DESIGN AND ANALYSIS OF SCALABLE AND INTERACTIVE NEAR VIDEO-ON-DEMAND SYSTEMS

Kamal K. Nayfeh and Nabil J. Sarhan

Electrical and Computer Engineering
Wayne State University
Detroit, Michigan 48202
knayfeh,nabil@wayne.edu

1. ABSTRACT

The design of interactive Video-on-Demand (VOD) systems is highly complicated when scalable stream merging is used. Support of requests under the Near Video-on-Demand (NVOD), where not all requests are serviced immediately, introduces additional complications. We develop an overall solution for designing interactive NVOD systems when stream merging is employed. The proposed solution supports user interactions with short response times and low rejection probabilities. In addition, we propose an efficient dynamic stream allocation policy and use a sophisticated client-side cache management policy. We analyze system performance, through extensive simulation, using realistic workload under different levels of user interactivity.

Index Terms— interactive requests, stream merging, Near Video-On-Demand (NVOD), multicast, realistic workload, True Video-On-Demand (TVOD), video-on-demand (VOD), video streaming

2. INTRODUCTION

The interest in multimedia streaming has grown substantially recently. Many applications have been developed to take advantage of this technology, including Video-on-Demand (VOD). VOD is expected to replace both broadcast-based TV and DVD rentals at stores.

The high network and server requirements limit the scalability of media streaming. The most common approaches used to address the scalability challenge are Content Distribution Networks and Peer-to-Peer [1]. While the first approach requires maintaining a huge number of geographically distributed servers [2], the second still relies heavily on central servers [3, 4]. Both of these approaches mitigate the scalability problem but do not eliminate it because the fundamental problem is due to unicast delivery [4]. As multicast is highly effective in delivering high-usage content and in handling flash crowds, there has been a growing interest in enabling native multicast to support high-quality on-demand video distribution, IPTV, and other major applications [4]. Moreover, applying multicast delivery schemes in fully owned networks like TV cable networks, dish networks, and universities is totally applicable. In this paper, we consider the multicast-based approach. Many resource sharing techniques [5, 6, 7] have been proposed to utilize the multicast facility. Batching [8] accumulates requests for the same video and services them together using multicast streams. Stream merging techniques combine streams to reduce the cost per request.

Providing successful VOD services requires support for interactive requests, which include requests not only to playback a video, but to Pause, Resume, Jump Backward, and Jump Forward. The overwhelming majority of prior studies used simple workload, which can be described as follows. (1) Media objects are homogeneous in type, length, and bit rate. (2) Objects are accessed sequentially from the beginning to the end without interactions. (3) The clients have homogeneous resources. More recent characterization studies [9, 10], however, reveal that the actual workload is more complex and dynamic. In particular, the workload contains many interactive operations, and a user may access an object from a point other than the beginning and may stop before the end. In addition, customer's behavior varies with media type, content type, and object length.

Only few studies have considered interactive VOD systems. In particular, using dedicated channels, called Interactive Streams or I-Streams, for servicing interactive requests was proposed by [11] to improve the customer experience in interactive workloads. These studies assumed all requests are serviced immediately (True VOD or TVOD) [6] or did not use stream merging [12]. Unfortunately, the stringent requirement of TVOD is hard to meet. (A NVOD system automatically converges to True VOD (TVOD) when the resources become sufficiently high.) Moreover, these studies dedicate a static portion of the server network capacity to I-Streams, and thus do not respond appropriately to dynamic workload variations.

Designing an interactive NVOD system is highly complicated when stream merging is used as interactions may cause clients to leave current video streams. In this paper, we study the design of interactive NVOD systems that support stream merging. The proposed solution enhances customer-perceived quality-of-service (QoS) by servicing requests quickly and ensuring pleasant and smooth playbacks. It also supports user interactions and realistic access patterns with short response times and low rejection probabilities. We generalize the Split and Merge protocol to work with stream merging techniques such as Patching and ERMT. In addition, we propose an efficient dynamic I-Stream allocation policy, which adjusts the number of I-Streams and B-Streams according to the system requirement at the moment. This policy allows the server to respond to workload variations. Furthermore, we use a sophisticated client-side cache management policy to maximize the percentage of interactive requests serviced from the client's own cache, which reduces the load on the server.

We evaluate through extensive simulations the effectiveness of major resource sharing techniques under NVOD using a realistic workload. We compare the system's performance under different levels of customer interactivity. We study the effect of a wide range of system parameters on the clients' waiting and blocking metrics. These parameters range from server network capacity, resource sharing technique, scheduling policy, number of I-Streams, interactivity level, dynamic I-Stream allocation. Furthermore, we develop a new metric,

called *Aggregate Delay*, to quantify the average delay experienced by a client due to both waiting and blocking during a streaming session.

The rest of this paper is organized as follows. Section 3 discusses the background information and related work. Section 4 presents the proposed solution. Section 5 discusses the performance evaluation methodology. Finally, Section 6 presents and analyzes the main results.

3. BACKGROUND INFORMATION AND RELATED WORK

Resource Sharing

In this subsection, we discuss three main resource sharing techniques: Batching, Patching, and ERMT. The simplest resource sharing technique is Batching [8], which accumulates requests to the same video and services them using a multicast stream. Clients usually need to wait before the video playback starts.

Stream merging techniques classify customers into groups of requests to the same streams. Patching [7] and references within joins new requests into the latest multicast stream to the same video. The server sends the missing portion to the new requests using a stream called a patch. When the client finishes playing the patch stream, the client continues playing from the already buffered data. It is more cost effective to start a full stream after some time. ERMT [5] is a near optimal stream merging technique. A new client or a newly merged group of clients listens to the closest stream it can merge with if no later arrivals can preemptively catch them. The target stream can later get extended to satisfy the needs of the new client (only after the merging stream finishes and merges with the target), and this extension can affect its own merge target. The three resource sharing techniques differ in complexity. Batching is the simplest to implement, since it starts a new full stream for every group of requests to a certain video. It does not allow for stream merging. Patching is next in complexity because it allows only one merge per client and allows only regular streams to be shared.

Request Scheduling

The server maintains a queue for waiting customers. When a new request arrives, the request is serviced immediately if there is enough server capacity. Otherwise, the request is queued with similar requests to the same video to be serviced later when server resources become available according to a Scheduling Policy. All requests to a certain video can be serviced using one channel. A channel is a set of server resources (network bandwidth, disk I/O bandwidth, etc...) needed to deliver a multimedia stream.

All scheduling policies try to optimize one of the following objectives: (i) Minimize the overall client defection probability. (ii) Minimize the average waiting time, and (iii) Minimize unfairness.

The defection probability is the likelihood that a new customer leaves the server without being serviced because the waiting time exceeded the user's tolerance. It is the most important metric because it translates into the number of concurrently serviced customers and the server throughput. The second and third objectives are indicators of the perceived Quality of Service (QoS). It is desirable that servers treat all requests for different videos equally. Unfairness measures the bias of a policy against unpopular videos. There are several studies on scheduling policies [13, 4, 14] for VOD servers. The most popular policies are *First Come First Come First Serve* (FCFS) [13], *Maximum Queue Length* (MQL) [13], and *Maximum Cost First* (MCF) [14] and references within. FCFS selects the queue with oldest request. It is the fairest policy. MQL selects the queue with the largest number of requests. It tries to maximize the number of requests that can be serviced with one channel. MCF selects the queue with the minimum cost in terms of stream length.

Interactive Streams (I-Streams) Handling

An I-Stream is a dedicated unicast stream used to service interactive requests only. The server utilizes I-Streams when it cannot service the request any other way. I-Streams can start from any playback point in the video and can last as long as the customer needs it. Since I-Streams are unicast, they cannot be shared with other requests. The Split and Merge (SAM) protocol [11] was proposed to service interactive requests when using Batching. As soon as a client issues an interactive request, the client is split from the multicast stream, and temporarily assigned an I-Stream to perform the interaction. Once the interaction is complete, the client is merged into an on-going stream. For Pause interaction, no I-Stream is required and the user is merged into an on-going stream as soon as the user resumes.

4. PROPOSED SOLUTION

Considered System

Figure 1 illustrates the proposed system design. The system consists of a VOD server, clients streaming videos from the server, and the network connecting the two. The major components of the VOD server are: waiting queues; one queue for each video, a blocking queue, a queuing manager, a dynamic I-Stream allocation module, a Split and Merge (SAM) technique, a resource sharing protocol, a waiting scheduling policy, a blocking scheduling policy, and streams. The streams are divided into full length multicast streams (B-Streams), multicast patch streams (P-streams), and unicast interactive streams (I-Streams). B-Streams and P-Streams are used to service waiting customers and can be shared with other requests to the same video. An I-Stream is a dedicated unicast stream used to service interactive requests only, and therefore cannot be shared with other requests. The server utilizes the SAM technique to manage I-Streams. When the I-Stream finishes delivering the requested data, the server frees the I-Stream, which becomes available for use by other requests.

The customer starts a streaming session by issuing a request to playback a certain video. The server makes the decision whether it can service the request or not based on the number of available server channels. If there are enough server channels, the server starts streaming the video to the customer using a resource sharing technique (Batching, Patching, or ERMT). If there are not enough network resources, the customer's request is placed in the waiting queue for that video. The server applies a waiting scheduling policy (FCFS, MQL, or MCF) to determine which waiting queue to service when streams become available. Waiting customers defect if they stay in the waiting queue for too long. During a streaming session, the customer issues interactive requests to pause the video, jump forward, or backward by a certain amount relative to the current playback position. Since pause and resume requests do not require any additional resources, they are serviced immediately. Jump requests are either serviced immediately if any additional resources are available, or block. Once a customer's request blocks, the customer stops listening to all streams. the server applies a blocking scheduling policy to service blocked customers when server channels become available. The waiting and blocking scheduling policies are not necessarily the same. Since blocked requests exhibit a different aggregation behaviour and the server's objective is to minimize the blocking time, the server applies the FCFS scheduling policy to blocked requests. Like wait-

ing customers, blocking customers defect if they stay in the blocking queue for too long.

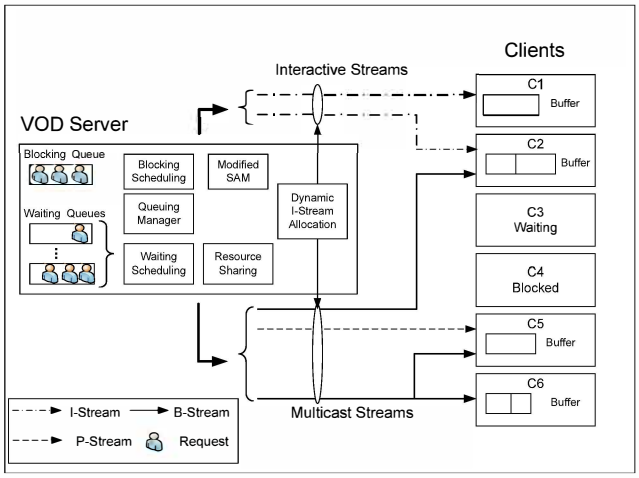


Fig. 1: Proposed System Design

Support for Interactive Requests

Figure 2 illustrates the methods by which the customer's jump requests are serviced: (1) The customer utilizes the cache. Each customer allocates a certain amount of memory to be used for stream merging, and to keep old data for future use. If the target playback point falls inside the customer's cache, the interactive request is serviced from the cache without demanding any extra resources from the server. (2) The server merges the request with an already existing B-stream for the same video. For a successful merge, the target stream playback point should be within a certain tolerance (playback point deviation tolerance) of the target playback point. (3) The server utilizes an available I-Stream. The server tries to minimize the skewness between the target playback point and the to be merged with B-stream playback point to minimize the I-Stream length. While an I-Stream is used by a certain customer, the server keeps searching B-Streams to merge the customer with. Upon a successful merge, the server frees the I-Stream. The order of the methods mentioned above depends on the type of interactive request and the streams currently used by the customer.

Dynamic I-Stream Allocation Policy

We generalize the Split and Merge (SAM) protocol to work with stream merging techniques (Patching, and ERMT) and take advantage of the buffer used by these techniques. Since the workload varies over time and hence the system's requirement of I-Streams changes accordingly, we propose a dynamic I-Stream allocation policy. This policy is guided by one of two objectives. (1) Minimize the aggregate delay experienced by the customer, which accounts for both the average waiting time and the average blocking time. (2) Minimize the overall customer defection probability, which encompasses waiting defection and blocking defection probabilities.

Figure 3 illustrates the proposed dynamic I-Streams allocation policy. Since the overall server capacity is fixed, this policy works by periodically adjusting the relative ratio of I-Streams and B-Streams in response to workload variations. Any addition to the I-Streams must be accompanied by a subtraction of the same amount to the B-Streams. The AdjustPeriod is the time interval between two consecutive adjusts of the relative ratio of I-Streams and B-Streams. The AdjustPeriod could be kept small to respond quickly to workload variations in rapidly changing workloads, or it could be

```
ServiceJumpRequest() {
  // Try to service request from client buffer
  for (s = 0; s < NumSegments; s++){
    if (Target≤Seg[s].end && Target≥Seg[s].start) {
      Service request from segment s ; break ; } //for
  // Try to merge with another B-Stream
  MinSkew = MAXFLOAT ; ClosestStream = −1 ;
  for (i = 0; i < NumStreams; i++){
    Skew = |Target − Stream[i].Playback| ;
    if (Skew < DeviationTolerance) and (Skew < MinSkew)
      ClosestStream = i ; } //for
  if (ClosestStream ≥ 0) {
    Client Stop listening to current stream;
    Merge request with Stream ClosestStream ;
    if (stream[current].NumListeners == 0) free current stream;
    break ;
  } //if
  // Try to service request with an I-Stream
  if (UsedIStreams < AvailableIStreams){
    MinSkew = MAXFLOAT ; ClosestStream = −1 ;
    for (v = 0; v < NumStreams; v++){
      Skew = |Target − streams[v].Playback| ;
      if (Skew < MinSkew) ClosestStream = v ; } //for
  } //if
  if (ClosestStream ≥ 0) {
    Client stop listening to current stream;
    Client start listening to ClosestStream stream;
    Client Start listening to I-Stream;
    if (streams[current].NumListeners == 0)
      free current stream ;
    break ;
  } //if
  else{
    Client stop listening to all streams ;
    Place request in blocking queue; }
} //ServiceJumpRequest
```

Fig. 2: Simplified Algorithm for Servicing Jump Forward and Jump Backward Requests

set to a moderate value in more stable workloads to prevent the server from over-reacting to temporary workload changes. The MaxAdjust is the maximum number of I-Streams that could be added or subtracted in one AdjustPeriod. The server limits the adjust amount each AdjustPeriod to prevent the I-Streams from rapidly oscillating back and forth, which degrades the system performance. The server calculates the average blocking time, the average waiting time, and the aggregate delay during every AdjustPeriod. The blocking time is given more importance than waiting time. The significance (weight) of the blocking time is doubled every instance the customer blocks in the same session. The server increases or decreases the I-Streams every AdjustPeriod based on the aggregate delay calculated during the current and previous AdjustPeriods. If the aggregate delay improved (became smaller) in the current AdjustPeriod compared to the previous AdjustPeriod, the server continues adjusting in the same direction (increase/decrease number of I-Streams) as it did in the last AdjustPeriod. Otherwise, the server reverses the adjust direction. The AdjustFactor determines how much change in the aggregate delay corresponds to how much change in the I-Streams. The AdjustFactor could be set to a more aggressive value for rapidly changing workloads and it could be kept at a moderate value for steady workloads.

Cache Management

We implement a sophisticated cache management policy to maximize the client cache utilization. The client's cache consists of one or more segments of data. Each segment is a contiguous set of video data. The data kept in the cache is fu-

```
DynamicIStreamAlloc() {
Direction_last = 1;
  for(c = 0; c < NumWaiting; c++){
    TotBlkingTime+ = 2^BlkingNo[c] * BlkingTime[c];
    TotWaitingTime+ = WaitingTime[c];
  } //for
  AvgWaitingTime = TotWaitingTime/NumWaiting;
  AvgBlkingTime = TotBlkingTime/NumBlking;
  AggrDelay_Curr = AvgWaitingTime + AvgBlkingTime;
  Diff = AggrDelay_Curr − AggrDelay_Prev;
  if(Diff < 0)
    Direction_Curr = Direction_Last;
  else
    Direction_Curr = −1 * Direction_Last;
  AdjustAmnt = (Diff/AggrDelay_Curr) * AdjustFactor;
  if(AdjustAmnt > MaxAdjust)
    AdjustAmnt = MaxAdjust;
  BStreams = BStreams + Direction * AdjustAmnt;
  IStreams = IStreams − Direction * AdjustAmnt;
  AggrDelay_Prev = AggrDelay_Curr;
  Direction_Curr = Direction_Last;
} //DynamicIStreamAlloc
```

Fig. 3: Simplified Algorithm for Dynamically Adjusting I-Streams Executed Periodically after AdjustPeriod Time

ture data for stream merging techniques, such as Patching, or past data the customer watched already. Both future and past data are utilized to service interactive requests. As the cache becomes full, the oldest data is purged. When two segments overlap, they are merged to form one contiguous segment.

5. PERFORMANCE EVALUATION

Simulation Platform

We developed a simulator for both the client-side caches/buffers and the VOD server which models various resource sharing and scheduling techniques. We consider the NVOD server model. The simulator stops after a steady state analysis with 95% confidence interval is reached. The simulator allows for the evaluation of a wide range of system parameters. However, we did not include all results due to space limitations.

Client and Server Characteristics

Table 1 summarizes the default parameters used. We characterize the waiting and blocking tolerance of customers as Poisson with a mean of 30 seconds. We examine the server at different loads and varying server capacities from 147 Gbps to 290 Gbps. 10% of the server capacity is reserved for I-Streams. We use MCF-PN scheduling policy for waiting customers and FCFS for blocking customers except when evaluating the effect of scheduling policies. The playback deviation point tolerance is kept at 10 seconds. The default client side cache size dedicated to the video streaming application is kept at 100 MB.

Workload Characteristics

The average arrival rate (λ) is 30 requests per minutes. We utilize the results of [9], which characterizes the workload of a media streaming server using actual access logs. We study 100 videos of varying lengths and request rates. The workload indicates that 91% of the requested files have average bit-rates of 300-350 Kbps. The average bit-rates of the remaining files are in the range of 200-300 Kbps. To be more reflective of recent video bit-rates, we use 2.1 Mbps for the first group and 2.45 Mbps for the latter.

Performance Metrics

In NVOD, we consider seven performance metrics: *average waiting time, waiting defection probability, unfairness against unpopular videos, blocking defection probability, average blocking time, blocking probability, and playback point deviation*. The waiting defection probability can be defined as the probability that the customer leaves the server because the waiting time exceeded the customer's tolerance. Likewise, the blocking defection probability can be defined as the probability that the customer leaves the server without finishing watching the video because the customer stayed in the blocking queue longer than its tolerance. The average waiting time is the average time the customer stays in the waiting queue. Likewise, the average blocking time is the average time the interactive request stays in the blocking queue. The blocking probability is defined as the likelihood of an interactive request to block. The playback point deviation is a measure of how close the requested target playback point to the actual playback point.

Table 1: Default Parameters Values

Parameter	Default Value(s)
Arrival Rate	30 Requests / minute
Number of Videos	100
Waiting Tolerance Model	Poisson with mean = 30 sec.
Blocking Tolerance Model	Poisson with mean = 30 sec.
Server Capacity	147 - 290 Gbps
Video Bit-rate	$2.1 - 2.45$ Mbps
I-Streams	10% of server capacity
Buffer Size	100 MByte
Playback Point Deviation Tolerance	10 seconds
Waiting Scheduling Policy	MCF-PN
Blocking Scheduling Policy	FCFS

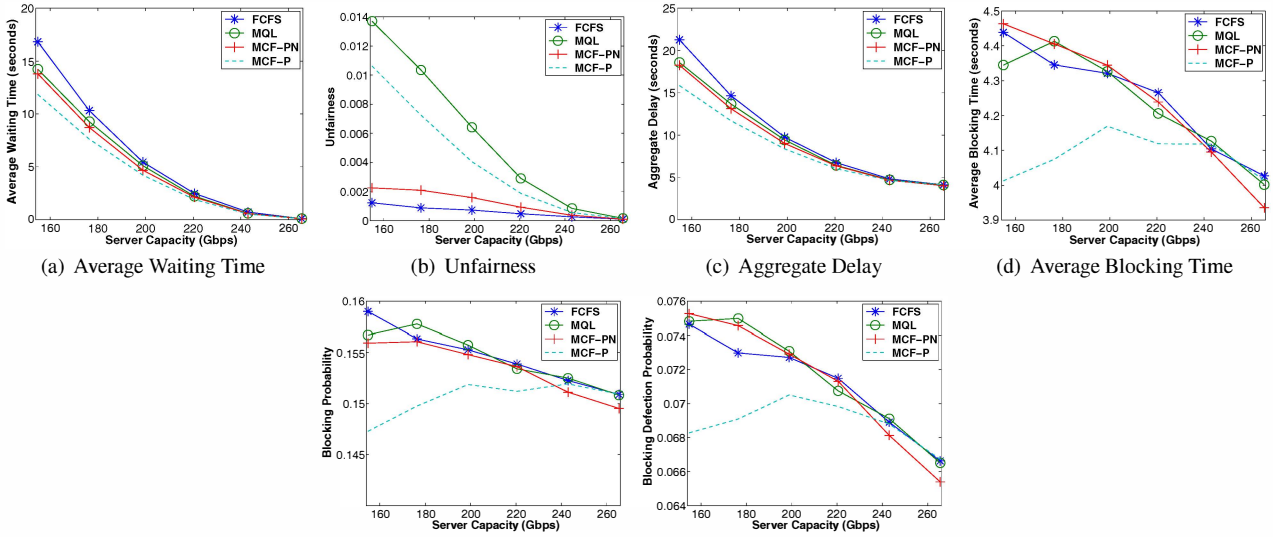
6. RESULT PRESENTATION AND ANALYSIS

We analyze the system performance through extensive simulation. We primarily consider NVOD server model. The Near Video On Demand (NVOD) model converges to True Video On Demand (TVOD) model when the server capacity becomes sufficiently high. (average waiting time = 0, waiting defection probability= 0, average blocking time = 0, blocking defection probability= 0). We pay special attention in our analysis for the issues introduced by realistic interactive workloads (blocking metrics) since this is an area that has not been investigated by previous studies.

Impact of the Scheduling Policy

Figure 4 compares the scheduling policies. FCFS and MQL perform worse than MCF-P in terms of waiting and blocking metrics. MCF-P selects the queue with the least cost per request. Hence, it selects the queue with the many waiting customers that require a short stream to service. Since shorter videos are more popular and tend to have shorter patches than longer videos, MCF-P is biased towards shorter popular videos and unfair against longer videos. Since longer videos are less popular and receive more interactive requests than shorter videos, they tend to block more often with longer blocking times than shorter videos. MCF-P outperforms other scheduling policies it terms of average waiting time and waiting defection probability because it minimizes the cost per request. MCF-P outperforms other scheduling policies in terms of blocking metrics also because it admits relatively more shorter videos requests that tend to block less with shorter blocking times.

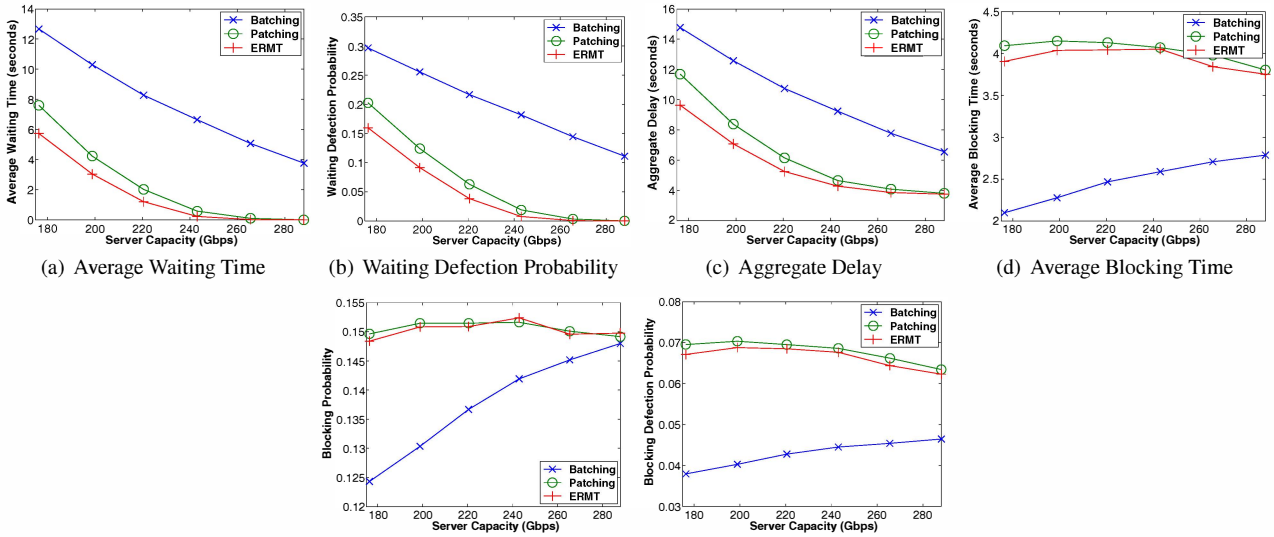
At lower server capacity, MCF-P admits a high percentage of shorter videos which tend to block less with shorter blocking times. Hence, the blocking metrics are kept relatively small. As the server capacity increases, more resources become available to entertain requests to longer videos. Thus, the blocking metrics worsen as we are increase the server capacity. This trend keeps going until we reach a server capacity where the relative number of requests to longer videos is not



(a) Average Waiting Time (b) Unfairness (c) Aggregate Delay (d) Average Blocking Time

(e) Blocking Probability (f) Blocking Defection Probability

Fig. 4: Comparing Effectiveness of Scheduling Policies



(a) Average Waiting Time (b) Waiting Defection Probability (c) Aggregate Delay (d) Average Blocking Time

(e) Blocking Probability (f) Blocking Defection Probability

Fig. 5: Comparing Effectiveness of Resource Sharing Techniques

increasing. Then the blocking metrics start improving.

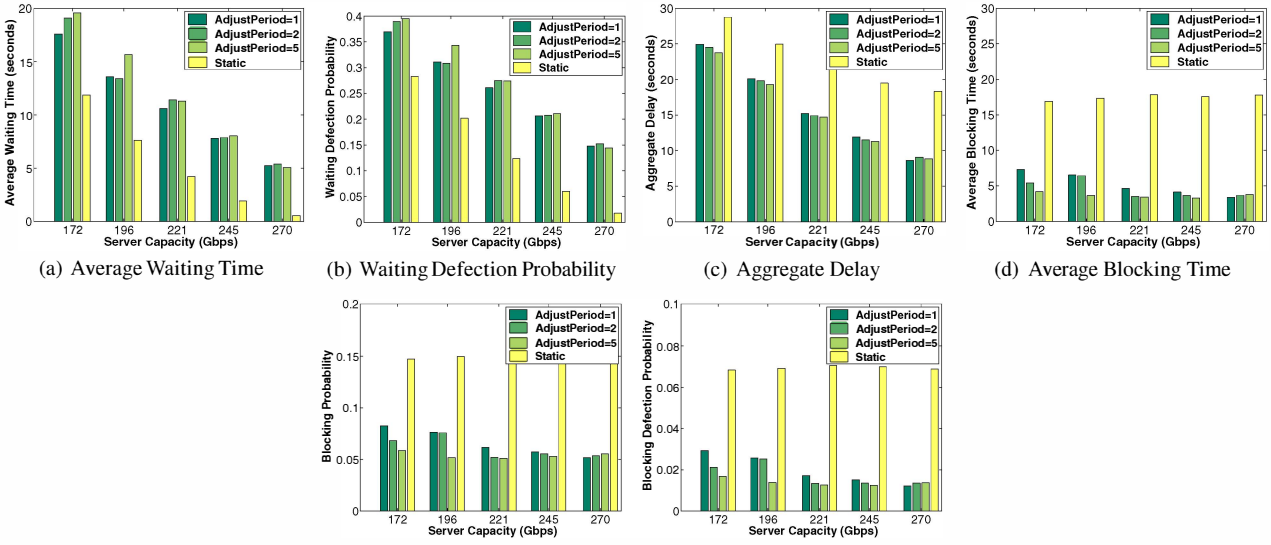
The main disadvantage of MCF-P is it is unfair as illustrated in figure 4(b). To overcome the unfairness of MCF-P, we experiment with normalizing the stream length required to service a request by the video length. We call this normalized scheduling policy (MCF-PN). MCF-PN takes into consideration the number of requests for each video and the stream length required to service a request divided by the video length. This normalization addressed the fairness issue. At the same time, it delivers a descent performance in terms of both waiting and blocking metrics as illustrated in figure 4. Therefore, we used MCF-PN for all subsequent sections.

Impact of the Resource Sharing Techniques

As shown in previous studies, ERMT outperforms other resource sharing techniques in terms of waiting metrics, followed by Patching as illustrated in Figures 5(a) and 5(b). Since we consider only B-Streams for merging when servicing blocked customers, Batching outperforms the other techniques in terms of the blocking defection and the blocking time because all Batching streams are B-Streams. However, in terms of the blocking probability, Patching outperforms Batching because Patching tends to buffer more future content leading to more interactive requests, especially jump forward to be serviced from the client's buffer.

Effectiveness of the Proposed Dynamic I-Streams Allocation

Figure 6 illustrates the effectiveness of the proposed dynamic I-Stream allocation policy and Patching technique. Using dynamic I-Stream allocation outperforms the static allocation of I-Streams in terms of the blocking metrics and average aggregate delay experienced by the client. Up to 50% improvement in the average aggregate delay is achieved. At lower server capacity, dynamically adjusting resources shows little improvement because the system favors one group of customers over another. Thus, either waiting or blocking customers suffer. Moreover, there is a cost associated with the process of switching resources. However, at higher server capacity, which is the preferred range of operation, this policy shows very significant improvements in the system performance. Using this policy with Batching and ERMT techniques showed similar results. The results were omitted for



(a) Average Waiting Time (b) Waiting Defection Probability (c) Aggregate Delay (d) Average Blocking Time

(e) Blocking Probability (f) Blocking Defection Probability

Fig. 6: Effectiveness of Dynamic I-Streams Allocation

space limitations.

7. CONCLUSION

We have studied the performance of NVOD servers using various resource sharing techniques and scheduling policies using a realistic workload. We evaluated the system performance using extensive simulations. The main results can be summarized as follows. (1) ERMT outperforms other techniques in terms of waiting metrics. Unlike previous studies, which studied NVOD servers using simple workloads, At low to moderate request rates the amount of improvement over Patching is insignificant. We conclude that there are fewer opportunities for stream merging in the presence of interactive requests. Given that ERMT is of higher implementation complexity, we recommend using Patching for systems with low to moderate request rates. (2) Blocking is a major issue for any NVOD server. An efficient NVOD server should continuously monitor the system performance and shift resources in order to minimize blocking. We proposed a new I-Streams allocation policy, where I-Streams are adjusted dynamically based on the system's requirement at the moment. Using our proposed policy reduces the blocking probability by 75% and the blocking defection to a virtual zero. Up to 50% reduction in the average aggregate delay experienced by the client is achieved. (3) Deciding which group of waiting customers to serve has a significant effect on the NVOD server performance. We proposed a new scheduling policy (MCF-PN), which addresses the unfairness issue of (MCF-P) and delivers a near optimal performance. (4) Using our proposed cache management policy, up to 88% of all interactive requests are serviced from the client's own buffer without requiring additional server resources.

8. REFERENCES

[1] Zhijie Shen, Jun Luo, Roger Zimmermann, and Athanasios Vasilakos, "Peer-to-peer media streaming: Insights and new developments," in *Proceedings of the IEEE 99(12)*, 2011, pp. 2089–2109.

[2] George Pallis and Athena Vakali, "Insight and perspectives for content delivery networks," *Communications of the ACM*, vol. 49, pp. 101–106, January 2006.

[3] B. Li C. Wu and S. Zhao, "Diagnosing network-wide p2p live streaming inefficiencies," in *Proc. of IEEE INFOCOM*, April 2009.

[4] V. Aggarwal, R. Caldebank, V. Gopalakrishnan, R. Jana, K. Ramakrishnan, and F. Yu, "The effectiveness of intelligent scheduling for multicast video-on-demand," in *Proc. of ACM Multimedia*, 2009, pp. 421–430.

[5] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective Video-on-Demand," in *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, Jan. 2000, pp. 206–215.

[6] Marcus Rocha, Marcelo Maia, Italo Cunha, Jussara Almeida, and Sergio Campos, "Scalable media streaming to interactive users," in *Proc. of ACM Multimedia*, Nov. 2005, pp. 966–975.

[7] Sung Soo Moon, Kyung Tae Kim, Seong Woo Lee, Hee Yong Youn, Ohyoung Song, and Ohyoung Song, "An efficient vod scheme combining fast broadcasting with patching.," in *ISPA*, 2011, pp. 189–194.

[8] 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.

[9] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto, "Analyzing client interactivity in streaming media," in *Proc. of The World Wide Web Conf.*, May 2004, pp. 534–543.

[10] F. Qiu and Y. Cui, "An analysis of user behavior in online video streaming," in *Proceedings of ACM International Workshop on Very-Large-Scale Multimedia Corpus, Mining and Retrieval*, Oct. 2010, pp. 49–54.

[11] Wanjium Liao and Victor O. K. Li, "The split and merge (sam) protocol for interactive video-on-demand systems," in *The 16th Annual Joint Conference of the IEEE Computer and Communications Societies Proceedings*, Apr. 1997, pp. 1349–1356.

[12] Emmanuel L. Abram Profeta and Kang G. Shin, "Providing unrestricted vcr functions in multicast video-on-demand servers," in *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, July 1998, pp. 66–75.

[13] 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.

[14] Nabil J. Sarhan, Mohammad A. Alsmirat, and Musab Al-Hadrusi, "Waiting-time prediction in scalable on-demand video streaming," *ACM Transactions on Multimedia Computing, Communications, and Applications (ACM TOMCCAP)*, vol. 6, no. 2, pp. 1–24, March 2010.