From figures 7, 8, and 9, we conclude that the enlargement of the cache size causes significant reduction of the mean response time, at the cost of an expected sacrifice with respect to the fairness of the algorithm.

We have also simulated FLUSH for the case of $\theta = 0$ (Uniform distribution), and we have noticed that when the cache size is very small (0.03% of the database), the system

Table 2
Cache = 0.03%, FLUSH, $\theta = 1.17$, 16 KB.

λ (requests/s)	LF-LRU (ms)	LRU-2 (ms)	LRU-10 (ms)
10	10.96	12.04	11.13
25	12.26	13.66	12.48
50	15.57	17.95	15.99
75	22.05	26.01	22.57
100	34.07	40.43	34.66
120	49.95	57.36	50.74

Table 3
Cache = 5%, FLUSH, $\theta = 1.17$, 16 KB.

λ (requests/s)	LF-LRU (ms)	LRU-2 (ms)	LRU-10 (ms)
10	2.56	2.67	2.35
25	2.65	2.74	2.40
50	2.76	2.85	2.48
75	2.85	3.00	2.58
100	2.99	3.15	2.68
120	3.10	3.28	2.81

behaves as if there is no cache since the objects in the cache are replaced continuously because of the fact that the object popularities are uniformly distributed. As the request arrival rate and the cache size increase the mean response time decreases, because the augmented load increases the possibility of requests for objects placed in the cache. The fairness of the system is abated, but even for cache size equal to 20% of the database, a fairness index higher than 50% is achieved.

LF-LRU versus LRU-k. In the remainder of this section we present the results comparing the relative performance (mean response time) of the cache replacement algorithms: LF-LRU, LRU-2, and LRU-10 for very small and large caches. We focus on skewed access distributions since they are the most interesting. We show the results for smaller objects (the results for larger objects follow the trends shown for cache size).

Looking at table 2 we conclude that for small caches LF-LRU outperforms LRU-2, across all workloads. The performance gain is from 10 to 20% for lighter and heavier workloads, respectively. LRU-10 is worse by only a few percent and thus has almost identical performance to LF-LRU.

Table 3 shows that even for large caches, LF-LRU continues to outperform LRU-2, only now by a smaller margin, while all the additional information kept by the LRU-10 algorithm seems that it starts paying off, outperforming LF-LRU by a margin of about 10%. (Note, however, that LRU-10 pays an increased administration cost, which prompted the authors of LRU-*k* to suggest that LRU-2 should be selected due to its reduced administrative cost.)

5. Transmission channel impact consideration

The estimation of the end-to-end system delay is a very strong indication of the overall efficiency of the proposed broadcast scheduling mechanisms. We have simulated three different speeds for the transmission channel namely, 2 Mbps

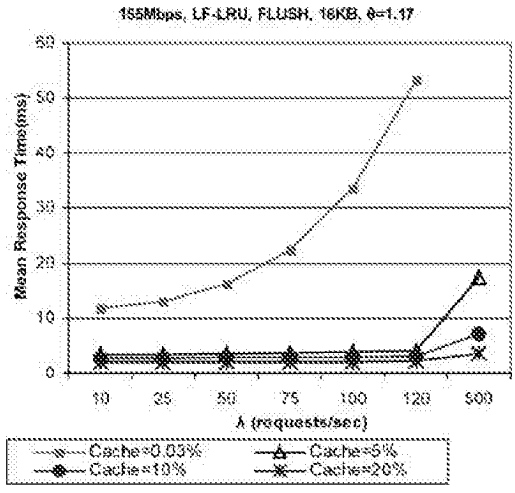


Figure 10.

(mostly referring to an outdoors wireless link), 155 Mbps and 620 Mbps (referring to ATM speeds OC-3 and OC-12, respectively). We present representative results for the case of 155 Mbps.

The system including the transmitter and the transmission channel was simulated as depicted in figure 1. The requests in the transmitter's queue are served in FIFO order. As in the previous section we present results only for the FLUSH algorithm, which performs better. We obtained results for many cache memory sizes in the range of 0.03% to 20% of the database size, but we only present some indicative results. We used the Zipf distribution for the simulation of the demands for various objects by the clients, with a variety of different values for the access skew coefficient θ, ranging from $\theta = 0$ to $\theta = 1.17$, and we only present the case of $\theta = 1.17$. The data item sizes that we have simulated are 16 KB and 200 KB.

Figure 10 shows the mean response time versus the request arrival rate λ, in the case of 16 KB data items, $\theta = 1.17$, for cache sizes equal to 0.03%, 5%, 10% and 20% of the database. We observe that the results in figure 7 are similar to those in figure 10 with an inappreciable increase of the mean response time, which is due to the additional transmission time. The results of the fairness index versus the request arrival rate λ are similar to those in figure 8, i.e., the fairness drops for larger cache sizes.

Figures 11 and 12 present the response time and fairness behavior versus the request arrival rate λ, in the case of 200 KB data item size, $\theta = 1.17$ and for cache sizes equal to 0.03%, 1%, 5% of the database, respectively. Comparing figure 11 with figure 9, we notice that the mean response times in the case of 0.03% cache size are similar, but as the cache size increases, the mean response time depicted in figure 11 increases compared to that of figure 9. For example, the mean response time in figure 9, in the case of cache size equal to 5% of the database, is around 8 ms for $\lambda = 120$ requests/s, while in figure 11 is around 79 ms. This is due to the fact that as the request arrival rate increases more requests are made for the most popular objects that are located in the cache memory and

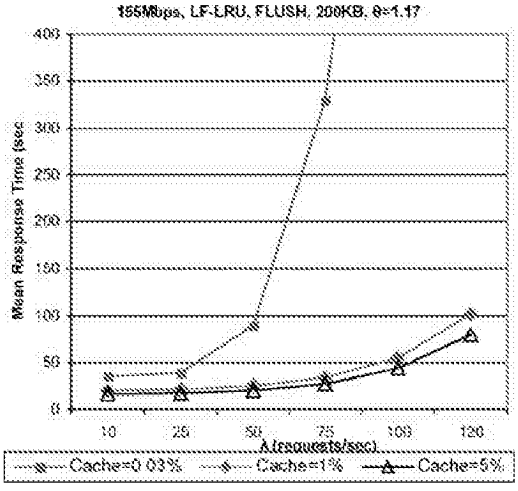


Figure 11.

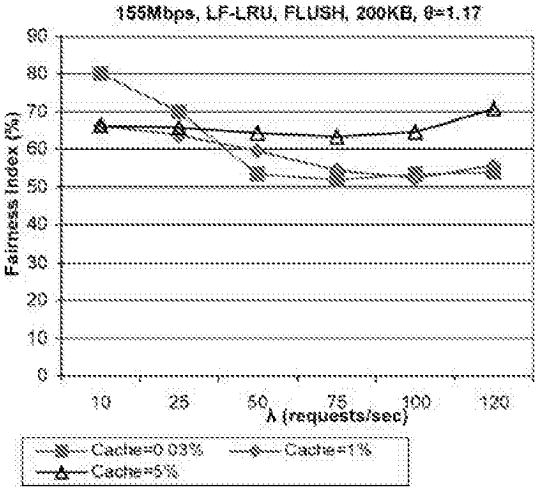


Figure 12.

are immediately directed to the transmitter queue inducing longer than before transmitter queues. As a result, the waiting time in the transmitter's queue increases the mean overall delay of the requests causing the difference in figures 11 and 9, as well as the change of the fairness index behavior shown in figure 12 (in contrast to the conclusion drawn from figure 8 that the fairness of the system decreases as the cache size increases).

Regarding the other results not shown here, the most noticeable are the following. From the results we obtained for the 2 Mbps transmission channel, we concluded that the channel speed is the system's bottleneck in that case. We examined FLUSH with LF-LRU with different cache sizes and we observed that the trend of decreasing mean response times as the cache size increases, observed in figures 10 and 12, is no longer valid, since the transmission channel is the bottleneck. When a 620 Mbps broadcast channel is considered, we observed that the mean response time behavior of the system is exactly the same with that presented in section 4.2. We conclude that for very high-speed channels the disk constitutes the systems' bottleneck.

6. Contributions and concluding remarks

Looking at related work for broadcast scheduling, one can find several interesting algorithms for deciding which data item to pick for broadcasting. However, all these algorithms make the (implicit or explicit) assumption that the chosen data item is immediately available to the transmitter for broadcasting. In a real system this obviously does not hold. This fact begs the question of how all the basic system components of a broadcast server's system infrastructure should interact in order to build high performance broadcast servers. With this paper we attempt to address this question. We put forward a comprehensive study from a system's viewpoint of the problem of broadcast scheduling. Our study is comprehensive in that it considers the interplay between the broadcast scheduling algorithms, the disk scheduling algorithms, and the cache management algorithms.

We first study the interplay between broadcast and disk scheduling algorithms, which will be the critical performance issue in applications and system configurations where the impact of caches will be secondary. We propose four novel scheduling algorithms, the ADoRe, FLUSH, OWeiST, and RxW/S classified under two categories: those that combine separate, "off-the-shelve" broadcast and disk scheduling algorithms and those that amalgamate the information available at the broadcast queue and at the disk queue, producing a single scheduling criterion. We study their performance in terms of mean response time and response time fairness, under different values of the problem parameters (system load, access distributions, object sizes, etc.).

Subsequently, we focus on the impact of the server's cache. We discuss a cache replacement scheme, the Least Frequently-Least Recently Used (LF-LRU) scheme, which has been found in our performance study to outperform other well-known cache replacement algorithms. In particular, it has been found to outperform LRU-2, while having similar administrative overheads. We have also studied the performance of the broadcast and disk scheduling algorithms coupled with this cache replacement algorithm. Finally, in an effort to study the end-to-end system performance, we also focused on the transmitter, studying its impact and presenting the end-to-end performance.

The major conclusions of this work are as follows.

In environments where the broadcast server depends heavily on the disk system, (i.e., it is disk-intensive as opposed to cache-intensive) the critical issue that affects the overall system performance is to design mechanisms which ensure that the broadcast server keeps the disk system highly utilized and allowing the disk scheduling algorithm to perform its optimizations. Our results show that a simple mechanism as embodied in our FLUSH algorithm outperforms significantly the other mechanisms. This is a strong indication that system developers need not spend time and effort developing mechanisms, which are based on more sophisticated and elaborate disk scheduling algorithms or on other sophisticated algorithms based on combining information at the broadcast and the disk queues.

Our performance study has shown that the proposed algorithms, namely, ADoRE, FLUSH, OWeiST, and RxW/S can achieve significantly better performance than that of the RxW algorithm (as it is straightforwardly implemented using the ADoRe (with $K = 1$)) in cache- or disk-intensive workloads and under a large variation of the values of the basic problem parameters (e.g., access skew distribution, system load, object sizes, etc.). Furthermore, and perhaps more importantly, our results show that a single algorithm, namely FLUSH, consistently outperforms, in terms of mean response times, all others under all the above workload and system parameters. At the same time, FLUSH produces fair schedules. This brings up another interesting issue raised from our comprehensive study, namely the following: one of the benefits of using RxW for broadcast scheduling, is the fact that it achieves fair schedules (through the W factor in its scheduling criterion). However, FLUSH also produces fair schedules due to the fairness achieved by the underlying disk scheduling algorithm (C-LOOK). Thus, when researchers are viewing in isolation and incomprehensively the various individual components, it is possible that considerable effort is spent to develop duplicate mechanisms to address the same problem (e.g., fairness), when a single mechanism would suffice.

A conclusion worthy of special notice is that in environments with or without cache memory (i.e., either with cache- or disk-intensive workloads) we have actually found that "efficient" broadcast scheduling algorithms found in the literature have a negligible impact. Specifically, if instead of employing the RxW algorithm at the broadcast queue we employed FCFS scheduling the difference in the overall performance of the broadcast system would barely be noticeable. Notice that, in the FLUSH algorithm, every time we have an arrival, it is passed to the disk system. Viewed differently, this implies that FLUSH essentially uses a FCFS scheduling at the broadcast queue. This conclusion is a strong indication that the main attention of most related research efforts needs to be refocused.

Finally, from our study incorporating the transmission queue we conclude that in wireless environments (i.e., with 2 Mbps bandwidth) the transmission server is the dominant performance bottleneck, reducing the significance of cache management and disk scheduling.

References

[1] S. Acharya, M. Franklin and S. Zdonik, Dissemination-based data delivery using broadcast disks, IEEE Personal Communications 2(6) (1995).

[2] S. Acharya, M. Franklin and S. Zdonik, Balancing push and pull for data broadcast, in: *Proc. ACM SIGMOD*, Tucson, AZ (May 1997).

[3] S. Acharya and S. Muthukrishnan, Scheduling on-demand broadcasts new metrics and algortihms, in: *Proc. ACM/IEEE Int. Conf. MobiCom'98*, Dallas (October 1998).

[4] D. Aksoy and M. Franklin, Scheduling for large-scale on-demand data broadcasting, in: *Proc. IEEE InfoCom*, San Francisco, CA (1998).

[5] R. Alonso, D. Barbara and H. Garcia-Molina, Data caching issues in an information retrieval system, TODS 15(3) (1990) 359–384.

[6] M.H. Ammar, Response time in a teletext system: an individual user's perspective, IEEE Transactions on Communications COM-35(11) (November 1987) 1159–1170.

[7] M.H. Ammar and J.W. Wong, On the optimality of cyclic transmission in teletext systems, IEEE Transactions on Communications, COM-35(1) (January 1987) 68–73.

[8] A. Bestavros and C. Cunha, Server initiated document dissemination for the WWW, IEEE Data Engineering Bulletin 19(3) (1996) 3–11.

[9] H. Dykeman, M.H. Ammar and J. Wong, Scheduling algorithms for videotext systems under broadcast delivery, in: *Proc. International Conference of Communications* (1996) pp. 1847–1851.

[10] M.J. Franklin, personal communication (January 2001).

[11] M.J. Franklin and S. Zdonik, A framework for scalable dissemination-based systems, in: *Proc. of the OOPSLA Int. Conf.* (1997) pp. 94–105.

[12] C. Georgiadis, P. Triantafillou and C. Faloutsos, Fundamentals of scheduling and performance of robotic tape libraries in video server environments, Multimedia Tools and Applications Journal (2001) (accepted, to appear).

[13] Intel Corporation, Intel Intercast Technology (1997) `http://www.intercast.com`

[14] R. Jain, D.-M. Chiu and W.R Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, DEC-TR-301 (September 1984).

[15] S. Jiang and N.H. Vaidya, Response time in data broadcast systems: Mean, variance and trade-off, in: *Proc. of Workshop on Satellite-Based Information (WOSBIS)*, Texas (October 1998).

[16] E.J. O'Neil, P.E. O'Neil and G. Weikum, An optimality proof of the LRU-K page replacement algorithm, Journal of the ACM 46 (January 1999).

[17] E. Pitoura and G. Samaras, *Data Management for Mobile Computing* (Kluwer Academic 1998).

[18] K. Stathatos, N. Rousopoulos and J.S. Baras, Adaptive data broadcast in hybric networks, in: *Proc. VLDB* (1997).

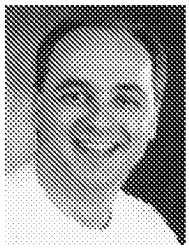
[19] C.-J. Su and L. Tassiulas, Broadcast scheduling for information distribution, in: *Proc. IEEE INFOCOM* CA (1997).

[20] C.J. Su and L. Tassiulas, Joint broadcast scheduling and user's cache management for efficient information delivery, in: *Int. Conf. MobiCom'98*, Dallas (October 1998).

[21] N.H. Vaidya and S. Hammed, Data broadcast in asymmetric wireless environment, in: *Proc. of Workshop on Satellite-Based Information (WOSBIS)*, New York (November 1996).

[22] J.W. Wong, Broadcast delivery, in: *Proc. of IEEE* (December 1988) pp. 1566–1577.

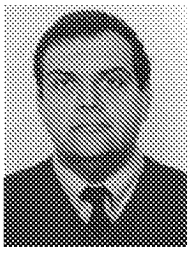
[23] B.L. Worthington, G.R. Ganger and Y. Patt, Scheduling algorithms for modern disk drivers, in: *Proc. of the 1994 ACM SIGMETRICS* (1994) pp. 241–251.



Peter Triantafillou received the Ph.D. degree from the Department of Computer Science at the University of Waterloo in 1991. Until August 1996 he was an Assistant Professor at the School of Computing Science at Simon Fraser University, Canada. From September 1994 till January 1996 he was on a leave of absence at the Department of Electronic and Computer Engineering at the Technical University of Crete, Greece. Since January 1996, he has been with the same department. Currently Peter Triantafillou is a Full Professor and Director of the Software Systems Engineering and Network Applications (SOFTNET) Laboratory. Prof. Triantafillou's research efforts currently focus on Internet content delivery and integration, peer-to-peer systems and publish/subscribe systems. In the recent past he worked on intelligent storage systems and on distributed servers, with emphasis on supporting multimedia applications. Earlier research activities had focused on multidatabases, distributed file systems, and highly-available distributed databases. Prof. Triantafillou has published extensively in all the

above areas and has been invited to serve in several Program Committees of related international conferences and as a reviewer in most relevant international journals and conferences.
E-mail: peter@softnet.tuc.gr
WWW: http://www.softnet.tuc.gr

Zaharoula Harpantidou received her five-year Diploma and M.Sc. degrees from the Department of Electronics & Computer Engineering of the Technical University of Crete in 1996 and 2000, respectively. During the period of September 1996–February 1997, she was a research assistant with CNUCE-CNR (Consiglio Nationale delle Ricerche) in Pisa, Italy. Her research interests are in the areas of computer network protocols, wireless communication networks, and real-time network applications.



Michael Paterakis received his Diploma degree from the National Technical University of Athens, Greece, his M.Sc. degree from the University of Connecticut, and his Ph.D. degree from the University of Virginia, in 1984, 1986, and 1988, respectively, all in electrical engineering. Since 1995, he is a faculty member in the Department of Electronic and Computer Engineering (ECE) at the Technical University of Crete, Greece, where he is currently a Professor and Director of the University's Telecommunication Systems Research Institute. During September 1999–August 2001, he served as the Chairman of the ECE Department. He was an Associate Professor in the Department of Computer and Information Sciences (CIS) at the University of Delaware, on the faculty of which he has been since September 1988. His research interests include computer communication networks with emphasis on protocol design, modeling and performance evaluation of broadband high speed networks, of multiple access wireless microcellular communication systems, and of packet radio networks; centralized and distributed multimedia information delivery systems; queueing and applied probability theory and their application to computer communication networks and information systems. He has published extensively (over 75 papers) in archival scientific journals, refereed conference proceedings and edited books, in the above-mentioned technical areas. He was invited to serve on the Technical Program Committees of the 1991 International Conference on Distributed Computing Systems, the 1992 and 1994 IEEE INFOCOM Conferences, the 1997 IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'97), the 6th IFIP Workshop on Performance Modeling and Evaluation of ATM Networks (IFIP ATM'98), and of the 2001 Very Large Databases Conference (VLDB'01). He has served as a reviewer for almost all of the major IEEE Transactions and other international archival scientific/technical journals and for most of the major international conferences in his research areas. Professor Paterakis is a Senior Member of the IEEE. He is also a member of the IEEE Technical Committee on Computer Communications, the Greek Chamber of Professional Engineers, and the Greek Association of Electrical and Electronic Engineers.