

A Novel Multicast Scheduling Scheme for Multimedia Servers with Variable Access Patterns

Minaxi Gupta, Mostafa Ammar
College of Computing
Georgia Institute of Technology
Atlanta, GA, U.S.A.
 minaxi, ammar@cc.gatech.edu

Abstract— Analysis of server logs from multimedia servers, a FTP server, and a web server suggest that irrespective of the content type and protocol used to retrieve it, the small percentage of files that account for the most load on the server exhibit a very dynamic popularity behavior. This observation has implications for content dissemination techniques like caching, server replication, content distribution networks, and multicast. This paper focuses on the impact of multimedia popularity on multicast scheduling. Guided by the file dynamics patterns observed in the server logs, we generate synthetic multimedia server logs with varying number of server accesses and evaluate existing multicast scheduling schemes in terms of client latency and renegeing of requests. Since existing scheduling schemes do not adapt gracefully to changing access conditions, we develop MWT, a new multicast scheduling scheme. MWT is fair to all multimedia files and reduces renegeing, leading to better utilization of server resources during heavy access conditions.

I. INTRODUCTION

Multicast efficiently accomplishes one-to-many and many-to-many delivery of data. Benefits of multicast are even more pronounced for delivery of multimedia content because of the large size of the files. Studies of media server workloads [1], [2] show that the accesses to the server are highly variable with time and the reason for this is the dynamic access patterns of a small number of popular files. To confirm our conjecture that other kinds of servers may also experience variable number of accesses due to dynamic popularity of a small percentage of their popular files, we collected logs from a FTP and a web server. Analysis of these logs helped us conclude that the accesses to these servers are also highly variable, indicating that the accesses to the popular files are very dynamic in nature.

The effectiveness of multicast depends on access patterns of the files, batch scheduling, channel allocation, network conditions, and the location of the receivers. Existing batch scheduling schemes like FCFS, MQL [3], MFQL [4], and RxW [5] are designed for static server access patterns. Our goal is to evaluate the performance of existing batch scheduling schemes under variable access patterns. To that end, and motivated by observations from the server logs, we generated a set of three synthetic logs with different server and file popularity dynamics to evaluate the performance of existing multicast scheduling schemes in terms of client latency and renegeing of client requests. While even during constant number of accesses to the server, some schemes perform better than others, all of them degrade in performance when the accesses to the server fluctuate.

During the periods when a small percentage of popular files exhibit dynamic profiles, all the schemes favor the dynamic files, giving them much lower client latency compared to the times when they have constant accesses. They do so at the cost of penalizing the less popular files who have not experienced a change in access patterns. Also, the renegeing during periods of higher accesses is very high.

To correct this situation, we developed a novel multicast scheduling scheme called MWT (*Minimum Waiting Time*) that provides lower client latencies to files that do not have dynamic profiles, while maintaining the response time for dynamic files. By trading of client latencies of dynamic files and the files with static profiles, MWT also reduces the renegeing of requests, leading to better server resource utilization.

The rest of this paper is organized as follows. Section II-A presents the results of our analysis of FTP and web server logs. Based on the observations of this section that the accesses to the server vary significantly with time, we generate three realistic synthetic multimedia server logs (section II-B). Section III presents the results of an evaluation of the performance of existing scheduling schemes for synthetic logs. These scheduling schemes do not perform well under variable server access patterns, motivating the need to develop a new scheduling scheme. Section IV presents the MWT batch scheduling scheme. The paper is concluded in section V.

II. FILE POPULARITY DYNAMICS

A. Server Log Analysis

The study [1] of educational media server workloads found that there are very few periods of stationary relative file access frequency for the media servers. This indicates that the accesses to media servers fluctuate with time and that media file popularity patterns are dynamic. Analysis of enterprise media logs [2] observed that the file popularity for the media server workloads can be approximated by a Zipf-like distribution at varying time scales and that in any given month, total accesses to the media servers are dominated by the new files introduced in that month. Furthermore, approximately 50% of the accesses to any file occur in the first week of their introduction. These findings corroborate the observations of [1] in terms of fluctuation in the number of server accesses and the dynamic nature of the file popularity. The duration of server logs used in these studies

	GT-FTP	GT-CoC
Duration	12/22/00-10/23/01	4/30/01-10/23/01
Total Sessions	14,344,037	37,372,732
# Unique Clients	84,144	1,103,692
# Unique Files	676,315	492,084
Avg. File Size (in MBytes)	.737	.046
Median File Size (in MBytes)	.017	.0015

TABLE I
PROFILE OF THE LOG COLLECTION SITES

varied from 1 month to 29 months, implying that these observations are not related to the duration of the logs. While the media servers used in both of these studies were not very busy servers (educational media servers in study [1] had 538 and 606 requests per day, and enterprise media servers used in study [2] had 23 and 753 requests per day), they raise an important point about file popularity dynamics. We ask the question if similar file popularity dynamics exist in workloads of other kinds of more busy servers, i.e., FTP and web servers; implying that variable server accesses are indeed widely prevalent.

To investigate this issue, we collected access logs from two unicast servers: *GT-FTP* (FTP logs collected from Georgia Tech’s Linux Mirror site); and *GT-CoC* (HTTP logs collected from Georgia Tech’s College of Computing web server). Table I lists some of the characteristics of each of the logs.

Table I shows that these logs vary from the server logs used in the multimedia studies [1], [2] in several ways. The duration of these logs, 10 months and 6 months falls in between that of the study in [1] (1 month and 3.5 months) and the study in [2] (21 months and 29 months). The number of requests that the FTP/web servers experience per day, 46, 876 and 211, 145 respectively, are much higher. The number of unique files accessed in FTP/web logs, 676, 315/492, 084 respectively is much higher than the 73, 1506 for study [1], and 2999, 412 for study [2].

We experimented with various daily and overall (for the entire duration of the logs) access thresholds for individual files to investigate the access patterns of the most popular files in the above logs. Choosing the daily access thresholds of 4, 000 for GT-FTP, and 15, 000 for GT-CoC; and overall access thresholds of 15, 000 for GT-FTP and 150, 000 for GT-CoC narrowed down the number of popular files to 13 (.002% of total) for GT-FTP and 23 (.005% of total) for GT-CoC. Figure 1 shows the overall access pattern at GT-FTP and the three types of access patterns that the most popular files exhibited. 6 out of the 13 most popular files peaked up in popularity for a very short time (see figure 1(b) for an example) and then were not accessed throughout the logs, 4 files had a modest profile (see figure 1(c) for an example), and 3 files peaked up quickly in popularity and then there was a slow decay (see figure 1(d) for an example). As the overall accesses graph indicates, the FTP server gained additional popularity into the 100th day into the logs, hence the most popular files belonged to the latter part of the logs.

Analysis of file access dynamics for GT-CoC led to different file dynamics. Figure 2 shows the overall access pattern at GT-WWW and the three types of access patterns that the most popular files exhibited. 20 out of the 23 most popular files exhibited a very modest profile (see figure 2(b) for an example), 2 files sharply peaked in popularity for a short time and then were not accessed again for the duration of the logs (see figure 2(c) for an example), and only 1 file showed a popularity pattern consisting of multiple popularity plateaus (see figure 2(d)).

While the mix of popular files with modest and other kinds of profiles may be site dependent, above analysis leads to the conclusion that dynamic file popularity is indeed a very pervasive phenomenon.

B. Synthetic Multimedia Logs

To test the impact of file popularity dynamics, we generated three synthetic logs, 12 hour each, for a multimedia server that serves 100 videos whose sizes range from 0 to 5MBytes. The idea was to have shorter duration logs with the same characteristics in terms of file access patterns as that of longer duration logs, only compressed in time. These logs are Zipf distributed with a parameter of 1.0. The exponentially distributed request arrival rate for these logs is 350 requests per minute. All the files in Ist log have approximately similar number of requests each hour. 95% of the files in the IInd log have constant access pattern throughout the logs but the 5% most popular files in the Zipf distribution (numbered 0 through 4) exhibit a dynamic profile, fluctuating the server accesses. In the IIIrd log, the top 10% of the files (numbered 0 through 9) exhibit a dynamic profile, forming two peaks in activity. In each peak of activity, 5% of the files participate. Also, the manner in which the 10% popular files exhibit dynamic profiles is different. The files in the first peak of activity rise gradually in popularity and either continue to be popular or gradually decrease in popularity. This peak in server accesses occurs from the 2nd hour until the 6th. On the other hand, the files in the second peak quickly rise in popularity at the 10th hour and the popularity subsides just as quickly. Figures 3 show the hourly server accesses and the hourly accesses for the individual popular files exhibiting dynamic profile. An important point to note about these logs is that all the three logs have the same total load on the multicast server in the 12 hour duration. This is important to study the impact of dynamic file profiles on the multicast batch scheduling. Multimedia clients can interact with the server by rewinding, pausing, or fast forwarding. For simplification purposes, the synthetic logs do not simulate information about client interactivity.

III. EVALUATION OF EXISTING MULTICAST SCHEDULING SCHEMES

While the impact of multimedia file popularity on other content distribution systems like caching, server replication, CDNs remains an area of future investigation, this paper only focuses on the impact of multimedia popularity dynamics on the scalability of multicast scheduling algorithms. Multicast can be accomplished at the routing layer [6] or at the application

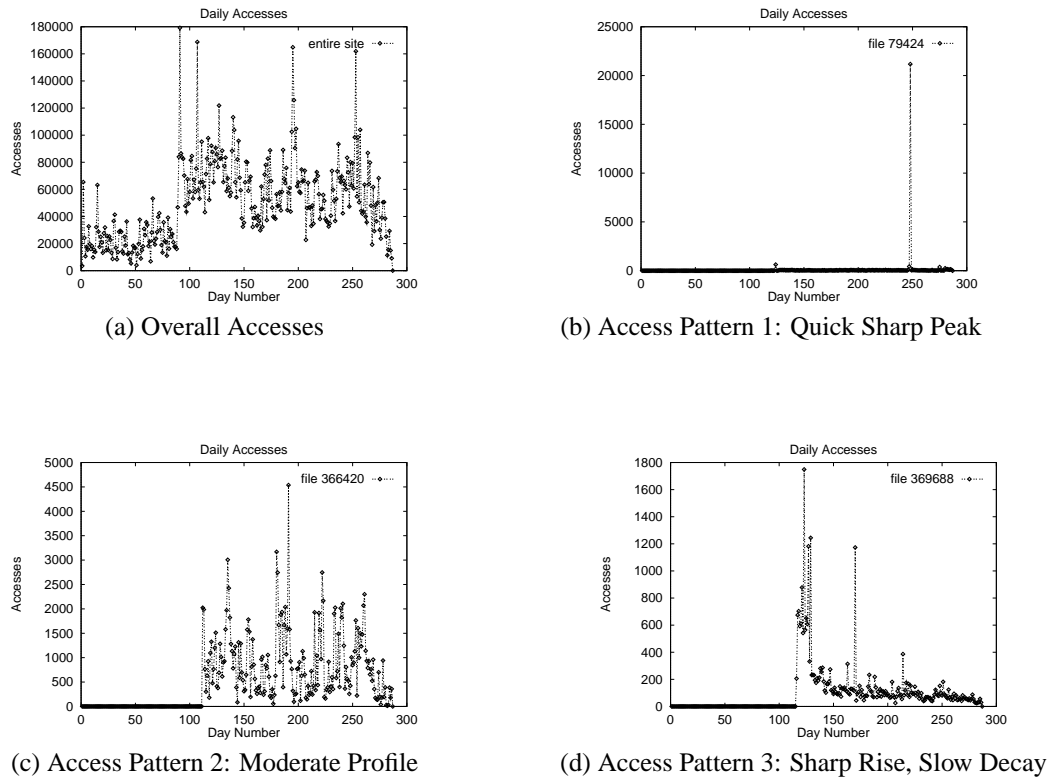


Fig. 1. FTP Server Access Patterns

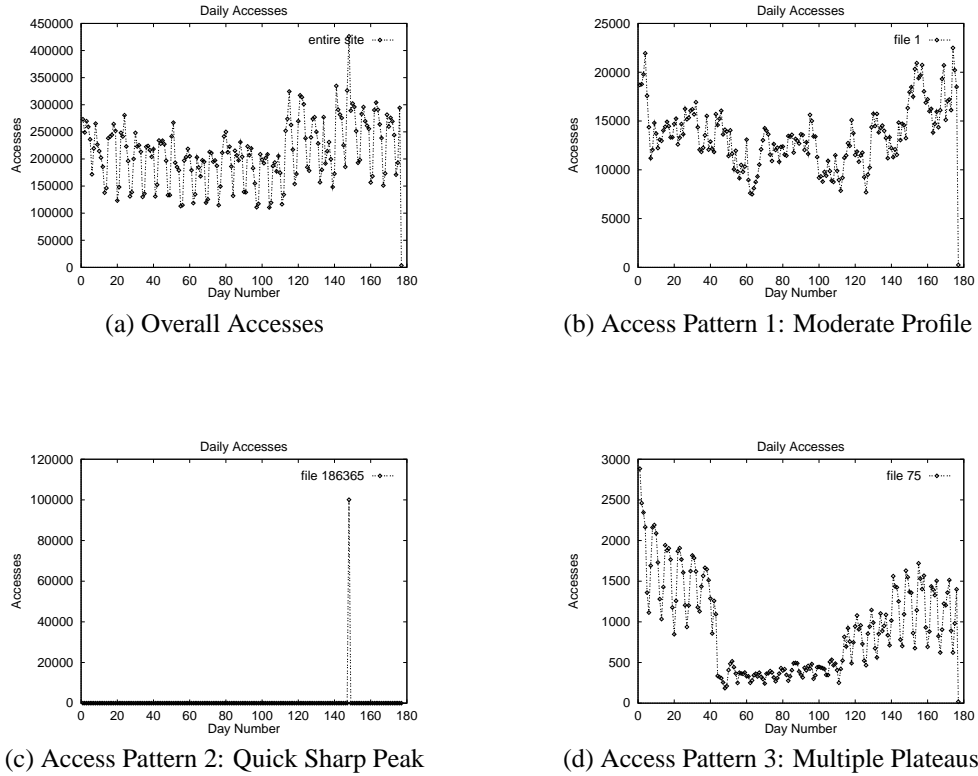


Fig. 2. WWW Server Access Patterns

layer [7]. The results of this paper are indifferent as to how multicasting of a multimedia file is accomplished.

A server accomplishes multicast of a file in essentially two

phases: the first phase is called *batch scheduling*, which involves selecting a batch of requests for a particular file; the second phase is called *channel allocation*, which involves deciding

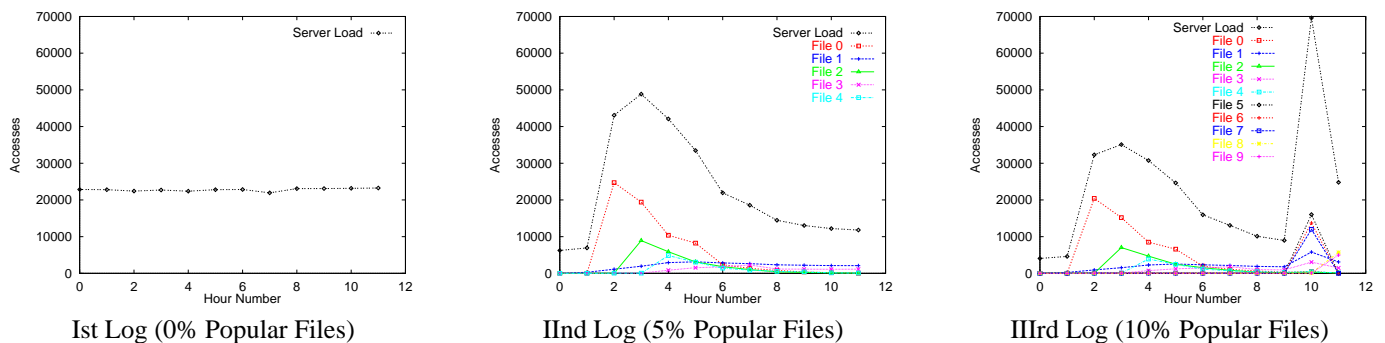


Fig. 3. Server and Popular File Workloads

how the channel should be allocated for the selected file. There are two major approaches for channel allocation for unicast and multicast: *persistent* channel allocation, and channel *merging*. In persistent channel allocation, once a channel is allocated to a file, it is used for multicasting the entire video to that batch of clients. Channel merging algorithms like the ones proposed in [8], [9], [10], [11] classify channels into persistent (regular) channels and patching channels. The basic idea is to transmit part of the video on the patching channel and possibly merge the clients on to the persistent channel that is in the process of transmitting the same video. This can be done only if the clients are able to receive data at a higher bit rates. We assume persistent channel allocation for simplicity and focus on the performance of the batch scheduling schemes.

In this section, we review various batch scheduling algorithms proposed in the literature. Dan et. al. [3] proposed FCFS (first come first served), MQL (maximum queue length), and FCFS- n as the batching policies. In FCFS, when a channel becomes available, the server multicasts the stream to the client that has been waiting the longest. All the clients that have requests already queued for the same video also get served. In MQL, the video that has the most number of requests queued for it is selected when a channel becomes available. FCFS- n is similar to FCFS, except that n channels are pre-allocated for the most powerful videos. FCFS is fair since it serves the client that has been waiting for the longest duration. MQL attempts to maximize the number of clients served by being biased for the more popular videos, but it does so at the expense of fairness to the clients that request the unpopular videos. FCFS- n was not observed to improve the performance of FCFS significantly and requires choosing an appropriate n .

MQL tends to be too aggressive in scheduling popular videos considering only the queue length, while FCFS completely ignores the queue length and focuses only on arrival time to reduce defections. Aggarwal et. al. [4] proposed MFQL (maximum factored queue length), a batch scheduling policy with a notion of factored queue length. The factored queue length is obtained by weighting each video queue length with the square root of its popularity, a factor which is biased against the popular videos. The authors show that MFQL yields excellent empirical results in terms of standard performance measures such

as average latency, reneging rates, and fairness.

With on-demand data broadcast in mind, Aksoy et. al. proposed RxW [5], a parameterized broadcast scheduling algorithm that makes scheduling decisions based on the current request queue and adapts well with client population and access pattern changes. At each scheduling decision, the RxW algorithm chooses to broadcast the page with the maximal $R * W$ value where R is the number of outstanding requests for a page and W is the time the the oldest outstanding request for that page has been waiting.

Our goal is to evaluate the performance of FCFS, MQL, MFQL, and RxW batch scheduling policies under dynamic multimedia file popularity using the synthetic logs we generated. The metrics being considered are client latency, and client reneging. Client latency can be defined in a variety of ways. One can define it to be the time it takes for the client to start receiving the video from the time of its request. Varying network conditions would make it hard to judge the impact of file popularity dynamics on each of the scheduling policies. Moreover, it is hard to simulate real world network conditions. Hence, we decided not to take into account the network conditions while finding the client latency. As a result, we define *client latency* to be the time a client request spends waiting for the server. It is the time from the point the server receives a request to the point it starts serving this request. After the request has been waiting in the queue for some amount of time, the client may decide to go away without waiting for the delivery of the file. The amount of time after which the client does so is called the *reneging time*. For our event driven simulations, we considered a multicast multimedia server with 100 transmission channels and a transmission rate of $100kbps$. For each of the logs, we studied the average hourly client latency for all the video files, just the popular files that are causing the increase in server accesses (dynamic files), and the rest of the files.

Recall that the MFQL batch scheduling algorithm requires the square root of file popularity in addition to the queue length. Since the file popularity of some of the files changes dynamically, we decided to experiment with various values of smoothing parameter α in the following standard moving weighted average equation:

$$f_{new} = \alpha * f_{old} + (1 - \alpha) * f_{sample}$$

f_{old} in the above equation represents the previously estimated popularity for a particular video, f_{sample} represents the number of requests for this video in this estimation interval, and f_{new} is the new popularity value for this video. We experimented with $\alpha = 0.1, 0.5, 0.8$ for two different estimation intervals. The first interval was 50 minutes, meaning that we estimated the popularity of each video every 50 minutes. We chose this interval to ensure the misalignment of the popularity estimation interval with increase in file popularity in the synthetic logs. The second interval we experimented with was 8 minutes. It turns out that $\alpha = 0.5$ and popularity estimation interval of 50 minutes produced the best results and ways of popularity estimation do not differ significantly from each other. We also experimented with *static* MFQL (static-MFQL) where the Zipf probabilities were used as static popularity values for the simulations (we refer to this case as the *static-MFQL*). In general such information is not likely to be available ahead of time.

Figure 4 shows the average hourly client latency and renegeing for MQL, MFQL (50 minute popularity estimation interval and $\alpha = 0.5$), *static-MFQL*, and RxW for the case when the load on the server is Zipf distributed but constant (basically, using Ist synthetic log). We eliminate presenting FCFS results because FCFS performs poorly on both counts. We tested the schemes for three renegeing times, 1 minute, 2 minutes, and 5 minutes.

As expected, MFQL (both static-MFQL and when $\alpha = 0.5$ and popularity estimation interval 50 minutes) outperform all other scheduling schemes in terms of providing the clients with lower access latency and in terms of the number of clients renegeing every hour. We then tested the performance of all the above schemes for IInd and IIIrd logs, where the most popular 5% and 10% of files have dynamic access profiles. Figure 5 shows the average hourly client latency and renegeing for all the files, just the top 5% dynamic files, and the rest of the 95% of the files for the IInd synthetic log when the renegeing happens after 2 minutes of waiting. The results for log III and other renegeing times were similar.

As figure 5 shows, MQL performs the worst among all the schemes in terms of graceful degradation when the accesses to the server is increased because of temporary increase in the number of accesses to the most popular 5% of the files. Even though the average client latency for all the files combined from hours 2 through 6 is lower than the case of Ist log (when the access patterns for all files were constant), it comes at the cost of penalizing the rest of the 95% of the files, whose access patterns are constant throughout the logs. The top 5% of the dynamic files get much better performance for all the schemes. Renegeing during the peak access period also goes up considerably and even the best performing schemes (RxW and static-MFQL) experience about 50% more renegeing for above graphs compared to the constant access case. Once again, most of the renegeing happens for the 95% of the files and the top 5% popular files fair much better.

IV. MINIMUM WAITING TIME SCHEDULING SCHEME

We propose a new multicast batch scheduling scheme called MWT (*Minimum Waiting Time*) for the multimedia multicast servers that experience variable number of accesses. MWT is designed with the following objective.

Objective: To provide similar average client latency to all files, irrespective of their popularity under all access conditions, while keeping the renegeing to a minimum.

To illustrate the objective, we observe that during the period the server is experiencing higher number of accesses

- a file f_i experiencing a surge in demand should get similar client latency to a file f_j whose accesses have not changed over time.
- the client latency of any file f_i should be the same as it would have been when the server accesses were constant.
- the system renegeing should be as close as possible to that experienced when the access pattern is constant.

Notice that MWT attempts to be as fair in terms of which files experience renegeing, as is important to reduce the number of requests that experience renegeing. It tries to reduce the renegeing of requests for files whose demands have not changed, at the same time not increasing the renegeing of files contributing to the heavy access conditions much. It accomplishes this goal by leveraging the fact that files contributing to heavy access conditions experience lower client latencies during peak access conditions under all other scheduling schemes compared to the case when their demand patterns are constant.

Since MFQL exhibits the best performance under constant access conditions and is known to be fair, during the periods when the server experiences constant accesses for all its files according to their Zipf popularity, MWT behaves just like MFQL. When accesses for some files increase, causing the accesses to the server to increase as well, MWT gives priority to the files not experiencing variation in their access pattern under certain conditions. If the first request for any of the dynamic files has been waiting for less time than a factor of the renegeing time, i.e., it forces the dynamic files to wait for a minimum amount of time and schedules the files whose access pattern has not changed instead. We call the factor that decides the minimum waiting time for the dynamic files to be the *min wait factor*. Figure 6 shows how MWT chooses one of the two files f_i and f_j to schedule an available channel under heavy access conditions:

For renegeing times of 1 minute, 2 minutes, and 5 minutes, we experimented with various *min wait factors* (4, 3, 2, 1.5, 1) for synthetic logs II and III. The *min wait factor* that provides the best results depends on the renegeing time of the file. We assume all the files have same renegeing time in one run. Assuming that the server has knowledge about the renegeing time, it can use the appropriate wait factor to increase fairness among the files in terms of client latency and to increase the system utilization by reducing the renegeing.

While we are not able to show graphs for all the results for both logs, figures 7 and 8 show the results for the case when the renegeing time is 2 minutes for synthetic logs IInd and IIIrd respectively. We compare the performance of MWT to static-MFQL.

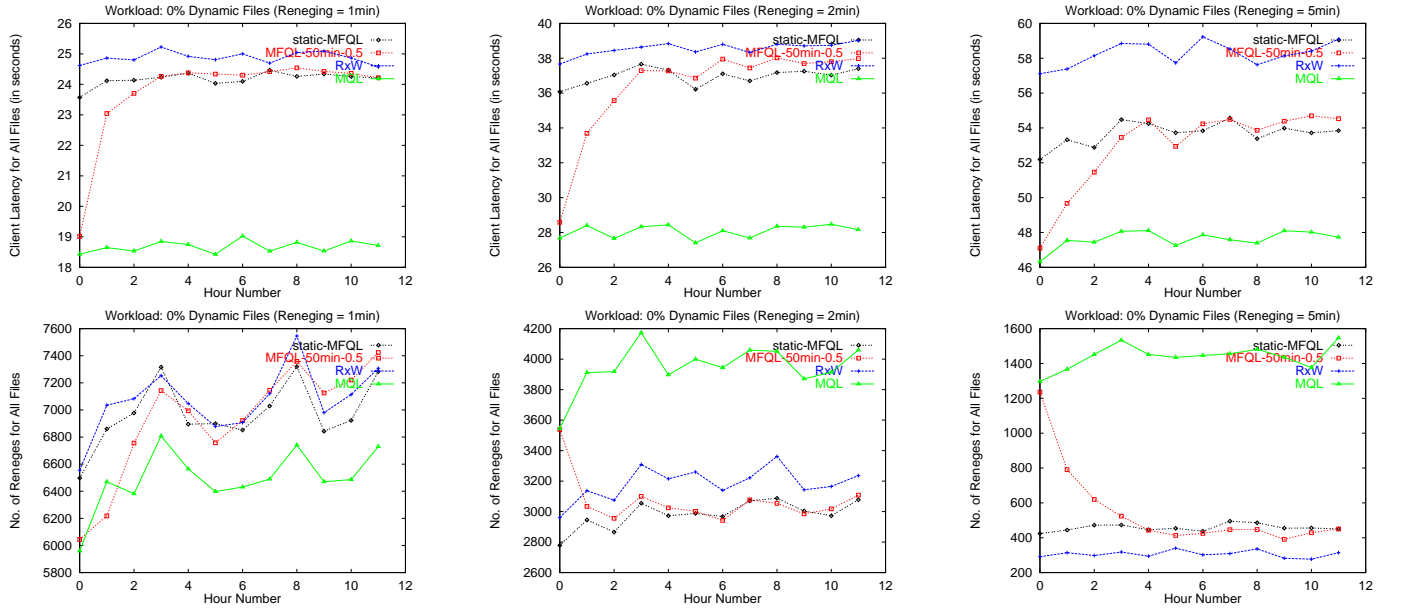


Fig. 4. Average Client Latency and Reneging for All Files Combined for Synthetic Log I

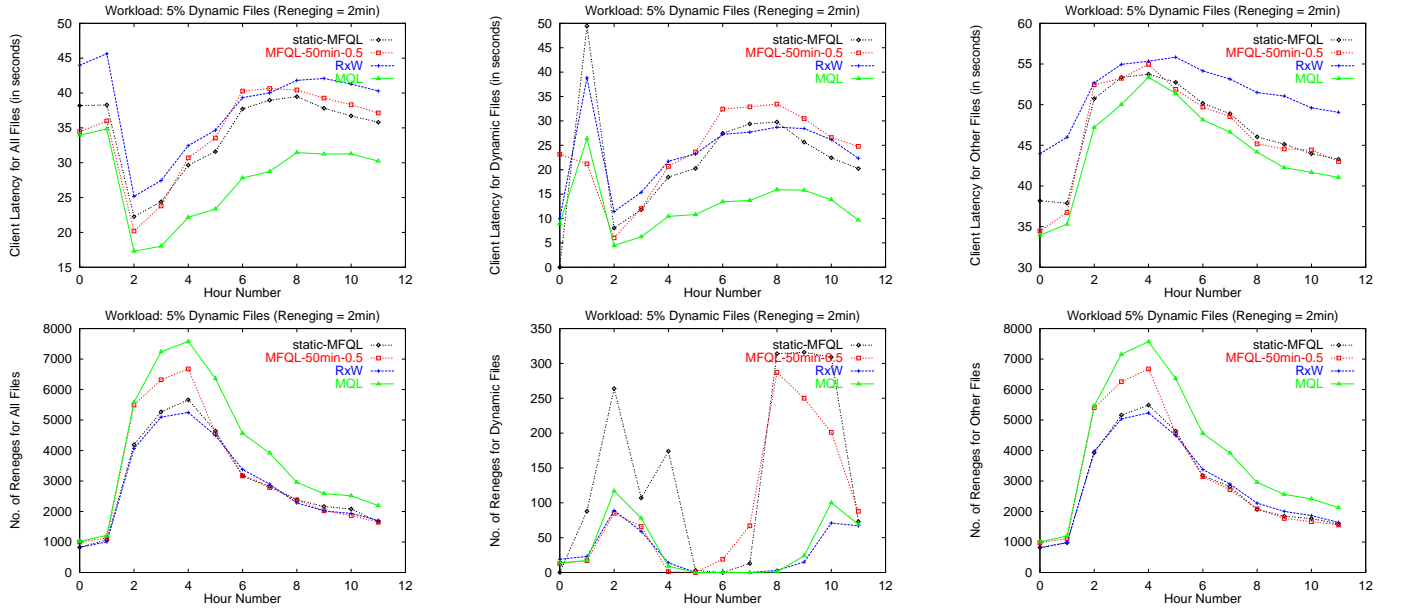


Fig. 5. Average Client Latency and Reneging for Synthetic Log II

```

if (f_i == dynamic_file and f_j != dynamic_file) {
  if (f_i->first_request_wait_time < reneging_time/min_wait_factor) {
    schedule f_j
  } else {
    do as MFQL would
  }
} else {
  do as MFQL would
}

```

Fig. 6. MWT Scheduling Algorithm for Heavy Access Conditions

As figure 7 shows, the average client latency for all files for a *min wait factor* of 2 comes close to the average latency for

the constant access case of log I, as shown in 4. Although the average client latency for the 95% of the files is better for *min*

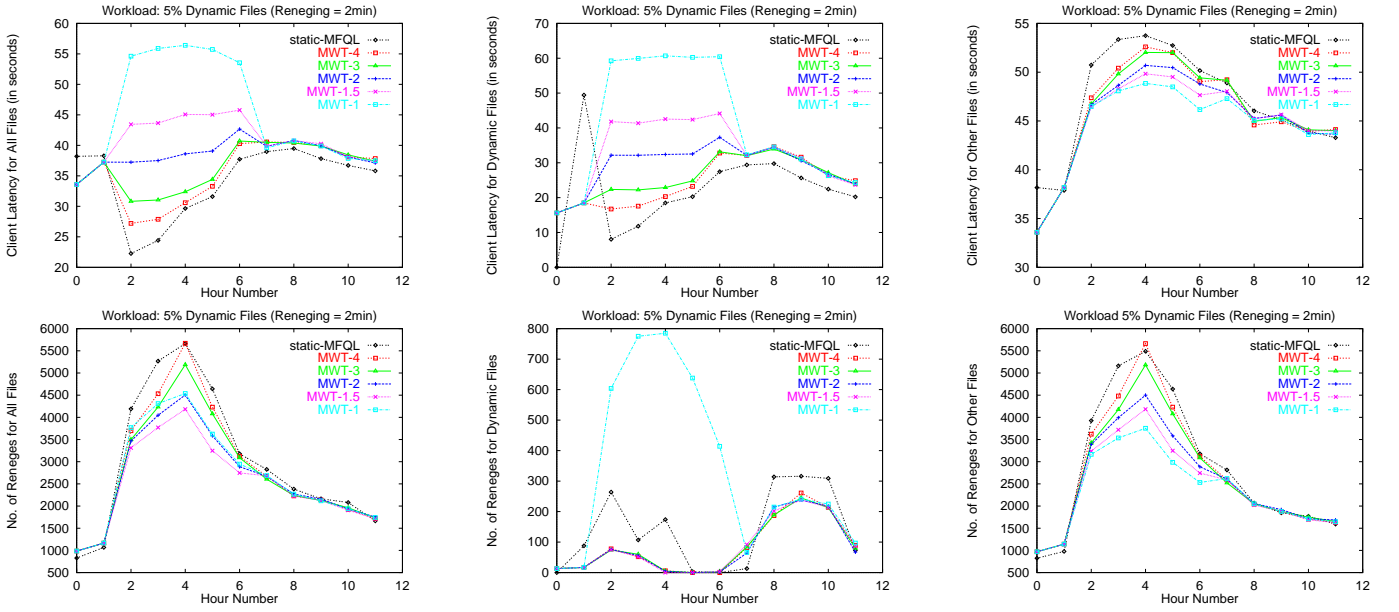


Fig. 7. MWT for Synthetic Log II (Renewing=2min)

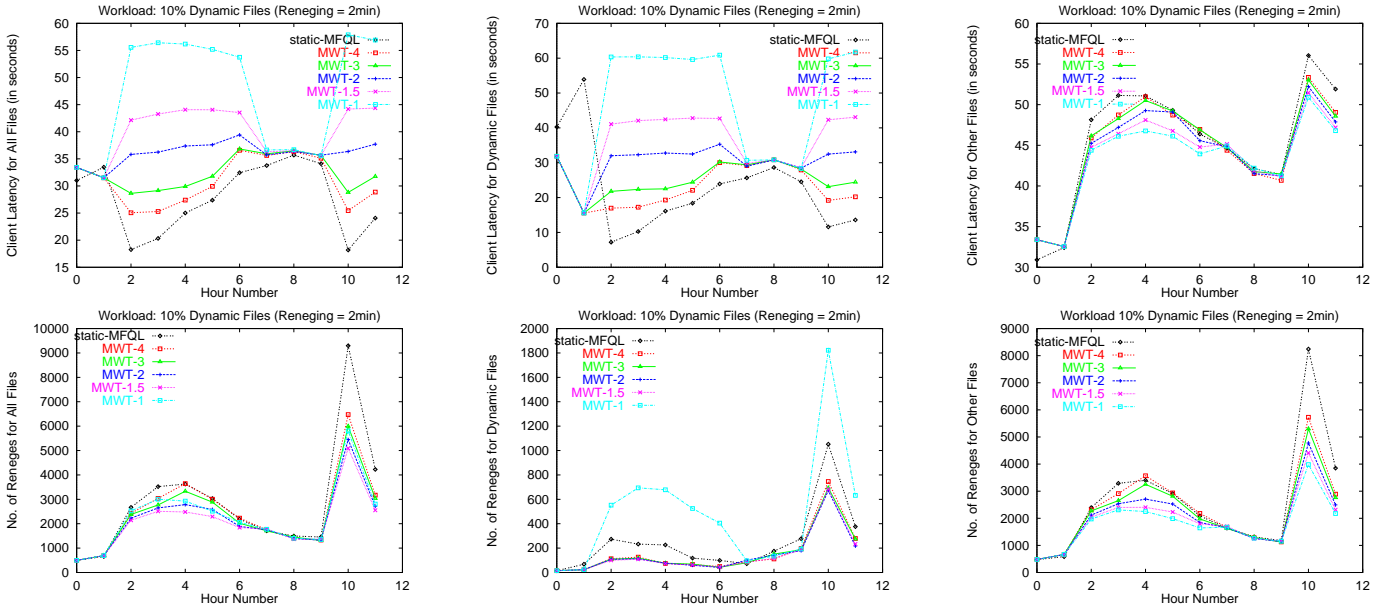


Fig. 8. MWT for Synthetic Log III (Renewing=2min)

wait factor of 1.5 and 1, it is not desirable, because the benefit comes in the form of significantly degrading the average client latency for the dynamic 5% files, impacting the overall latency as well. Overall reneging as well as the reneging for 5% and 95% of the files are significantly better than static-MFQL for both *min wait factor* of 2 and 1.5. However, when the *min wait factor* decreases to 1 (causing the first request of 5% files to wait at least the same amount as the renewing time), the overall reneging increases, because of the reneging in the popular files requests.

As figure 8 shows, the results for log III are similar. This log has two peaks, one between the hours of 2 and 6, and another sharp one at the 10th hour. The first peak is contributed by

the top 5% most popular files and the second one by the next 5% most popular files. The best *min wait factor* for log III is 2. As the graphs show, the benefits of using MWT are more pronounced in terms of decrease in the reneging of requests for the second peak.

The results of experimenting with various *min wait factors* for other renewing times (1 minute, and 5 minutes) were similar. The bigger the renewing time, the smaller the *min wait factor* required (implying larger the average delay in scheduling the dynamic files) for MWT.

V. CONCLUSION

This paper makes two main contributions. Using logs from FTP and web logs, it recognizes the fact that the accesses to

servers vary significantly over time. This is primarily due a small percentage of popular files that exhibit dynamic profiles.

Using synthetic workloads, we conclude that existing multicast schemes do not perform well under variable access patterns. To alleviate that situation, we presented a novel multicast scheduling scheme for multicast multimedia servers with variable access patterns. MWT keeps the average client latency for all files during periods of heavy accesses similar to that during constant server access conditions. Compared to MFQL, it trades the lower than average latency for the dynamic files to provide better response time to the files whose request pattern remain unchanged over time. It also reduces the renegeing of requests for all files compared to MFQL.

REFERENCES

- [1] J. Almeida, J. Krueger, D. Eager, and M. Vernon. *Analysis of Educational Media Server Workloads*. NOSSDAV'01. Jun 2001.
- [2] L. Cherkasova and M. Gupta. *Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads*. NOSSDAV'02. May 2002.
- [3] A. Dan, D. Sitaram, and P. Shahabuddin. *Scheduling Policies for an On-Demand Video Server with Batching*. ACM Multimedia. October 1994.
- [4] C. Aggarwal, J. Wolf, and P. Yu. *The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems*. IEEE Transactions On Computers 50(2), pages 97-110. 2001.
- [5] D. Aksoy and M. Franklin. *RxW: A Scheduling Approach for Large-Scale On-Demand Data Broadcast*. IEEE Transactions On Networking, Vol. 7, Number 6, pages 846-860. December 1999.
- [6] K. Almeroth. *The Evolution of Multicast: From the Mbone to Inter-Domain Multicast to Internet2 Deployment*. IEEE Network Special Issue on Multicasting. January/February 2000.
- [7] Y. -H. Chu, S. G. Rao, and H. Zhang. *A Case for End System Multicast*. ACM SIGMETRICS. June 2000.
- [8] K. A. Hua, Y. Cai, and S. Sheu. *Patching: A Multicast Technique for True Video On-Demand Services*. ACM Multimedia. September 1998.
- [9] K. A. Hua and S. Sheu. *Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video On-Demand Systems*. ACM SIGCOMM. September 1997.
- [10] C. Aggarwal, J. Wolf, and P. Yu. *A Permutation Based Pyramid Broadcasting Scheme for Video On-Demand Systems*. IEEE International Conference on Multimedia Computing and Systems (ICMCS). June 1996.
- [11] D. Eager, M. Vernon, and J. Zahorjan. *Optimal and Efficient Merging Schedules for Video On-Demand Servers*. ACM Multimedia. November 1999.