PREDICTIVE COST-BASED SCHEDULING FOR SCALABLE MEDIA STREAMING

Mohammad A. Alsmirat and Nabil J. Sarhan

Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI 48202, USA
Email: {msmirat,nabil}@wayne.edu

ABSTRACT

We propose a scheduling policy, called *Predictive Cost-Based Scheduling* (PCS) for scalable video streaming. PCS schedules the waiting requests based on their required delivery costs, predicts future system state, and uses the prediction results to possibly alter the scheduling decisions. We also present two alternative implementations of PCS and analyze its waiting-time predictability. The simulation results show that PCS can achieve significant performance improvements and can provide users with highly accurate expected waiting times.

Keywords: Scheduling, stream merging, video streaming, waiting-time prediction.

1. INTRODUCTION

The interest in media streaming has grown dramatically. This paper considers video streaming of pre-recorded content. Unfortunately, the number of video streams that can be supported concurrently is highly constrained by the stringent requirements of multimedia data. Stream merging techniques [1, 2, 3] address this challenge by aggregating clients into larger groups that share the same multicast streams. Periodic broadcasting [4] (and references within) also address this challenge but can be used for only popular videos and require the requests to wait until the next broadcast time of the first corresponding segment. This paper uses the stream merging approach. A cost-based scheduling policy, called *Minimum Cost First* (MCF) [5], has recently been proposed for use with stream merging. It captures the significant variation in stream lengths by selecting the requests requiring the least cost. MCF performs significantly better than all other scheduling policies, such as *First Come First Serve* (FCFS) [6], *Maximum Queue Length* (MQL) [6], and *Maximum Factored Queue Length* (MFQL) [7].

We propose a scheduling policy, called *Predictive Cost-Based Scheduling* (PCS). Like MCF, PCS is cost-based, but it predicts future system state and uses the prediction results to potentially alter the scheduling decisions. It delays servicing requests at the current scheduling time (even when resource are available) if it is expected that shorter streams will be required at the next scheduling time. We present two alternative implementations of PCS. Moreover, we analyze its waiting-time predictability. Motivated by the rapidly growing interest in human-centered multimedia, the ability to inform users about how long they need to wait for service has become of great importance [8]. Providing users with waiting-time feedback enhances their perceived quality-of-service (QoS) and encourages them to wait, thereby increasing server throughput.

The simulation results show that the proposed scheduling policy leads to significant performance improvements and can provide users with highly accurate waiting times for service.

The rest of the paper is organized as follows. Section 2 discusses the background information. Section 3 presents the proposed policy and its implementations. Subsequently, Section 4 discusses the performance evaluation methodology and presents and analyzes the main results.

2. BACKGROUND INFORMATION

Stream merging techniques include *Patching* [1] and *Earliest Reachable Merge Target* (ERMT) [2]. Patching allows a new request to join immediately the latest video multicast stream and to receive the missing portion as a unicast stream. Full streams are retransmitted when the required patch length for a new request exceeds a pre-specified value. ERMT is a near optimal hierarchical stream merging technique. A new client or a newly merged group of clients snoops on the closest stream that it can merge with if no later arrivals preemptively catch them [2].

A scheduling policy determines the next video to be serviced whenever a *server channel* becomes available. All waiting requests for the selected video can be serviced using only one channel. A channel is a set of resources needed to deliver a multimedia stream. The number of channels is called *server capacity*. *Minimum Cost First* (MCF) [5] is the best existing scheduling policy. It selects the video requiring the least cost.

This work is supported in part by NSF grant CNS-0626861.

The length of the required stream (in time) is directly proportional to the cost of servicing that stream because the server allocates a channel for the entire time the stream is active. *MCF-P* (P for "Per") is the best implementation of MCF. It selects the video with the least cost per request. We consider here the preferred variant of MCF-P that computes the cost of regular streams as if they were patches.

3. PREDICTIVE COST-BASED SCHEDULING

We propose a scheduling policy, called *Predictive Cost-Based Scheduling* (PCS). PCS is based on MCF, but it predicts future system state and uses this prediction to possibly alter the scheduling decisions. The basic idea can be explained as follows. When a channel becomes available, PCS determines using the MCF-P objective function the video V_{Now} which is to be serviced tentatively at the current scheduling time (T_{Now}) and its associated delivery cost. To avoid unfairness against videos with high data rates, we use the required stream length for the cost [5]. (Please note the distinction between video lengths and required stream lengths. Due to stream merging, even the requests for the same video may require different stream lengths.) Before actually servicing that video, PCS predicts the system state at the next scheduling time (T_{Next}) and estimates the delivery cost at that time assuming that video V_{Now} is not serviced at time T_{Now}. PCS does not service any request at time T_{Now} and thus postpone the service of video V_{Now} if the delivery cost at time T_{Next} is lower than that at time T_{Now}. Otherwise, video V_{Now} is serviced immediately.

To reduce possible server underutilization, PCS delays the service of streams only if the number of available server channels ($freeChannels$) is smaller than a certain threshold ($freeChannelThresh$). Figure 1 shows a proposed algorithm to dynamically find the best value of $freeChannelThresh$. The algorithm changes the value of the threshold and observes its impact on customer defection (i.e., turn-away) probability over a certain time interval. The defection probability is the probability that customers leave the system without being serviced because of waiting times exceeding their tolerance. This probability directly translates to server throughput. The value of the threshold is then updated based on the trend in defection probability (increase or decrease) and the last action (increase or decrease) performed on the threshold. The algorithm is to be executed periodically but not frequently to ensure stable system behavior.

We present two alternative implementations of PCS: *PCS-V* and *PCS-L*. These two implementations differ in how to compute the delivery cost or required stream length at the next scheduling time. PCS-V predicts the video to be serviced at the next scheduling time and simply uses its required stream length. The video prediction is done by utilizing detailed information about the current state of the server in a manner similar to that of the waiting-time prediction approach in [8].

```
currDefectionRate =
            defectedCustomers/servedCustomers;
if (currDefectionRate < lastDefectionRate) {
   if (last action was decrement and freeChannelThresh > 2)
      freeChannelThresh − −;
   else if (last action was increment)
      freeChannelThresh + +;
} else if (currDefectionRate > lastDefectionRate){
   if (last action was increment and freeChannelThresh > 2)
      freeChannelThresh − −;
   else if (last action was decrement)
      freeChannelThresh + +;
}
lastDefectionRate = currDefectionRate;
```

Fig. 1: Simplified Algorithm for Dynamically Computing *freeChannelThresh*

This information includes the number of waiting requests for each video, the completion times of running streams, and statistics such as the average request arrival rate for each video (which is to be updated periodically). Figure 2 shows a simplified algorithm for PCS-V.

```
if (freeChannels ≥ freeChannelThresh)
   Service the requests for V_Now;
else {
   V_Now =
         find the video that will tentatively be serviced at T_Now;
   currStreamLen =
         find required stream length to service V_Now at T_Now;
   V_Next =
         find the video that is expected to be serviced at T_Next;
   nextStreamLen =
         find required stream length to service V_Next at T_Next;
   if (currSreamLen ≤ nextStreamLen)
      Service the requests for V_Now;
}
```

Fig. 2: Simplified Algorithm for PCS-V

In contrast with PCS-V, PCS-L computes the expected required stream length at the next scheduling time based on the lengths of all possible video streams that may be required and their probabilities. A simplified algorithm for PCS-L is shown in Figure 3. The probability that a video is selected is equal to the probability that it has at least one waiting request at time T_{Next} times the probability that all video streams with lower cost (i.e. shorter required streams) are not selected. The probability that video v has at least one arrival during duration $T_{Next} - T_{Now}$ can be found as one minus the probability of exactly zero arrivals: $1 - e^{-\lambda_v \times (T_{Next} - T_{Now})}$, where λ_i is the request arrival rate for video v and assuming a Poisson arrival process. If the video has already one waiting request, then this probability is 1. Sorting the videos according to the scheduling objective function is required to determine the probability

that all videos with lower cost (or higher objective) are not selected.

```
V_Now = find the video that will tentatively be serviced at T_Now;
if (freeChannels ≥ freeChannelThresh)
    Service the requests for V_Now;
else {
    currStreamLen =
            find required stream length to service V_Now at T_Now;
    Calculate objective function for each video at T_Next;
    Sort videos from best to worst according to objective function;
    expectedStreamLen = 0; // initialization
    // loop to find expected stream length at T_Next
    for (v = 0; v < N_v ; v++){ // for each video
        nextStreamLen =
                find required stream length to service v at T_Next;
        Prob(video v is selected) =
                Prob(no other video with better objective is selected)
                 * Prob(video v has at least one arrival);
        expectedStreamLen +=
                Prob(video v is selected ) * nextStreamLen;
    }
    if (currSreamLen ≤ expectedStreamLen)
        Service the requests for V_Now;
}
```

Fig. 3: Simplified Algorithm for PCS-L

4. EVALUATION METHODOLOGY AND RESULTS

We study the effectiveness of the proposed policy through simulation. The used simulator extends the media streaming simulator in [9] by implementing waiting-time prediction and new algorithms and enhancements. Table 1 summarizes the workload characteristics used. We characterize the waiting tolerance of customers by three models. In *Model A*, the waiting tolerance follows an exponential distribution with mean μ_{tol}. In *Model B*, users with expected waiting times less than μ_{tol} will wait and the others will have the same waiting tolerance as Model A. We use *Model C* to capture situations in which users either wait or defect immediately depending on the expected waiting times. The user waits if the expected waiting time is less than μ_{tol} and defects immediately if the waiting time is greater than $2\mu_{tol}$. Otherwise, the defection probability decreases linearly from 1 to 0 for the expected waiting times between μ_{tol} and $2\mu_{tol}$.

The analyzed performance metrics include customer defection probability, average waiting time, and unfairness against unpopular videos. The waiting-time predictability is analyzed by two metrics: waiting-time prediction accuracy and the percentage of clients receiving expected waiting times. The waiting-time prediction accuracy is determined by the average deviation between the expected and actual waiting times. For waiting-time prediction, we use the algorithm in [8]. Note that this algorithm may not provide an expected waiting time to each client because the prediction may not always be performed accurately.

Table 1: Summary of Workload Characteristics

Parameter	**Model/Value(s)**
Request Arrival	Poisson Process
Request Arrival Rate	40 Req./min
Server Capacity	300 to 750 channels
Video Access	Zipf-Like ($\theta = 0.271$)
Number of Videos	120
Video Length	120 min
Waiting Tolerance Model	A, B, C
Waiting Tolerance Mean	$\mu_{tol} = 30$ sec

For space limitation, we discuss next only the main results. Figures 4 and 5 demonstrate the effectiveness of the two implementations of PCS when applied with ERMT and Patching, respectively, in terms of the defection percentage, average waiting time, and unfairness. The figures shows that PCS outperforms MCF-P in terms of both the two most important performance metrics (defection percentage and waiting time), whereas MCF-P is fairer towards unpopular videos. The two implementations of PCS perform nearly the same and thus PCS-V is preferred because of its simplicity.

Figures 6 and 7 compare the predictability of PCS-V with MCF-P in term of the average deviation and the percentage of clients receiving expected waiting times under waiting tolerance models B and C, respectively. The results are shown when ERMT is applied. Patching exhibits similar results and thus not shown. As we can see, PCS-V reduces the average deviation by approximately 60% and 80% for models B and C, respectively. This significant enhancement in prediction accuracy happens at the expense of decreasing much less significantly the number of clients receiving expected times.

5. CONCLUSIONS

The results show that the proposed PCS scheduling policy outperforms the best existing policy in terms of customer defection probability and average waiting time. The waiting times can also be predicted more accurately with PCS.

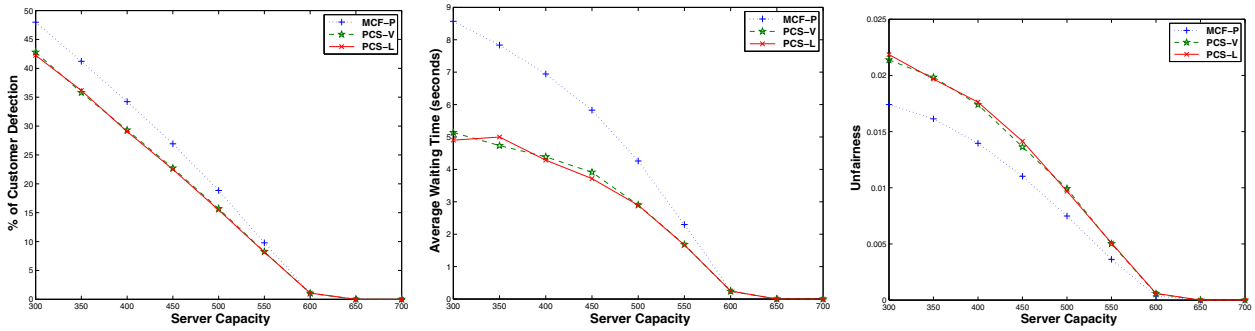
6. REFERENCES

[1] Y. Cai and Kien A. Hua, "Sharing multicast videos using patching streams," *Multimedia Tools and Applications journal*, vol. 21, no. 2, pp. 125–146, Nov. 2003.

[2] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for Video-on-Demand servers," in *Proc. of ACM Multimedia*, Oct. 1999, pp. 199–202.

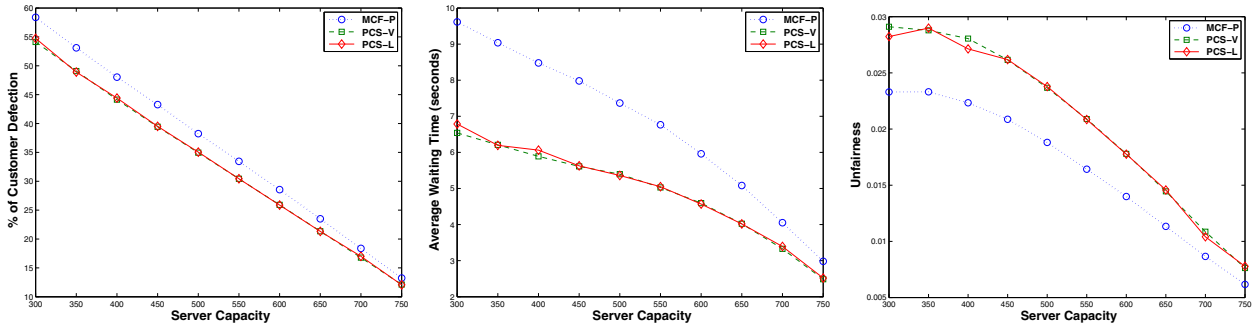
[3] B. Qudah and N. J. Sarhan, "Towards scalable delivery of video streams to heterogeneous receivers," in *Proc. of ACM Multimedia*, Oct. 2006, pp. 347–356.

[4] L. Shi, P. Sessini, A. Mahanti, Z. Li, and D. L. Eager, "Scalable streaming for heterogeneous clients," in *Proc. of ACM Multimedia*, Oct. 2006, pp. 337–346.



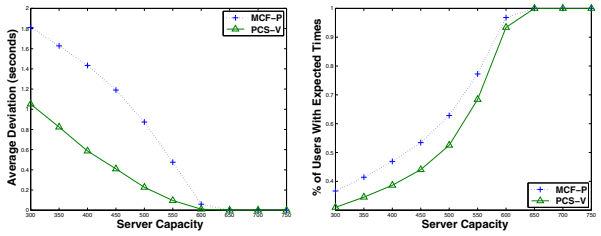
(a) Defection Rate (b) Average Waiting Time (c) Unfairness

Fig. 4: Effectiveness of PCS-V and PCS-L [*ERMT*, Model A]



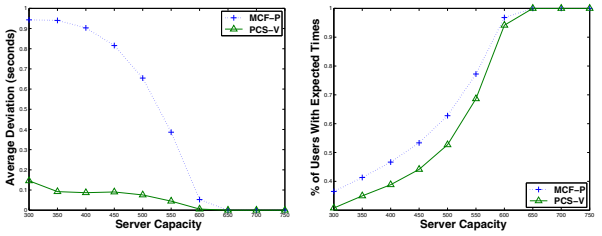
(a) Defection Rate (b) Average Waiting Time (c) Unfairness

Fig. 5: Effectiveness of PCS-V and PCS-L [*Patching*, Model A]



(a) Average Deviation (b) Customers with Expected Time

Fig. 6: Predictability of PCS-V vs. MCF-P [*ERMT*, $W_p = 0.5\mu_{tol}$, Model B]



(a) Average Deviation (b) Customers with Expected Time

Fig. 7: Predictability of PCS-V vs. MCF-P [*ERMT*, $W_p = 0.5\mu_{tol}$, Model C]

[5] N. J. Sarhan and B. Qudah, "Efficient cost-based scheduling for scalable media streaming," in *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, January 2007.

[6] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. of ACM Multimedia*, Oct. 1994, pp. 391–398.

[7] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "The maximum factor queue length batching scheme for Video-on-Demand systems," *IEEE Trans. on Computers*, vol. 50, no. 2, pp. 97–110, Feb. 2001.

[8] M. Alsmirat, M. Al-Hadrusi, and N. J. Sarhan, "Analysis of waiting-time predictability in scalable media streaming," in *Proc. of ACM Multimedia*, Sept. 2007, pp. 727 – 736.

[9] B. Qudah and N. J. Sarhan, "Analysis of resource sharing and cache management techniques in scalable video-on-demand," in *Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept. 2006, pp. 327–334.