Synergistic Memory Management and Disk Scheduling in the HYDRA Media Stream Recording System

Kun Fu and Roger Zimmermann
Integrated Media Systems Center, University of Southern California
Los Angeles, CA, USA 90089-2561
[kunfu, rzimmerm]@usc.edu

ABSTRACT
Presently, digital continuous media (CM) is well established as an integral part of many applications. In recent years, a considerable amount of research has focused on the efficient retrieval of such media. Scant attention has been paid to servers that can record such streams in real time. However, more and more devices produce direct digital output streams. Hence, the need arises to capture and store these streams with an efficient data stream recorder that can handle both recording and playback of many streams simultaneously and provide a central repository for all data. Memory and disk bandwidth are two of the most crucial resources in a recording system. Because of the continuously decreasing cost of memory, more and more memory is available on a large scale recording system. Unlike most previous work that focuses on how to minimize the server buffer size, this paper investigates how to effectively utilize the additional available memory resources in a recording system through an appropriately designed deadline setting policy (DSP) for disk scheduling. We propose an effective resource management framework that is composed of two parts. The first one is a dynamic memory allocation strategy, which applies to different playback and recording streams. The other one is a deadline setting policy that can be applied consistently to both playback and recording streams, satisfying the timing requirements of continuous media, and also ensuring fairness among different streams. Furthermore, in order to find the optimal memory configuration, we construct a probability model based on the classic $M/G/1$ queuing model and the recently developed Real Time Queuing Theory (RTQT). Our model can be used to predict (a) the missed deadline probability of a playback stream with a given memory allocation, and (b) the blocking probability of recording streams. The model is applicable to admission control and capacity planning in a recording system.

Keywords: continuous media recording system, memory management, EDF scheduling

1. INTRODUCTION
Digital continuous media (CM) is an integral part of many new applications. Two of the main characteristics of such media are that (1) they require real-time storage and retrieval, and (2) they require high bandwidth and space. Over the last decade, a considerable amount of research has focused on the efficient retrieval of such media for many concurrent users. However, more and more devices produce direct digital output streams. Hence, the need arises to capture and store these streams with an efficient data stream recorder that can support many concurrent recording and playback streams simultaneously and provide a central repository for all data.

Memory and disk bandwidth are two of the most important resources in a recording system. Most of the previous studies,\textsuperscript{1–3} are based on the minimum buffer settings. With the continuously decreasing cost of memory, a large memory capacity can be available on a large scale recording system. In this paper, we study the memory management and disk scheduling issues in a media stream recording system, HYDRA,\textsuperscript{4} which has limited, but more than minimal, memory resources. More specifically, we are interested to answer the following questions: (1) How should we allocate the buffer pool among playback and recording streams? (2) Can we allocate more buffers to one stream compared with others so that the stream with more resources gets better QoS? (3) Can we find an optimal buffer allocation configuration to maximize the system throughput while satisfying the client's QoS requirements?

To answer these questions, we propose an effective resource management framework that is composed of two parts. The first component is a dynamic memory allocation strategy, which applies to different playback and recording streams. The

\textsuperscript{∗}This research has been funded in part by NSF grant EEC-9529152 (IMSC ERC), CMS-0219463 (TR), and equipment gifts from the Intel Corporation, Hewlett-Packard, Sun Microsystems and Raptor Networks Technology.
second part is a deadline setting policy (DSP) that can be applied consistently to both the playback and recording streams, satisfying the timing requirements of continuous media streams, and also ensuring fairness among different streams. Furthermore, in order to find the optimal memory configuration, we construct a probability model based on the classic $M/G/1$ queueing model and the recently developed Real Time Queueing Theory (RTQT). Our model can be used to predict (a) the missed deadline probability of a playback stream with a given memory allocation, and (b) the blocking probability of recording streams. The model is applicable to admission control and capacity planning in a recording system. To the best of our knowledge, the probability model we have constructed is the first model that applies RTQT in the timing analysis of a multimedia streaming system. Specifically, our model characterizes: (1) deadline driven disk scheduling, (2) random data placement, (3) variable bit rate (VBR) streams, (4) concurrent reading and writing of streams, (5) the difference in disk read/write bandwidth, (6) the transfer rate variability in multi-zoned disks, and (7) the variable seek time and variable rotational latency of disk I/O operations.

The remainder of this paper is organized as follows. In Section 2 we review some of the related work. Section 3 describes our proposed dynamic buffer sharing scheme and in Section 4 we discuss our design and analysis of deadline setting policies. Section 5 includes the probability model we have constructed to evaluate the performance trade-off between buffer allocation and QoS of end users. In Section 6 we present our evaluation results. Finally, Section 7 concludes the paper and presents some future research directions of this work.

2. RELATED WORK

There are only a few papers that have studied both disk read and write issues for multimedia streaming architectures. Most of these techniques assume a minimum server buffer environment (i.e., double buffering) except. Aref et al. proposed a technique that dynamically computes the deadlines for disk writing requests under a shared server write buffer for video editing systems. Their technique assigns the same deadline to all the writing requests and consistently adjusts the deadline based on the available amount of buffers in the system. Since they constantly modify the deadline, the overhead of reorganizing the request waiting queue can be very high when the system experiences workload fluctuations. Moreover, since they are targeting video editing applications, higher priority is given to read requests compared with write requests. Rangaswami et al. investigated the design of an interactive media server based on a fine-grained device management strategy, which consists of three components: disk profiler, data placement, and I/O scheduler. They adopted a very constrained data placement technique for multi-zone disks and their approach is targeted to MPEG movie streams with limited VCR-like controls. They also adopted the double buffering scheme in their system. In our initial memory management paper we have studied the minimum server buffer (MSB) problem, which minimized the buffer size with a given service requirement. Ghandeharizadeh and Kim had proposed a multi-buffer technique to minimize the hiccup probability in a continuous media editing server.

To our knowledge, no prior work has investigated the fairness issue of deadline setting policy with respect to the performance of streaming applications, and no prior work has studied the issue of dynamic allocation of buffer size to different streams in the context of recording systems. Furthermore, no prior work has constructed a probability model that can evaluate performance trade-offs between different deadline setting policies and buffer allocation on the recording system.

3. DYNAMIC MEMORY BUFFER SHARING

System Architecture Figure 1 shows the simplified system architecture of the HYDRA media stream recording system (see details in our previous paper). There are two generally accepted paradigms to assign data blocks to the magnetic disk drives that form the storage system: in a round-robin sequence, or in a random manner. Traditionally, the round-robin placement utilizes a cycle-based approach to scheduling of resources to guarantee the service quality, while the random placement utilizes a deadline-driven approach. The latter provides a number of advantages such as support for multiple or variable delivery rates with a single storage data block size, easy support for interactive applications, and support for data reorganization during storage system scaling. All these features may be supported with cycle-based scheduling, however, it results in a complex implementation and – most importantly – many of the disk parameters must be assumed with their worst case values. Therefore, deterministic guarantees are obtained at the expense of efficiency. Deadline-driven scheduling can be configured to be both very efficient and to incur a very low probability of disruptions. The HYDRA system adopts random data placement with an earliest deadline first (EDF) disk scheduling algorithm. Therefore, it can achieve high system utilization and provide statistical service guarantee.
Table 1 lists all the important parameters and their definitions used in this paper.

**Design Goals** To effectively utilize server memory for playback and recording streams, we identify the following desirable *Design Goals* for the buffer sharing scheme:

**DG1:** Share the memory buffers among playback and recording streams as much as possible to exploit the statistical multiplexing gain for a better system performance.

**DG2:** Dynamically allocate the available buffer to playback or recording streams on demand.

**Comparison of Buffer Usage for a Playback and a Recording Stream:** Consider the following scenario, if given \( n \) buffers, how would a playback and a recording stream use them to meet their quality of service (QoS) requirement. Note in such a recording system, the most important QoS criteria for a playback stream is the probability that the client has used up all the data in the buffer (i.e. all buffers are empty), which happens when disk I/O cannot fetch data in time. On the other hand, for a recording stream, the most important QoS criteria is the probability of a recording client finding that all the allocated buffers are used up (i.e. all buffers are full), which happens when a disk cannot write data in time. Therefore, to provide the highest QoS, a playback stream issues the next block read request immediately after a data buffer is consumed. Similarly, a recording stream initiates a block write request immediately after a data buffer becomes full. When a recording system is under normal workload, the buffer usage for a playback stream and a recording stream are shown in Figure 2. For the playback stream, one buffer (e.g. buffer 1) is holding the movie data that is currently being displayed, and another buffer (i.e. buffer \( n \)) is accumulating the data being retrieved from disks, and the rest of the buffers are full of retrieved movie data. The situation is different for a recording stream: one buffer (e.g. buffer 1) is accumulating data from a recording device (e.g. camera), buffer \( n \) is currently being writing to disk, while the rest of the buffers are empty.
Table I. List of terms used repeatedly in this study and their respective definitions.

**Buffer Sharing Scheme:** Based on the buffer usage comparison above, we make several important observations: (1) Sharing buffers among several playback streams is not feasible because the buffers allocated to each playback stream are usually occupied by the retrieved movie data as shown in Fig. 2a. (2) Sharing buffers among a playback and a recording stream is difficult because of the almost conflicting buffer usage of playback and recording streams. (3) Sharing buffers among several recording streams is possible, because under normal system workload, all the recording streams usually have plenty of empty buffers, these empty buffers could be easily made available to other recording streams.

Based on these observation, we decide to partition the total memory buffers dynamically for each playback stream respectively and for all the recording streams as a shared buffer pool. Let $M$ denote the total number of buffers available in the system. Let $M_R$ and $M_W$ denote the number of buffers allocated to playback and recording streams, respectively. Let $M_R(i)$ denote the number of buffers allocated to playback stream $i$, where $i$ is an index of all the playback streams.

Another important result we have obtained from our simulation is that each playback stream may require different buffer sizes to achieve the same QoS requirement, (i.e. the probability of a request missed deadline). Intuitively, this is because each stream may have different bandwidth requirements. For this reason, it is not desirable to evenly allocate the reading buffers $M_R$ among all playback streams.

Next, we need to determine the appropriate values for $M_R$, $M_W$ and $M_R(i)$, where $i \in [1, N_{R}]$. Let $N_{R}$ and $N_{WS}$ denote the number of concurrent playback and recording streams in the system. Our first approach is to allocate the

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{Request}(i)$</td>
<td>the issue time of the disk read request for block $i$ of a playback stream</td>
<td></td>
</tr>
<tr>
<td>$D_{Deadline}(i)$</td>
<td>the deadline of the disk read request for block $i$ of a playback stream</td>
<td></td>
</tr>
<tr>
<td>$D_{consume}(S(i,j))$</td>
<td>the stream data consumption duration of block $i$, $i + 1, \ldots, j$</td>
<td></td>
</tr>
<tr>
<td>$T_{playbackRequest}$</td>
<td>the time when a client sends a playback request to server</td>
<td></td>
</tr>
<tr>
<td>$T_{movieStart}$</td>
<td>the time when a client begins to watch the movie content</td>
<td></td>
</tr>
<tr>
<td>$T_{RequestFinish}(i)$</td>
<td>the finish time of the disk read request for block $i$ of a playback stream</td>
<td></td>
</tr>
<tr>
<td>$T_{DiskRequest}(i)$</td>
<td>the deadline of the disk write request for block $i$ of a recording stream</td>
<td></td>
</tr>
<tr>
<td>$D_{requestDeadline}(i)$</td>
<td>the deadline of the disk write request for block $i$ of a recording stream</td>
<td></td>
</tr>
<tr>
<td>$D_{recordDeadline}$</td>
<td>the deadline of the disk write request for block $i$ of a recording stream</td>
<td></td>
</tr>
<tr>
<td>$T_{recordStart}$</td>
<td>the time when a client begins to record (really generating data stream)</td>
<td></td>
</tr>
<tr>
<td>$T_{RecordFinish}(i)$</td>
<td>the number of available (empty) buffers for a recording stream at time $t$</td>
<td></td>
</tr>
<tr>
<td>$T_{current}$</td>
<td>the current time</td>
<td></td>
</tr>
<tr>
<td>$T_{Deadline}$</td>
<td>the deadline of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{Deadline}$</td>
<td>the remaining time until the deadline of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{requestStart}$</td>
<td>the issue time of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{requestEnd}$</td>
<td>the actual disk I/O service start time of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{requestFinish}$</td>
<td>the actual disk I/O service finish time of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{wait}$</td>
<td>the waiting time of a disk I/O request in the request waiting queue</td>
<td></td>
</tr>
<tr>
<td>$T_{service}$</td>
<td>the actual service time of a disk I/O request, i.e. $T_{service} = T_{requestFinish} - T_{requestStart}$</td>
<td></td>
</tr>
<tr>
<td>$T_{System}$</td>
<td>the system time of a disk I/O request, i.e. $T_{System} = T_{wait} + T_{service}$</td>
<td></td>
</tr>
<tr>
<td>$N_{RS}$</td>
<td>the number of concurrent playback streams</td>
<td></td>
</tr>
<tr>
<td>$M_{RS}$</td>
<td>the total number of buffers allocated to playback streams</td>
<td></td>
</tr>
<tr>
<td>$M_{WS}$</td>
<td>the number of concurrent recording streams</td>
<td></td>
</tr>
<tr>
<td>$M$</td>
<td>the total number of buffers available in a recording system</td>
<td></td>
</tr>
<tr>
<td>$M_R(i)$</td>
<td>the number of buffers allocated to playback stream $i$, $i$ is an index of playback streams in the system</td>
<td></td>
</tr>
<tr>
<td>$M_W$</td>
<td>the number of buffers allocated to recording streams</td>
<td></td>
</tr>
<tr>
<td>$\lambda$</td>
<td>the average arrival rate of disk reading requests</td>
<td></td>
</tr>
<tr>
<td>$\mu$</td>
<td>the average arrival rate of disk writing requests</td>
<td></td>
</tr>
<tr>
<td>$\lambda_R$</td>
<td>the average arrival rate of disk I/O requests, $\lambda = \lambda_R + \lambda_W$</td>
<td></td>
</tr>
<tr>
<td>$\mu_R$</td>
<td>the average bandwidth requirement of disk reading requests</td>
<td></td>
</tr>
<tr>
<td>$\mu_W$</td>
<td>the average bandwidth requirement of disk writing requests</td>
<td></td>
</tr>
<tr>
<td>$P_{read,missed}(i)$</td>
<td>the probability that a new disk write request is rejected due to the limitation of write buffer</td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>size of a buffer or a data block</td>
<td></td>
</tr>
<tr>
<td>$T_{DiskBegin}$</td>
<td>the finish time of the disk write request for block $i$ of a recording stream</td>
<td></td>
</tr>
<tr>
<td>$T_{DiskStart}$</td>
<td>the start time of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$T_{DiskEnd}$</td>
<td>the finish time of a disk I/O request</td>
<td></td>
</tr>
<tr>
<td>$h$</td>
<td>the maximum disk rotational latency</td>
<td></td>
</tr>
<tr>
<td>$w_{RS}, \tau_{RS}$</td>
<td>Disk parameters for read and write cycle</td>
<td></td>
</tr>
</tbody>
</table>
memory resources according to streams’ bandwidth requirements. Thus, we obtain

\[ M_W = M_{\sum_{i=1}^{N} \mu_i^R} \]  
\[ M_R(i) = M_{\sum_{i=1}^{N} \mu_i^W} \]  

(1)

where \( \mu_i^R \) and \( \mu_i^W \) denote the average bandwidth requirement for playback stream \( i \) and recording stream \( i \), respectively.

Note, we can obtain \( M_R \) as \( M_R = M - M_W \). Our second approach is to construct a probability model (in Section 5) that can be used to find the optimal configurations.

4. DEADLINE SETTING POLICY FOR N-BUFFERING SCHEME

When a stream is allocated with \( N \) buffers, the buffer allocation scheme is termed an \( N \)-buffering scheme. With more buffers allocated, there is more flexibility in configuring the disk I/O deadlines. In this section, we study how to set the deadlines for disk I/O requests in HYDRA. First, we establish the timing requirement for playback and recording streams respectively. Based on these timing requirements, we deduce the timing requirements for the deadline setting. Then, we propose several deadline setting policies that can satisfy these timing requirements. Note that we have included all the proofs of this section in Appendix A.

4.1. Timing Analysis for Continuous Media Streams with Multiple Buffers

From the storage system point of view, a playback/recording stream is essentially a sequence of disk read/write requests with real-time requirements. The real-time requirement for a disk read or write request can be characterized by two parameters: (1) the request issue time, and (2) the true deadline \(^1\) of the request.

**Definition 4.1.** The True Deadline of a disk I/O request is the deadline that must be satisfied in order to ensure the continuous playback and recording of streaming applications.

One of the design principles for the disk scheduling algorithm that we have chosen is to ensure the work-conserving property. Therefore, the disk I/O requests are always issued as early as possible. For a playback stream, due to the \( N \) buffer limitation, a block read request can not be issued before the data in an existing buffer are consumed. On the other hand, for a recording stream, a disk write request can be generated immediately after the recording stream fills up a block buffer.

![Timing information for a playback stream.](image)

**Figure 3.** Timing information for a playback stream.

**Timing Analysis for Playback Streams**  Figure 3 shows the disk read accesses during the playback of a movie with \( N \)-buffering scheme. \( T_{playReq,issue} \) represents the time client issued a play command (e.g. RTSP play), the server prefetched \( N-1 \) blocks before initiating client’s display at movie start time \( T_{movieStart} \). Note that at \( T_{movieStart} \), the server also issues the disk read request to fetch block \( N \), and the request issue time for block \( N \) is denoted as \( T_{blkReq,issue}(N) \). As time passes, movie blocks are consumed one by one by the client. Each time a block is consumed, a disk read request is issued to fetch the next movie block from disk.

**Definition 4.2.** \( D_{consume}(S(i,j)) \) denotes the consumption duration of the movie data from block \( i \) to \( j \) where \( i \leq j \). If \( i = j \), it is the consumption time of a single block. In case of \( i > j \), \( D_{consume}(S(i,j)) = 0 \).

Let \( T_{blkReq,finish}(i) \) denote the time instant when the disk finished servicing the reading request for movie block \( i \). We can show that \( T_{blkReq,finish}(i) \) must satisfy the following timing requirements for playback streams.

\(^1\)We use true deadline here to avoid confusion with the concept of virtual deadline introduced later in this paper.
**Lemma 4.3.** With an N-buffering scheme, a media server can satisfy every block reading request for a continuous playback stream if and only if \( \forall i \geq N \), such that \( T_{\text{blkReq\_finish}}^R(i) \leq T_{\text{movie\_Start}} + D_{\text{consume}}(S(1, i - 1)) \).

If we denote the true deadline and the issue time for a block \( i \) read request as \( T_{\text{blkReq\_deadline}}^R(i) \) and \( T_{\text{blkReq\_issue}}^R(i) \) respectively, based on Lemma 4.3, we can deduce that

**Theorem 4.4.** With an N-buffering scheme, for a playback stream, to ensure a hiccup-free playback, \( \forall i \geq N \) the block \( i \) read request issue time \( T_{\text{blkReq\_issue}}^R(i) \) and its true deadline \( T_{\text{blkReq\_deadline}}^R(i) \), must satisfy the following three requirements:

\[
\begin{align*}
\text{R1: } & T_{\text{blkReq\_issue}}^R(i) \geq T_{\text{movie\_Start}} + D_{\text{consume}}(S(1, i - N)) \\
\text{R2: } & T_{\text{blkReq\_deadline}}^R(i) \geq T_{\text{blkReq\_issue}}^R(i) \\
\text{R3: } & T_{\text{blkReq\_deadline}}^R(i) = T_{\text{movie\_Start}} + D_{\text{consume}}(S(1, i - 1))
\end{align*}
\]

**Figure 4.** Timing information for a recording stream.

**Timing Analysis for Recording Streams** We depict the timing information of a disk access for a recording stream in Figure 4. At time \( T_{\text{record\_Start}} \), a recording device begins to generate the recording stream which continuously fills the server buffer.

**Definition 4.5.** \( D_{\text{fill}}(S(i, j)) \) denotes the buffer filling duration of the recording stream data from block \( i \) to \( j \) where \( i \leq j \). If \( i = j \), it is the filling time of a single block. In case of \( i > j \), \( D_{\text{fill}}(S(i, j)) = 0 \).

Let \( T_{\text{blkReq\_finish}}^W(i) \) denote the time instant when the disk finished servicing the writing request for recording stream block \( i \). To guarantee that all the data blocks can be written to disk timely, \( T_{\text{blkReq\_finish}}^W(i) \) must satisfy the following timing requirements.

**Lemma 4.6.** With an N-buffering scheme, a media server can satisfy every block writing request for a continuous recording stream if and only if \( \forall i \geq 1 \), such that \( T_{\text{blkReq\_finish}}^W(i) \leq T_{\text{record\_Start}} + D_{\text{fill}}(S(1, N + i - 1)) \).

Let \( T_{\text{blkReq\_deadline}}^W(i) \) denote the true deadline for a block \( i \) write request. Based on Lemma 4.6, we can deduce that the issue time and true deadline for the recording requests for a recording stream must satisfy the following

**Theorem 4.7.** With an N-buffering scheme, for a recording stream, to ensure no loss recording, the write request issue time \( T_{\text{blkReq\_issue}}^W(i) \) and its true deadline \( T_{\text{blkReq\_deadline}}^W(i) \) for block \( i \), must satisfy the following three requirements:

\[
\begin{align*}
\text{R1: } & T_{\text{record\_Start}} + D_{\text{fill}}(S(1, i)) \leq T_{\text{blkReq\_issue}}^W(i) \\
\text{R2: } & T_{\text{blkReq\_deadline}}^W(i) \geq T_{\text{blkReq\_issue}}^W(i) \\
\text{R3: } & T_{\text{blkReq\_deadline}}^W(i) = T_{\text{record\_Start}} + D_{\text{fill}}(S(1, i + N - 1))
\end{align*}
\]

**4.2. Valid Deadline Setting Policies**

**Important Concepts** We define the two very important concepts of Virtual Deadline and Deadline Setting Policy below.

**Definition 4.8.** The Virtual Deadline of a disk I/O request is the deadline that is used by the deadline driven disk scheduling algorithm to choose which request should be served from the request waiting queue in the system.

**Definition 4.9.** A Deadline Setting Policy (DSP) is an algorithm that is used to compute the issue time and virtual deadline for every disk requests in a playback or recording stream.
It is intuitively clear that the deadline setting policy in a recording system should meet the timing requirements of continuous playback and recording streams, which we have discussed in the previous section. Thus, we define the concepts of a Valid Playback Deadline Setting Policy (VP-DSP) and a Valid Recording Deadline Setting Policy (VR-DSP) as follows:

**Definition 4.10.** With an $N$-buffering scheme, any playback DSP is called a Valid Playback DSP (VP-DSP) if the computed read request issue time and associated virtual deadline for each disk movie block satisfy the following three requirements:

- **R1:** $T_{\text{BlkReq,issue}}^R(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N))$
- **R2:** $T_{\text{BlkReq,deadline}}^R(i) \geq T_{\text{BlkReq,issue}}^R(i)$
- **R3:** $T_{\text{BlkReq,deadline}}^R(i) \leq T_{\text{BlkReq,deadline}}^R(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1))$

**Definition 4.11.** With an $N$-buffering scheme, any recording DSP is called a Valid Recording DSP (VR-DSP) if the computed write request issue time and associated virtual deadline for each stream block satisfy the following three requirements:

- **R1:** $T_{\text{BlkReq,issue}}^W(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))$
- **R2:** $T_{\text{BlkReq,deadline}}^W(i) \geq T_{\text{BlkReq,issue}}^W(i)$
- **R3:** $T_{\text{BlkReq,deadline}}^W(i) \leq T_{\text{BlkReq,deadline}}^W(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i + N - 1))$

Note definitions above are deduced from Theorem 4.4 and Theorem 4.7, respectively. Since we have adopted work-conserving scheduling design policy, in both VP-DSP and VR-DSP, we set the block request issue times to their earliest possible time.

**Several Valid Deadline Setting Policies** Theorem 4.4 and 4.7 provide a useful guideline to the design of a valid DSP since we adopt the work-conserving scheduling design principle. For a playback or recording stream, when the movie start time $T_{\text{movieStart}}$ and recording start time $T_{\text{recordStart}}$ are determined, the disk I/O requests’ issue times can be determined. Thus, the key design space for a DSP is the determination of the virtual deadline for each disk I/O. Based on how to compute the virtual deadlines, we propose three sets of valid DSP: (1) true deadline setting policy (T-DSP), (2) dynamic deadline setting policy (D-DSP), and (3) fair deadline setting policy (F-DSP).

**T-DSP** sets the virtual deadline of each request to their true deadline.

**Definition 4.12.** With an $N$-buffering scheme, a Read Fixed Distance playback DSP (RFD-DSP) computes the disk read request issue time $T_{\text{BlkReq,issue}}^R(i)$ and its corresponding virtual deadline $T_{\text{BlkReq,deadline}}^R(i)$ for each movie block $i$ as

$$
T_{\text{BlkReq,issue}}^R(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N)),
$$

$$
T_{\text{BlkReq,deadline}}^R(i) = T_{\text{BlkReq,issue}}^R(i) + D_{\text{consume}}(S(i, i - N + i - 1))
$$

**Definition 4.13.** With an $N$-buffering scheme, a Write Fixed Distance recording DSP (WFD-DSP) computes the disk write request issue time $T_{\text{BlkReq,issue}}^W(i)$ and its corresponding virtual deadline $T_{\text{BlkReq,deadline}}^W(i)$ for each stream block $i$ as

$$
T_{\text{BlkReq,issue}}^W(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))
$$

$$
T_{\text{BlkReq,deadline}}^W(i) = T_{\text{BlkReq,issue}}^W(i) + D_{\text{fill}}(S(i + 1, i + N - 1))
$$
**D-DSP** sets the virtual deadline of each request according to the available memory resources. Let \(N^r_{\text{available}}(t)\) denote the number of fetched movie blocks for a playback stream at time \(t\) during a playback session.

**Definition 4.14.** With an \(N\)-buffering scheme, a Read Dynamic Adjustment playback DSP (RDA-DSP) computes the disk read request issue time \(T^R_{\text{blkReq,issue}}(i)\) and its corresponding virtual deadline \(T^R_{\text{blkReq,deadline}}(i)\) for block \(i\) as

\[
T^R_{\text{blkReq,issue}}(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N))
\]

\[
T^R_{\text{blkReq,deadline}}(i) = T^R_{\text{blkReq,issue}}(i) + D_{\text{consume}}(S(i - N^r_{\text{available}}(T^R_{\text{blkReq,issue}}(i)), i - 1))
\]

Let \(N^w_{\text{available}}(t)\) denote the number of empty buffers allocated for a recording stream at time \(t\).

**Definition 4.15.** With an \(N\)-buffering scheme, a Write Dynamic Adjustment recording DSP (WDA-DSP) computes the disk write request issue time \(T^W_{\text{blkReq,issue}}(i)\) and its corresponding virtual deadline \(T^W_{\text{blkReq,deadline}}(i)\) for block \(i\) as

\[
T^W_{\text{blkReq,issue}}(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))
\]

\[
T^W_{\text{blkReq,deadline}}(i) = T^W_{\text{blkReq,issue}}(i) + D_{\text{fill}}(S(i + 1, i + 1 + N^w_{\text{available}}(T^W_{\text{blkReq,issue}}(i))))
\]

**F-DSP** sets the virtual deadline of each request based on the consumption or fill duration of 2 buffers of data.

**Definition 4.16.** With an \(N\)-buffering scheme, a Read Fair playback DSP (RF-DSP) computes the disk read request issue time \(T^R_{\text{blkReq,issue}}(i)\) and its corresponding virtual deadline \(T^R_{\text{blkReq,deadline}}(i)\) for each movie block \(i\) as

\[
T^R_{\text{blkReq,issue}}(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N))
\]

\[
T^R_{\text{blkReq,deadline}}(i) = T^R_{\text{blkReq,issue}}(i) + D_{\text{consume}}(S(i - 1, i - 1))
\]

**Definition 4.17.** With an \(N\)-buffering scheme, a Write Fair recording DSP (WF-DSP) computes the disk write request issue time \(T^W_{\text{blkReq,issue}}(i)\) and its corresponding virtual deadline \(T^W_{\text{blkReq,deadline}}(i)\) for each stream block \(i\) as

\[
T^W_{\text{blkReq,issue}}(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))
\]

\[
T^W_{\text{blkReq,deadline}}(i) = T^W_{\text{blkReq,issue}}(i) + D_{\text{fill}}(S(i + 1, i + 1))
\]

### 4.2.1. Verification of the Validity of the Proposed DSP

**Proposition 4.1:** With \(N\)-buffering scheme, RFD-DSP policy is a valid playback deadline setting policy (VP-DSP).

**Proposition 4.2:** With \(N\)-buffering scheme, WFD-DSP policy is a valid recording deadline setting policy (VW-DSP).

**Proposition 4.3:** With \(N\)-buffering scheme, RDA-DSP policy is a valid playback deadline setting policy (VW-DSP).

**Proposition 4.4:** With \(N\)-buffering scheme, WDA-DSP policy is a valid recording deadline setting policy (VW-DSP).

**Proposition 4.5:** With \(N\)-buffering scheme, RF-DSP policy is a valid playback deadline setting policy (VW-DSP).

**Proposition 4.6:** With \(N\)-buffering scheme, WF-DSP policy is a valid recording deadline setting policy (VW-DSP).

We have included all the proofs in Appendix A.

### 4.3. Measuring Fairness in Deadline Setting Policies

The issue of fairness is raised frequently in the evaluation of queuing disciplines. Therefore, in the HYDRA system, fairness issue on DSP is a naturally coming up due to the existence of disk I/O requests queue. Furthermore, since the HYDRA system provides statistical service guarantees, the system might temporarily experience overloading, which may cause some of the disk I/O requests to miss their deadline. In such an environment, a fair DSP is necessary, because (1) it can ensure that those clients that have been allocated with similar memory and disk bandwidth resources can receive similar QoS, in terms of the missed deadline probability, and (2) it can also provide better QoS to a specific stream by allocating more system resources to it.
**Fairness Measure for Deadline Setting Policies** There have much research on "flow-fairness" in networking area, where users are associated with streams of packets and the server (typically a router) aims to provide the proper throughput to the various flows while maintaining some fairness to the sources of the flows. A popular measure of fairness in that context is the relative fairness bound, which measures the "fairness of throughput of streams". However, in a streaming environment, only measuring "fairness of throughput of streams" is not enough because the timing requirement imposed by multimedia applications. Inspired by an earlier paper, which investigates the fairness measure of individual job treatment, where the effect of seniority (measured in the time already spent waiting) is also considered, we define two measures to compare the fairness of deadline setting policy in HYDRA: (1) a throughput measure to evaluate the fairness in disk I/O throughput, and (2) a system time measure to evaluate the fairness on timing requirements. In our simulation experiments, we found that all the proposed valid DSP could ensure the fairness in the disk throughput allocation. Therefore, we only focus on the second measure in this paper.

**Definition 4.18.** The System Time Measure (STM) captures the fairness in satisfying the timing requirement by measuring the distribution of $T_{\text{system}}$ (system time) of a request of a targeted stream.

In HYDRA, A closely matched $T_{\text{system}}$ distribution from any two streams implies that the DSP ensures a better fairness in satisfying the timing requirements.

---

**5. QUEUEING SYSTEM MODEL**

![Figure 5. Queueing system structure of our system.](image)

In this section, we construct a queueing system model that can compute the probability that a request missed its deadline for each playback stream, and the request blocking probability for recording streams. Figure 5 shows the general queueing system structure of our recording system. Note that in the figure we logically show each stream with a different queue, whereas in real system there could be only one waiting queue.

**5.1. Characterization of Queueing Model**

We assume that the disk I/O requests arrive according to a Poisson distribution. We prove that this assumption is valid through a hypothesis test of the measured samples from various streaming workloads generated from different real-movie traces in Section 6. We assume that when a disk I/O request is inserted into the system waiting queue, it will wait until it gets serviced.

We construct a realistic service time model by considering a detailed disk I/O model, which incorporates the following: (a) Modeling for multi-zoned disks. Fig. 6b illustrates that the disk transfer rates of current generation drives is platter location dependent. The outermost zone provides up to 30% more bandwidth than the innermost one. (b) Modeling of the variable seek time (Fig. 6a) and variable rotational latency that is naturally part of every data block read and write operation. (c) Modeling of the random data placement. In our model, we assume a work-conserving service discipline with uninterruptible service. The disk I/O requests will be served one by one according to the earliest deadline first (EDF) scheduling policy. In our model, we consider a single server (disk) at this point.
Figure 6. Important disk parameters (a Seagate Cheetah X15 disk drive) that must be considered in the model.

Figure 7. The conceptionsal queueing model of the recording system. Reading requests’ queue is modeled as $M/G/1/\infty$/$Deadline$ queueing system, writing requests’ queue is modeled as a $M/G/1/K/Deadline$ queueing system.

5.2. Analyzing the Queueing Model

To study the queueing system shown in Figure 5, the queueing system can be conceptually converted into the queueing model shown in Figure 7. There are two ways to model the system:

M1 From the figure, it is natural to consider one waiting queue in the queueing system. Then the whole system can be modelled by a $M/G/1/\infty$/$Deadline$ queueing system.

M2 Because of the buffer sharing scheme (Sec. 3) and bandwidth sharing policy$^1$ of our design, the queueing system can also be treated as two logical queues: a reading request queue ($RQueue$) and a writing request queue ($WQueue$) as shown in Figure 7. We model the $RQueue$ and $WQueue$ as a $M/G/1/\infty$/$Deadline$ and $M/G/1/K/Deadline$ queueing system, respectively.

We choose the general service time distribution because the disk service time is not exponentially distributed. Note in model M2, we choose a limited waiting room to model $WQueue$, but we use an unlimited waiting to model $RQueue$.

5.2.1. $M/G/1/\infty$/$Deadline$ Model for $RQueue$

For a playback stream, a disk reading request is driven by the consumption of movie data. When the system is heavily loaded, a disk may not be able to retrieve data blocks in time, which could result in a fairly long reading request waiting queue. Thus, from a read request point of view, there is unlimited waiting space in the $RQueue$. This is the essential reason why we model the $RQueue$ as a $M/G/1/\infty$/$Deadline$ queueing system. When a system is overloaded, some requests may miss their deadline. Therefore, we are interested to compute the probability of missing a request deadline for a playback stream, denoted as $P_{read,missed}(i)$, where $i$ is an index for all the playback streams.

$^1$Since HYDRA employs a random data placement scheme, the workload will be naturally balanced across multiple disks. Therefore, it is fairly easily to extend our model to a multi-server environment. But extension to multi-server is not in the scope of this paper.
5.2.2. M/G/1/K/Deadline Model for WQueue

For a recording stream, data is continuously generated by a recording device and is accumulated in a server write buffer. When the server buffer becomes full, a new disk write request is generated. When the system is overloaded, the disk may not be able to service write requests in time, and thus WQueue may build up. Since each write request occupies a server buffer, the buffers allocated for writing may be used up. Thus, from a writing request point of view, WQueue has limited waiting room. That is why we model the WQueue as a M/G/1/K/Deadline queuing system.

Recall that all the recording streams share a write buffer with size \( M_Y \). When new data is arriving, if all the \( M_Y \) are used, the server has two choices: (Method 1: Blocking) throw away new data; or (Method 2: Overwriting) overwrite the data that has not been written to disk. With both approaches, some data might be lost. In HYDRA, we adopt the Blocking approach for the following reasons: (1) compared with the Overwriting approach, the data lost in the Blocking approach is the more recent. Since the client buffer usually caches some of the most recent data that was sent out to the recording server, the data lost in the Blocking approach has a higher probability to be found in client buffer, which could potentially be resent to the server some time later. (2) From the implementation point of view, Blocking is easier to implement, because with the Overwriting approach, the server still needs to decide which buffer to overwrite. Thus, with WQueue, we are interested in finding the blocking probability of a new writing request, denoted as \( P_B \).

5.3. Notation of Laplace Transform and Probability Generating Function

**Definition 5.1.** The Probability Generating Function (PGF) of a nonnegative discrete random variable \( X \) is denoted as \( \mathcal{G}_X(z) \), where the probability mass function (PMF) of \( X \) is defined as

\[ p_X(n) = P(X = n) \]  

where \( n = 0, 1, 2, ... \). In addition, the \( \mathcal{G}_X(z) \) can be computed as

\[ \mathcal{G}_X(z) = E(z^X) = \sum_{n=0}^{\infty} p_X(n)z^n \]  

**Definition 5.2.** The Laplace Transform of the probability density function (pdf) of a nonnegative random variable \( X \) is denoted as \( \mathcal{L}_X(s) \). If we denote the pdf of \( X \) as \( f_X(x) \), the \( \mathcal{L}_X(s) \) can be computed as

\[ \mathcal{L}_X(s) = E(e^{-sX}) = \int_{x=0}^{\infty} e^{-sx}f_X(x)dx \]

5.4. Deriving Disk I/O Service Time Distribution

The service time \( T_{service} \) of a disk I/O request is composed of three components: seek time \( T_{diskSeek} \), rotational latency \( T_{diskRot} \), and transfer time \( T_{diskTransfer} \).

\[ T_{service} = T_{diskSeek} + T_{diskRot} + T_{diskTransfer} \]

We would like to derive the Laplace transform of pdf of \( T_{service} \), denoted as \( \mathcal{L}_{T_{service}}(s) \). Since \( T_{diskSeek}, T_{diskRot} \), and \( T_{diskTransfer} \) are independent, using the property of the Laplace transform, based on Equation 11, we can obtain

\[ \mathcal{L}_{T_{service}}(s) = \mathcal{L}_{T_{diskSeek}}(s) \times \mathcal{L}_{T_{diskRot}}(s) \times \mathcal{L}_{T_{diskTransfer}}(s) \]

5.4.1. Deriving \( \mathcal{L}_{T_{diskRot}}(s) \): Laplace transform of pdf of \( T_{diskRot} \)

Since HYDRA system adopted random data placement policy, the rotational latency \( T_{diskRot} \) follows uniform distribution with pdf \( f_{diskRot}(x) = \frac{1}{h} \) where \( x \in [1, h] \), and \( h \) denotes the maximum rotational latency of the disk drive. \( h \) can be obtained from the specification of disk manufacturer. For example, \( h = 0.004 \) second for a Seagate Cheetah X15 disk. By Definition 5.2, we can compute \( \mathcal{L}_{T_{diskRot}}(s) \) as

\[ \mathcal{L}_{T_{diskRot}}(s) = \int_{x=0}^{\infty} e^{-sx}f_{diskRot}(x)dx = \frac{1-e^{-sh}}{sh} \]
5.4.2. Deriving $L_{diskTransfer}(s)$: Laplace transform of pdf of $T_{diskTransfer}$

Most magnetic disk drives feature variable transfer rates due to a technique called zone-bit recording (ZBR), which increases the amount of data being stored on a track as a function of its distance from the disk spindle. We model the variable zone read transfer rates with random variable $R_{Dv}$. Let $L$ denote the starting location of each disk access, and $L$ can be quantified using the percentage value of the total disk capacity, i.e., $L \in [0, 100]$. From the disk transfer rate profile (see Fig. 6b), the relationship between $R_{Dv}$ and $L$ is modeled as

$$R_{Dv}(j) = \begin{cases} v_1 & \text{if } 0 \leq L \leq k_1 \ (L \in \text{Zone 1}) \\ \vdots & \vdots \\ v_w & \text{if } k_{w-1} < L \leq k_w \ (L \in \text{Zone w}) \end{cases}$$

(14)

where $w$ is the number of zones, and $v_i$ and $k_i$ model the multi-zone characteristics, where $i \in [1, w]$, $v_1 > \cdots > v_w$, $0 < k_1 < \cdots < k_w = 100$, and $[0, k_1], [k_1, k_2], \ldots, [k_{i-1}, k_i], \ldots, [k_{w-1}, k_w]$ represent zones $1, 2, \ldots, i, \ldots, w$, respectively. These $w$, $v_i$, and $k_i$ are termed disk transfer rate modeling parameters. Because of the random data placement, $L$ is uniformly distributed with pdf $f_L(l) = \frac{1}{100} (l \in [0, 100])$, let $B$ denote the size of a data block, which will be read or written to disk for each I/O request.

$$T_{diskTransfer} = \frac{B}{(\alpha + (1 - \alpha)B)R_{Dv}}$$

(15)

where $\alpha$ is the percentage of reading load in the system, and $\beta$ models the difference between the disk read/write bandwidth (See paper$^1$ for detailed derivation of Equation 15). Consequently, with the introduction of Dirac delta functions,$^{12}$ we can derive the pdf $f_{T_{diskTransfer}}(x)$ as shown in Eq. 16.

$$f_{T_{diskTransfer}}(x) = \sum_{i=1}^{w} \frac{k_i - k_{i-1}}{100} \delta(x - \frac{B}{(\alpha + (1 - \alpha)B)v_i})$$

(16)

Thus, by Definition 5.2, we obtain $L_{T_{diskTransfer}}(s)$ as

$$L_{T_{diskTransfer}}(s) = \int_{0}^{\infty} e^{-sx} f_{T_{diskTransfer}}(x) dx = \sum_{i=1}^{w} \frac{k_i - k_{i-1}}{100} \times e^{-s \frac{B}{(\alpha + (1 - \alpha)B)v_i}}$$

(17)

5.4.3. Deriving $L_{diskSeek}(s)$: Laplace transform of pdf of $T_{diskSeek}$

Let $S$ denote the percentage value (between 0 and 100) of the total disk storage capacity. We express the relationship among $T_{diskSeek}(j), S$ through disk profiling and modeling$^{13}$ as

$$T_{diskSeek} = \begin{cases} a_1 + b_1 \sqrt{S} & \text{if } 0 \leq S \leq r \\ a_2 + b_2 S & \text{if } r < S \leq 100 \end{cases}$$

(18)

where $a_1, b_1, a_2, b_2, r$ are the disk seek modeling parameters. Because of the random data placement, both $S$ follow uniform distributions with pdf $f_S(x) = \frac{1}{100} (x \in [0, 100])$. Figure 6a shows the seek time profile of a Seagate Cheetah X15 disk, which has the following parameters: $a_1 = 0.001, b_1 = 0.0006, a_2 = 0.0021, b_2 = 0.00005, r = 5$. Thus, by Definition 5.2, we compute $L_{T_{diskSeek}}(s)$ as

$$L_{T_{diskSeek}}(s) = E\left(e^{-sT_{diskSeek}}\right) = \int_{y=-\infty}^{\infty} E\left[e^{-sT_{diskSeek}}|S = y\right] f_S(y)$$

$$= \int_{y=0}^{\infty} e^{-s(a_1 + b_1 \sqrt{y})} \frac{1}{100} dy + \int_{y=r}^{\infty} e^{-s(a_2 + b_2 y)} \frac{1}{100} dy$$

(19)

$$= \frac{1}{100} \times \left[ -e^{-s(a_1 + b_1 \sqrt{r})} + \frac{50s^2 b_1^2 e^{(a_1 + b_1 \sqrt{r})r}}{100} \times (-sb_2) \right]$$

We have now obtained all three components to evaluate $L_{T_{service}}(s)$ as given by Equation 12.
5.5. Deriving the Distribution of the Number of Requests in RQueue

Let \( N_{\text{ReqSys},R} \) denote the number of requests in the RQueue. Since \( N_{\text{ReqSys},R} \) is a discrete random variable, to find its PMF, we can derive the probability generating function of \( N_{\text{ReqSys},R} \), denoted as \( G_{N_{\text{ReqSys},R}}(z) \).

According to the derivation in Kleinrock’s classical queueing system book,\(^5\) we have observed one of the basic fact in a \( M/G/1 \) queueing system:

**Remark 5.1:** the Pollaczek-Khinchin (P-K) transform equation that compute the distribution of number of requests in the system can apply to \( M/G/1 \) model with any queueing discipline.\( ^1 \)

Since RQueue is modeled as a \( M/G/1/\infty/\text{Deadline} \) queuing system, based on Fact 5.1, we can apply Pollaczek-Khinchin (P-K) transform equation in\(^5\) to derive the \( G_{N_{\text{ReqSys},R}}(z) \)

\[
G_{N_{\text{ReqSys},R}}(z) = \mathcal{L}_{T_{\text{service}}}(\lambda_R - \lambda_R z) \frac{(1 - \rho_R)(1 - z)}{\mathcal{L}_{T_{\text{service}}}(\lambda_R - \lambda_R z) - z} \tag{20}
\]

where \( \rho_R \) denotes the system utilization of RQueue and can be obtained by

\[
\rho_R = \frac{\lambda_R}{\mu_R} \tag{21}
\]

Let \( \mu_{T_{\text{service, only}}} \) denote the average disk request service rate when all the requests are read requests, \( \mu_R \) can be computed as

\[
\mu_R = \frac{\alpha}{\mu_{T_{\text{service, only}}}} \tag{22}
\]

Note that \( \mu_{T_{\text{service, only}}} \) can be obtained by using the distribution function derived in Section 5.4, where \( \alpha = 1 \), and for Seagate Cheetah X15 \( \beta = 0.6934 \).

5.6. Deriving \( P_B \): the Probability of Blocking for Recording Streams

Since the WQueue is modeled as a \( M/G/1/K/\text{Deadline} \) system, by applying the formulas in the classical M/G/1/K model,\(^1\) we can obtain the probability of a new disk request being blocked due to the write buffer limitation \( M_W \). We summarize the main steps as follows. Let \( \pi_i^{(\infty)} \) denote the probability of \( N_{\text{ReqSys},W} \) is \( i \) in the corresponding \( M/G/1/\infty/\text{Deadline} \) WQueue system. Follow the similar method as previous section, we can obtain \( \pi_i^{(\infty)} \).

\[
\pi_i^{(\infty)} = p_{N_{\text{ReqSys},W}}(i) = P(N_{\text{ReqSys},W} = i) \tag{23}
\]

Next, obtain the PGF of \( N_{\text{ReqSys},W} \) for the corresponding \( M/G/1/\infty/\text{Deadline} \) WQueue system, denoted as \( G_{N_{\text{ReqSys},W}}^{\infty}(z) \).

\[
G_{N_{\text{ReqSys},W}}^{\infty}(z) = \mathcal{L}_{T_{\text{service}}}(\lambda_W - \lambda_W z) \frac{(1 - \rho_W)(1 - z)}{\mathcal{L}_{T_{\text{service}}}(\lambda_W - \lambda_W z) - z} \tag{24}
\]

where \( \rho_W \) can be computed as

\[
\rho_W = \frac{\lambda_W}{\mu_W} \tag{25}
\]

Let \( \mu_{T_{\text{service, only}}} \) denote the average disk request service rate when all the requests are write requests, \( \mu_W \) can be computed as

\[
\mu_W = \frac{1 - \alpha}{\mu_{T_{\text{service, only}}}} \tag{26}
\]

Note that \( \mu_{T_{\text{service, only}}} \) can be obtained by using the distribution function derived in Section 5.4, where \( \alpha = 0 \), and for Seagate Cheetah X15 \( \beta = 0.6934 \). After computing \( G_{N_{\text{ReqSys},W}}^{\infty}(z) \), we can obtain \( \pi_i^{(\infty)} \). Next, by applying the formulas in classical M/G/1/K model,\(^1\) we obtain PMF of \( N_{\text{ReqSys},W} \) as follows:

\[
p_{N_{\text{ReqSys},W}}(i) = \frac{\pi_i^{(\infty)}}{1 - q_k \rho_W} \quad \text{if } i = 0, 1, 2, ..., K - 1 \tag{27}
\]

13
where \( q_k \) denotes the tail probability of the infinite queue and can be computed as

\[
q_k = \sum_{i=K}^{\infty} \pi_i(\infty) = 1 - \sum_{i=0}^{K-1} \pi_i(\infty)
\]  

(29)

Now, we have obtained one of the most important parameter, \( P_B \), the probability of a new disk request being blocked due to the write buffer limitation \( M_W \).

\[
P_B = P_{\text{new,svr}} (M_W)
\]  

(30)

5.7. Deriving the Probability that a Disk Read Request Will Miss its Deadline

Next, we derive the probability that a reading request missed its deadline in \( R Queue \), denoted as \( P_{\text{read,missed}}(i) \), \( i \) is an index of all the playback streams. Note for convenience purpose, we assume that all the requests we discussed in this section will be requests from playback stream \( i \) unless explicitly stated otherwise. Therefore, we will omit the index \( i \) in the discussion.

5.7.1. General Guideline

![Diagram](image)

Figure 8. lifetime diagram of a disk I/O request.

Figure 8 depicts the lifetime of a disk I/O request, which starts from the instant \( T_{\text{req,issue}} \) when the request is initiated until the point \( T_{\text{req,svr,Fin}} \) when the server finishes servicing the request. Note that we define the request system time \( T_{\text{system}} \) as \( T_{\text{wait}} + T_{\text{service}} \). One of the most important concepts we would like to introduce is the request lead time.

**Definition 5.3.** The Lead Time of a request \( T_{\text{leadtime}} \) at time instant \( T_0 \) is defined as the remaining time since \( T_0 \) until the deadline of a disk I/O request. That is \( T_{\text{leadtime}} = T_{\text{deadline}} - T_0 \). Lead Time can also be called Relative Deadline.

Notice that the Lead Time of a request will decrease linearly as time passes. For example, as shown in Figure 8, \( T_{\text{leadtime}} \) denotes the request lead time at time instant \( T_0 \). Let \( T_{\text{req,issue}}, T_{\text{deadline}} \) and \( T_{\text{system}} \) denote the issue time, system time, and deadline for a request from playback stream \( i \), respectively, as shown in Figure 8.

\[
P_{\text{read,missed}}(i) = P[\text{a read request from stream } i \text{ missed its deadline}] \\
= P[T_{\text{req,issue}} + T_{\text{system}} > T_{\text{deadline}}]
\]  

(31)

Let \( T_{\text{leadtime,arrival}} \) denote the \( T_{\text{leadtime}} \) for a request at the time instant when the request just arrived into the system. Thus, as indicated in Fig. 8, we have \( T_{\text{deadline}} = T_{\text{req,issue}} + T_{\text{leadtime,arrival}} \). Thus, Equation 31 could be simplified as

\[
P_{\text{read,missed}}(i) = P[T_{\text{system}} > T_{\text{leadtime,arrival}}]
\]  

(32)

Random variable \( T_{\text{leadtime,arrival}} \) is determined by the inherent bandwidth characteristics for playback stream \( i \) and deadline setting policy. Next, we derive the distribution of \( T_{\text{system}} \) for a request of a playback stream \( i \) in \( R Queue \) by utilizing the real time queuing theory (RTQ).

14
5.7.2. Summary of Main Results from RTQT

RTQT incorporates deadline scheduling as the queuing discipline. It can determine the distribution for lead-time profile for the customers waiting in the queue. Let \( \lambda \) denote the mean customer arrival rate. \( Q \) is the length of the waiting queue at time \( T_0 \). Let \( G(x) \) denote the cumulative distribution function (CDF) of the lead time of arriving customers. For a \( M/G/1 \) queue with EDF queuing discipline, at time \( T_0 \), given \( \lambda, Q, G(x) \), RTQT computes the lead-time profile of all the requests in the queue as a probability distribution with a probability density function (pdf) given by:

\[
f_X(x) = \begin{cases} 
\frac{\lambda}{Q} (1 - G(x)) & \text{if } L(Q) \leq x \leq \infty \\
0 & \text{if } x < L(Q)
\end{cases}
\]  

where \( L(Q) \) is the departing customer’s (the next customer to be serviced) lead time at \( T_0 \), which can be determined as

\[
\frac{\lambda}{Q} \int_{L(Q)}^{\infty} (1 - G(x)) \, dx = 1
\]

5.7.3. Applying RTQT to Our Problem

With Equation 33, we obtain the CDF of the lead time profile as

\[
F_X(t) = \int_{-\infty}^{t} f_X(x) \, dx = \begin{cases} 
\frac{\lambda}{Q} \int_{L(Q)}^{T_{leadtime,arrival} - t + T_{req,issue}} (1 - G(x)) \, dx & \text{if } L(Q) \leq t \leq \infty \\
0 & \text{if } x < L(Q)
\end{cases}
\]

Recall that \( T_{leadtime,arrival} \) denotes the \( T_{leadtime} \) for a request (denoted as \( R EQ_A \)) at the time instant when the request just arrived into the system. Note that this CDF can come from any playback stream. Let \( N_{req, before} \) denote the number of requests lined up before \( R EQ_A \) at any time instant \( t \). Suppose at time \( t \), the waiting queue length is \( Q_t \). As shown from Figure 8, the lead time for \( R EQ_A \) at time \( t \) can be computed as \( T_{leadtime,arrival} - t + T_{req,issue} \). Thus, by RTQT, using Equation 35, we get

\[
N_{req, before}(t) = Q_t \times \frac{\lambda}{Q_t} \int_{L(Q_t)}^{T_{leadtime,arrival} - t + T_{req,issue}} (1 - G(x)) \, dx
\]

Let \( Y \) denote a duration after the arrival of \( R EQ_A \), then using Equation 36 we can compute the number of request before \( R EQ_A \) at time instant \( T_{req,issue} + Y \). Now, consider \( Y = T_{wait} \), then, the time instant \( T_{req,issue} + Y = T_{req,Startup} \), the service start time of \( R EQ_A \). Because at \( T_{req,Startup} \), there are no requests lined up before \( R EQ_A \). Thus, with Equation 36, we obtain

\[
N_{req, before}(T_{req,issue} + T_{wait}) = \frac{\lambda}{Q_t} \int_{L(Q_t)}^{T_{leadtime,arrival} - T_{wait}} (1 - G(x)) \, dx = 0
\]

In a real system, usually \( \lambda > 0 \), and \( G(x) \) can not always be 1, therefore, we can deduce that

\[
T_{leadtime,arrival} - T_{wait} = L(Q) \iff T_{wait} = T_{leadtime,arrival} - L(Q)
\]

Again, we want to emphasize that \( T_{leadtime,arrival} \) denotes the initial lead time of a request from any playback stream. Because \( T_{system} = T_{wait} + T_{service} \), thus we can obtain

\[
T_{system} = T_{leadtime,arrival} - L(Q) + T_{service}
\]

Substituting \( T_{system} \) in Equation 32, we obtain

\[
\text{read,missed}(i) = P \left[ T_{overall} \geq T_{leadtime,arrival} - L(Q) + T_{service} > T_{stream,i} \right]
\]

Note in Equation 32, the \( T_{leadtime,arrival} \) denotes the lead time of a read request from a specific playback stream \( i \). While in Equation 39, the \( T_{system} \) denotes the lead time of a read request from any playback stream. To avoid confusion, we distinguish them with \( T_{leadtime,arrival} \) and \( T_{overall} \) respectively. Recall that we already obtained the distribution of \( N_{ReqSysR} \) in Section 5.5, which is the distribution \( Q \). And from RTQT, we can compute the distribution of \( L(Q) \). In addition, in Section 5.4, we obtained the distribution \( T_{service} \). Assume we know the detailed bandwidth characteristics of all the playback streams in the system, then we can obtain the distributions of \( T_{leadtime,arrival} \) and \( T_{overall} \). Therefore, as indicated by Equation 40, we have obtained all the components to compute \( \text{read,missed}(i) \).
6. PERFORMANCE EVALUATION

6.1. Experimental Setup

We have implemented our proposed deadline setting policies (DSPs) and dynamic buffer allocation scheme in a simulation system shown in Fig. 9. The WorkLoad Generator produces client playback or recording requests based on a Poisson process with the mean inter-arrival time $\frac{1}{\lambda} = 5$ seconds. The movie blocks are randomly placed onto a disk and block requests are scheduled based on the EDF scheduling policy. Deadlines for block requests are computed based on (a) DSP, (b) buffer allocation, and (c) movie trace profile from Movie Trace Library. The block requests are forwarded to the Disk by the Disk Access Scheduler at the set times. The Measure & Report module monitors the disk system performance and generates the result output. The WorkLoad Generator has several configurable parameters: the mean inter-arrival time $\frac{1}{\lambda}$, the number of retrieval streams $n_r$, and the number of recording streams $n_w$. Our disk system simulates a Seagate Cheetah X15 (Model ST336752LC) disk drive. Disk block size $B_{disk}$ is set to 1.0 MB. Table 2 summarizes the parameters used in the experiments and analysis.

![Figure 9. Experimental system setup.](image)

**Table 2.** Parameters used in the experiments and analysis.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test movie “Twister”</td>
<td>MPEG-2 video, AC-3 audio</td>
</tr>
<tr>
<td>Average bandwidth</td>
<td>698,594 Bytes/sec</td>
</tr>
<tr>
<td>Length</td>
<td>50 minutes</td>
</tr>
<tr>
<td>Throughput std. dev.</td>
<td>140/456.8</td>
</tr>
<tr>
<td>Test movie “Saving Private Ryan”</td>
<td>MPEG-2 video, AC-3 audio</td>
</tr>
<tr>
<td>Average bandwidth</td>
<td>757,298 Bytes/sec</td>
</tr>
<tr>
<td>Length</td>
<td>50 minutes</td>
</tr>
<tr>
<td>Throughput std. dev.</td>
<td>169/743.6</td>
</tr>
<tr>
<td>Test movie “Charlie’s Angels”</td>
<td>MPEG-1 video, Stereo audio</td>
</tr>
<tr>
<td>Average bandwidth</td>
<td>180/129 Bytes/sec</td>
</tr>
<tr>
<td>Length</td>
<td>70 minutes</td>
</tr>
<tr>
<td>Throughput std. dev.</td>
<td>500/441</td>
</tr>
</tbody>
</table>

| Disk Model                          | Seagate Cheetah X15 (Model ST336752LC)             |
| Mean inter-arrival time $\frac{1}{\lambda}$ of streaming request | 5 seconds                                          |
| Disk block size $B_{disk}$           | 1.0 MB                                              |
| Deadline Setting Policy (DSP)       | T-DSP, D-DSP and F-DSP                              |

6.2. Fairness Comparison between T-DSP, D-DSP and F-DSP

Figure 10 shows the experimental results with the three proposed deadline setting policies: a true deadline setting policy (T-DSP), a dynamic deadline setting policy (D-DSP), and a fair deadline setting policy (F-DSP). The comparison is based on the System Time Measure (STM) defined in Section 4.3. In the simulation, 58 concurrent playback streams are running, each stream is generated from the trace of the DVD movie “Twister”. Half of the 58 streams are each allocated with 2 buffers, and the other 29 streams each is allocated with 30 buffers. Each figure shows two curves of the relative frequency histogram of the system time $T_{system}$. One of the curve is from a stream with 2 buffers, the other is from a stream with 30 buffers. Figure 10(a), (b) and (c) show the results with T-DSP, D-DSP and F-DSP, respectively. With T-DSP, the $T_{system}$ for the stream with 2 buffers is significantly less than the stream with 30 buffers as shown in Figure 10(a) in terms of both the mean value and the standard deviation. This implies that T-DSP is unfair for these two streams. On the other hand, with
Figure 10. Comparison of the distribution of the $T_{system}$ system time of requests from two playback streams, one having 2 buffers, the other having 30 buffers. Note that the experiments are conducted with 58 playback streams, which are generated from the trace of DVD movie “Twister”. Half of the streams (29 streams) each has with 2 buffers, and the other half 29 streams each has 30 buffers. The same experiments are conducted with three proposed DSPs: a true deadline setting policy (T-DSP), a dynamic deadline setting policy (D-DSP), and a fair deadline setting policy (F-DSP).

Figure 11. Comparison of the distribution of the $T_{system}$ system time of requests from two playback streams, one having 2 buffers, the other having 10 buffers. Note that the experiments are conducted with 220 playback streams, which are generated from the trace of VCD movie “Charlie’s Angels”. Half of the streams (110) each is employed with 2 buffers, and the other half 110 each has 10 buffers. The same experiments are conducted with the three proposed DSPs: a T-DSP, a D-DSP, and a F-DSP.

Figure 12. Comparison of the distribution of the $T_{system}$ system time of requests from two playback streams, one having 2 buffers, the other having 30 buffers. Note that the experiments are conducted with 54 playback streams, which are generated from the trace of DVD movie “Saving Private Ryan”. Half of the streams (27 streams) each has 2 buffers, and the other half 27 streams each has 30 buffers. The same experiments are conducted with the three proposed DSPs: a T-DSP, a D-DSP, and a F-DSP.

F-DSP, the $T_{system}$ distribution curves match reasonably well for those two streams. The results of D-DSP lie between the results of T-DSP and F-DSP. Note that the two streams used in the comparison are randomly selected from those two sets of streams with different buffer sizes. Therefore, we can conclude that, among all three proposed DSPs, the F-DSP is the fairest DSP, whereas the T-DSP is the most unfair one.

Similar experimental results are also collected under different streaming workloads generated from other real-movie traces. For example, Figure 11 and Figure 12 show the results with movie trace from VCD “Charlie’s Angels” and DVD “Saving Private Ryan”, which confirm that the F-DSP is the fairest DSP among all three proposed DSPs.
Next, we evaluate the impact of T-DSP, D-DSP and F-DSP on the probability of a request missing its deadline for playback streams with different buffer sizes. In each experiment, 58 DVD “Twister” concurrent playback streams are running in the system. 29 of them (those with client index between 29 and 57) are always allocated with 2 buffers. The other 29 streams (those with client index between 0 and 28) always have the same number of buffers, and we increase their buffer size by one in each experiment from 2 to 30. Note that we use the varying buffer size of the second group as an index of buffer allocation schemes for all the experiments, termed Buffer Allocation Index shown in Figure 13. In each experiment, we measure the probability of a request missed deadline for each of the 58 streams. The same set of experiments are repeated for all three proposed DSPs. Fig. 13(a) shows that with T-DSP all the streams experience a similar request missed deadline probability. However, with F-DSP, the results (shown in Fig. 13c) are more desirable because the stream with more buffers may experience a lower requests missed deadline probability consistently. The results of D-DSP lie between the two extreme cases, that is T-DSP and F-DSP. The main reason causing the difference is that F-DSP can ensure fairness among all the streams in terms of request system time distribution. Since both T-DSP and D-DSP can not provide fair service, the benefit of more buffer allocation is compromised by the unfair treatment in the scheduling of requests. Furthermore, we observe that by using F-DSP different QoS can be provided to different streams with different memory allocation. Similar results are also confirmed from a series of similar experiments under the streaming workload generated from the trace of DVD movie “Saving Private Ryan” as shown in Figure 14.

6.3. Evaluation of Buffer Sharing Scheme for Recording Streams

Recall that we have proposed that a write buffer pool will be shared among all the recording streams in HYDRA, and we call this scheme Shared Buffer Scheme (SBS). In this section, we evaluate the effectiveness of our proposed shared buffer scheme by comparing it with a Partitioned Buffer Scheme (PBS), where the server buffer capacity is equally partitioned for each recording stream. In HYDRA, when the write buffers are used up, the new arriving media data will be thrown away (i.e., blocked). Therefore, the comparison metric that we have adopted is the blocking probability for recording streams.

We conduct recording experiments with different system configurations: (1) different buffer allocation schemes (both
Figure 15. Evaluation of buffer sharing schemes for recording streams, where the blocking probability of arriving new data changes as a function of server memory capacity with different buffer allocation schemes, i.e., both SBS and PBS.

SBS and PBS, (2) different streaming workloads (different media type and different number of concurrent streams), and (3) different server memory capacity. Note that in all the recording experiments, we adopted the Fair Deadline Setting Policy (F-DSP) since F-DSP ensures the fairness treatment among all the streams as shown in the previous section.

Figure 15 shows some of the experimental results where the blocking probability of new arriving data changes as a function of the server memory capacity with different buffer allocation schemes. Figure 15(a) and (b) show the results collected when DVD “Saving Private Ryan” movie trace was used to generate the recording streams. In both figures, the blocking probability decreases gradually to zero as the server buffer capacity increases for both PBS and SBS. Notice that SBS consistently results in a lower blocking probability than PBS throughout all the memory configurations. In general, the benefit of SBS increases as memory capacity increases. For example, as shown in Figure 15(a), with 38 “Saving Private Ryan” recording streams, the difference of blocking probability between PBS and SBS increases from 0.056% with 76 MB memory to approximately 0.128% with 190 MB memory. Similar trends were also obtained from recording experiments generated from DVD “Twister” movie trace as shown in Figure 15(c) and (d), where the number of concurrent streams were 41 and 43 respectively.

Based on the results of our recording experiments, we make the following observations:

i In the recording experiments, the blocking probability of SBS are always less than that of PBS. This is intuitively easy to understand since a PBS only exploits the statistical multiplexing benefit due to bandwidth variability for each stream respectively while SBS can take advantage of the bandwidth variability not only within each stream themselves but also across different streams.

ii With memory capacity increases, the advantage of SBS over PBS also increases gradually. This is because with more memory SBS can better exploit the statistical gains across different streams.

ii With more memory, the blocking probability decreases gradually as expected. When memory capacity is more than certain threshold, the blocking probability becomes zero for both SBS and PBS. Note the SBS’s threshold is always
smaller than that of PBS. The threshold values are actually the minimum buffer required to ensure all the recording streams are correctly archived in the system. And the threshold value changes as system workload changes. The threshold value might be compute by using the probability model developed in Section 5.

In summary, compared to Partitioned Buffer Scheme, our proposed Shared Buffer Scheme improves the HYDRA system performance by taking advantage of the statistical multiplexing of different I/O bandwidth requirements across different recording streams.

6.4. Choosing $N$ for Different Media Type with N-Buffering Scheme

In this section, we conducted playback experiments to verify that playback streams of different media types might achieve the same level of QoS requirement with different memory buffer requirements. Note that one Seagate Cheetah X15 disk is used in these experiments, and each playback stream is allocated with the same amount of buffers. Moreover, we adopted the $F$-DSP to compute the virtual deadline used for disk scheduling.

![Figure 16](image1.png)

**Figure 16.** Comparison of the probability of a request missed its deadline from two playback streams as a function of the number of clients with different number of buffers for DVD movie “Saving Private Ryan” and VCD “Charlie’s Angels”.

Figure 16 shows the experimental results that the probability of a request missed deadline changes as a function of both the number of concurrent streams and the number of buffers allocated. Similar experiments were conducted for workload generated from three different movie types: DVD movie “Saving Private Ryan”, DVD movie “Twister”, and VCD movie “Charlie’s Angels”, where the corresponding results are presented as Figure 16(a), (b) and (c), respectively. Figure 16(a) and (b) shows a quite similar trend: the probability of request missed deadline increase as the number of concurrent streams increase, and it decreases as the number of allocated buffer increases.

![Figure 17](image2.png)

**Figure 17.** Comparison of the disk system utilization, percentage of server busy time, and the probability of a request missed its deadline from two playback streams as a function of the number of clients with different number of buffers for DVD movie “Saving Private Ryan” and VCD “Charlie’s Angels”.

Note that the upper limiting number of streams for each movie type shown in the figure 16 were selected carefully through experiments such as system utilization is less than 1 (This is necessary to ensure that the queueing system is
stable). Figure 17 shows the playback experimental results of the disk system utilization, percentage of the server busy time, and the request missed deadline probability as a function of the number of concurrent streams for three different movie types. As expected all three performance metrics increase as the number of streams increases. The disk system utilization is the same as fraction of server busy time when the system utilization is less than 1. The system utilization increases linearly as the number of streams increases. The limiting numbers are 54, 59, and 222 for DVD movie "Saving Private Ryan", "Twister" and VCD movie "Charlie's Angeles" respectively as shown in Figure 17.

As suggested from the Figure 16, 27 is the minimum number buffers required for DVD movie "Saving Private Ryan" such that no requests missed their deadline. Similarly, 24 buffers are required for DVD movie "Twister". While for VCD movie "Charlie's Angeles", 2 buffers are enough. These experiments confirm that playback streams of different media types might achieve the same level of QoS requirements with different memory buffer requirements, which motivates our design of dynamic buffer allocation scheme for playback streams.

6.5. Hypothesis Test of Disk I/O Request Arrival Process

To verify our model assumption that the interarrival time of the disk I/O requests follows an exponential distribution, we conduct a set of $\chi^2$ Chi-Square Goodness-of-Fit Test\textsuperscript{17} on a set of measured samples collected under various streaming workloads generated from different real-movie traces. Here are our Hypothesis:

$H_0$: The measured disk I/O requests interarrival time follows exponential distribution.

$H_a$: The measured disk I/O requests interarrival time do not follow exponential distribution.

$\chi^2$ Goodness-Of-Fit Test Statistics: For the Chi-Square goodness-of-fit test, the measured data samples are divided into $k$ bins and the test statistics is defined as

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$ (41)

where $O_i$ is the observed frequency of measured samples for bin $i$. $E_i$ is the expected frequency of samples for bin $i$, which can be computed as

$$E_i = N(e^{-\lambda Y_1} - e^{-\lambda Y_2})$$ (42)

where $N$ denotes the total number of measured data samples, $Y_0$ and $Y_1$ denote the upper and lower limit for each bin, $\lambda$ is the mean arrival rate of disk I/O requests, which can be estimated by computing the average arrival rate from the measured samples. Basically, smaller values of the $\chi^2$ test statistics show the observed and expected distributions are more similar (in a better fit).

Figure 18 shows the observed measured samples and the computed expected values based on Equation 42 in different experimental setups. In Figure 18(a), data samples are collected when 200 concurrent DVD “Saving Private Ryan” streams are running in the system. In Figure 18(b), data samples are collected when 300 concurrent VCD “Charlie’s Angeles” streams are running in the system. The observed samples in Figure 18(c) are collected from 200 DVD “Twister” streams. Finally, Figure 18(d) presents the result of a mixed streams experiments where 200 streams are running simultaneously (1/3 of them are DVD “Twister”, another 1/3 are DVD “Saving Private Ryan” streams, the rest of them are VCD "Charlie’s Angel’s" streams). In all these experiments, the computed expected frequencies fit the measured frequencies of data samples surprisingly well. Actually, they all passed the Chi-Square Goodness of fit test with a reasonable significance level, e.g., 0.95. To be complete, we also conducted experiments to find out the changing trends of $\chi^2$ test statistics value for three different movie types, DVD “Twister”, DVD “Saving Private Ryan”, and “Charlie’s Angeles”. The experiments results are shown in Figure 19. All three curves indicate the similar trend that as the number of concurrent streams increases, the observed inter-arrival time distribution becomes more and more closely matched with exponential distribution.
6.6. Verification of \textit{RTQT} in HYDRA

Since nobody has applied \textit{RTQT} to the streaming environment where the VBR streams can generate an arbitrary deadline distribution for arrival requests, we have conducted extensive simulations to verify the applicability of \textit{RTQT} in our HYDRA system.

We first conduct experiments to verify \textit{RTQT} under different streaming workloads generated different real-movie traces. Figure 20 compares the \textit{RTQT} predicted lead time (\textit{T}_{leadtime}) profile with the measured average lead time snapshots when the request waiting queue length is 100. Figure 20(a), (b) and (c) show the results when workload are generated from different movie traces DVD “Saving Private Ryan”, DVD “Twister” and VCD “Charlie’s Angeles”, respectively. Each figure contains 5 curves, the last two curves labelled with “predicted gX CDF” and “avg LT snapshot” are the \textit{RTQT} prediction and measurement results (the other 3 curves are some intermediate computation results of \textit{RTQT}). The average lead time values are collected from a large number of snapshots. 149 snapshots for experiment with “Saving Private Ryan” streams, 99 snapshots for test with “Twister” streams, and an amazing 1879 snapshots for test with VCD streams. Note all the streams are playback streams. In all three figures, \textit{RTQT} predictions closely match the simulation results.

We also conduct experiments to verify \textit{RTQT} with different waiting queue lengths, such as 25, 50 and 75 percentiles of the wait queue length distribution. Figure 21 compares the \textit{RTQT} predicted results with the measured results collected with recording and playback experiments under the streaming workload generated from the trace of DVD movie “Twister”. Note Figure 21 (a) and (c) shows the wait queue length distributions under recording and playback experiments respectively. Figure 21 (b) and (d) shows the corresponding \textit{RTQT} predicted results and measured results. Again, in all experiments
Figure 19. The general trend of $\chi^2$ test statistics with different number of concurrent streams for several different movie types, DVD movie “Saving Private Ryan”, DVD movie “Twister”, and VCD movie “Charlie’s Angels”.

Figure 20. The fraction of disk I/O requests less than lead time $T_{\text{leadTime}}$ for different media types, DVD movie “Saving Private Ryan” (54 streams), DVD movie “Twister” (55 streams), VCD movie “Charlie’s Angeles” (222 streams) when waiting queue length is 100.

and at different requests’ wait queue lengths, $RTQT$ closely predicted the lead-time profile of the requests in the waiting queue. In summary, we argue that $RTQT$ is applicable to our timing analysis in HYDRA.

7. CONCLUSIONS

We presented an effective resource management framework that is composed of the design of dynamic memory allocation strategy, which applies to different playback and recording streams, and a novel deadline setting policy ($F-DSP$) that can be applied consistently to both the playback and recording streams, satisfying the timing requirements of continuous media streams, and also ensuring fairness among different streams. Moreover, we constructed a probability model based on the classic $M/G/1$ queuing model and the recently developed Real Time Queuing Theory ($RTQT$) that can be used to evaluate the performance trade-off of different buffer allocation and deadline setting policies. Our next step is to apply the developed model to find the optimal memory resource allocation, and to provide admission control and capacity planning.

REFERENCES

Figure 21. Waiting queue length distribution and the fraction of disk I/O requests less than lead time $T_{leadTime}$ during a recording only experiment with 42 concurrent streams and a playback only experiment with 59 concurrent streams. The streaming load is generated by DVD movie “Twister”. Snapshots are taken at different waiting queue length. Comparing the predicted results with the average values of snapshots during the tests. Fig. 21a-c show the waiting queue length distribution.


APPENDIX A. PROOF OF LEMMAS AND THEOREMS

A.1. Proof of Lemma 4.3

**Lemma 4.3**  With an N-buffering scheme, a media server can satisfy every block reading request for a continuous playback stream if and only if \( \forall i \geq N \), such that \( T^{R}_{\text{blkReq\_finish}}(i) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \).

**Proof:** We prove by verifying the statement is true in both directions.

**Step 1: proof of \( \Rightarrow \).** In other words, we need to prove if all the block reading requests for a playback stream are satisfied timely, then every block are available (finished fetching from disk into buffer) before their corresponding start display time. (Case I: \( \forall i < N \) with N-buffering scheme, because the first \( N - 1 \) blocks (i.e., block 1, 2, ..., \( N - 1 \)) are prefetched before move start time \( T_{\text{movieStart}} \). Thus we have

\[
T^{R}_{\text{blkReq\_finish}}(i) \leq T_{\text{movieStart}}
\]

According to Definition 4.2, \( \forall i \geq 1 \), \( D_{\text{consume}}(S(1, i - 1)) \geq 0 \). Thus, by Equation 43, \( \forall i < N \) we obtain \( T^{R}_{\text{blkReq\_finish}}(i) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \). (Case II: \( \forall i \geq N \) recall that we denote a request finish time for block \( i \) as \( T^{R}_{\text{blkReq\_finish}}(i) \). We also know that for block \( i \), its start showing time is \( T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \). Thus, we obtain \( T^{R}_{\text{blkReq\_finish}}(i) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \). By Case I and II, we have proved Step 1.

**Step 2: proof of \( \Leftarrow \).** We need to prove that \( \forall i \), if \( T^{R}_{\text{blkReq\_finish}}(i) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \), then statement 1 or statement 2 is true.

**statement 1:** movie can be played without hiccup, in other words, at any given time instant \( t \), there always exist the appropriate movie content available for display.

**statement 2:** \( \forall i \geq 1 \), at time \( t_{i} = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \), the next block, i.e., block \( i \), is always available for display. $^{5}$

(Case I: \( \forall i < N \) with N-buffering scheme, because the first \( N - 1 \) blocks (i.e., block 1, 2, ..., \( N - 1 \)) are prefetched before move start time \( T_{\text{movieStart}} \). Therefore, they will be available at their corresponding display time \( t_{i} = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \), where \( i \leq N \).

(Case II: \( \forall i \geq N \) proof by induction).

**Substep 1:** when \( i = N \), then \( t_{N} = T_{\text{movieStart}} + D_{\text{consume}}(S(1, N - 1)) \). Therefore, at \( t_{N} \), block 1, 2, ..., \( N - 1 \) have been consumed. As since \( T^{R}_{\text{blkReq\_finish}}(N) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, N - 1)) = t_{N} \), thus, block \( N \) must be available for display.

**Substep 2:** induction step. Suppose if \( \forall i, i \leq k, k \geq N \), statement 2 is true. That is, if \( \forall t_{i} \leq t_{k}, k \geq N \), block \( i \) is available for display at time \( t_{i} \), then, block \( k \) is available for display at time \( t_{k} \). Considering time \( t_{k+1} = T_{\text{movieStart}} + D_{\text{consume}}(S(1, k)) \), since \( T^{R}_{\text{blkReq\_finish}}(k + 1) \leq T_{\text{movieStart}} + D_{\text{consume}}(S(1, k)) = t_{k+1} \), therefore, block \( k + 1 \) is available at \( t_{k+1} \), by Substep 1 and 2, we have proved Case II of Step 2.

Thus, with Case I and II of Step 2, we have proved Step 2.$^{\spadesuit}$

A.2. Proof of Theorem 4.4

**Theorem 4.4.** With an N-buffering scheme, for a playback stream, to ensure a hiccup-free playback, \( \forall i \geq N \), the block \( i \) read request issue time \( T^{R}_{\text{blkReq\_issue}}(i) \) and its true deadline \( T^{R}_{\text{blkReq\_deadline}}(i) \), must satisfy the following three requirements:

1. **R1:** \( T^{R}_{\text{blkReq\_issue}}(i) \geq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N)) \)
2. **R2:** \( T^{R}_{\text{blkReq\_deadline}}(i) \geq T^{R}_{\text{blkReq\_issue}}(i) \)
3. **R3:** \( T^{R}_{\text{blkReq\_deadline}}(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \)

**Proof:**

$^{\spadesuit}$This is because a block is the atomic disk access unit in disk I/O, thus if at time \( t_{i} = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \), block \( i \) is available, then, during the next the next block consumption period \( D_{\text{consume}}(S(i, i)) \), there can not be any hiccup at all.
Step 1: proof of R1.

**Case I:** \( i = N \) With N-buffering scheme, a playback stream is allocated with \( N \) buffers, and the first \( N - 1 \) blocks are prefetched before \( T_{\text{movieStart}} \). Therefore, \( T_{\text{blkReqIssue}}(N) \), the earliest time to issue read request for block \( N \) will be \( T_{\text{movieStart}} \). By Definition 4.2, \( D_{\text{consume}}(S(1,0)) = 0 \). Thus, if \( i = N \), \( T_{\text{blkReqIssue}}(N) \geq T_{\text{movieStart}} + D_{\text{consume}}(S(1, N - N)) \).

**Case II:** \( i > N \) Due to the buffer limitation, \( T_{\text{blkReqIssue}}(i) \), the issue time of disk read request for block \( i \), can not be earlier than the time when \( i - N \) movie blocks are consumed. That is, \( T_{\text{blkReqIssue}}(i) \geq T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N)) \).

Step 2: proof of R2. This requirement is intuitively obvious because the true deadline of a disk block read request must be greater than its issue time.

Step 3: proof of R3. To ensure a hicc-up free display, the true deadline of a disk block read request must be earlier than the latest (maximum) request finish time computed in Lemma 4.3.

By Steps 1, 2, and 3, we have proved the Theorem. \( \blacksquare \)

A.3. Proof of Lemma 4.6

**Lemma 4.6** With an N-buffering scheme, a media server can satisfy every block writing request for a continuous recording stream if and only if \( \forall i \geq 1 \), such that \( T_{\text{blkReqFinish}}^W(i) \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, N + i - 1)) \).

**Proof:** We prove by verifying the statement is true in both directions.

Step 1: proof of \( \Rightarrow \).
If all the writing requests from a continuous recording stream are satisfied timely, that implies that every blocks are correctly written to disks. With N-buffering scheme, suppose \( N \) buffers are allocated to a recording stream. Thus, for a block write request, it is written to a disk before the continuous incoming data stream fill up all the available empty buffer space. Because the maximum empty buffer space for the disk writing request is \( N - 1 \) buffers.* Therefore, every writing request from the recording stream must be written to disks before the incoming stream filled up the \( N - 1 \) buffers. Thus, we obtain

\[
T_{\text{blkReqFinish}}^W(i) \leq T_{\text{blkReqIssue}}^W(i + 1) + D_{\text{fill}}(S(1, N + i - 1)) \tag{44}
\]

Since a block write request can not be issued before all the data for the block are generated. Therefore, we obtain:

\[
T_{\text{blkReqIssue}}^W(i) \geq T_{\text{recordStart}} + D_{\text{fill}}(S(1, i)) \tag{45}
\]

By Equations 44 and 45, we obtain

\[
T_{\text{blkReqFinish}}^W(i) \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, N + i - 1)) \tag{46}
\]

*One buffer is occupied by the data block that needs to be written to disk.
Step 2: proof of $\Leftarrow$. We only need to prove $\forall i$, if $T^{W}_{\text{blkReq,finish}}(i) \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, N + i - 1))$, statement 1 or statement 2 is true.

**statement 1:** At any time $t \geq T_{\text{recordStart}}$, there exist empty buffer to hold incoming data stream.

**statement 2:** When a buffer being filled up, i.e., $t = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))$, there exist empty buffer to hold incoming data stream.

Note that statement 1 and statement 2 are equivalent, because when $t \geq T_{\text{recordStart}}$ but $t \neq T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))$, the recording stream having filling up a writing buffer. In other words, there are empty buffer space to hold incoming data.

To prove statement 2, we only need to prove statement 3.

**statement 3:** $\forall i, i \geq 1$, $t = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i))$, there exist a empty buffer available to be filled.

We prove statement 3 by induction on $i$.

**Substep 1 (Initial Step):** $i \leq N$ (Case 1: $i \leq N - 1$) with N-buffering scheme, i.e., the recording stream is allocated with $N$ buffers, then if $i \leq N - 1$, i.e., $t \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, N - 1))$, the $N$ buffers have not been used up. Thus, on these time instants, there exist a empty buffer available to be filled.

(Case 2: $i = N$) since for block 1,

$$ T^{W}_{\text{blkReq,finish}}(1) \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, N)) \tag{47} $$

Thus, when $i = N$, $t = T_{\text{recordStart}} + D_{\text{fill}}(S(1, N))$, block 1 has been written to disk. Accordingly, the buffer that is occupied by block 1 has been become empty. In other words, there exist a empty buffer available.

**Substep 2 (Induction Step):** $i > N$ Suppose $\forall i$, $i \leq k$ and $k \leq N$, statement 3 is true. Thus, when $i = k$, $t = T_{\text{recordStart}} + D_{\text{fill}}(S(1, k))$ denoted as $t_{k}$, and blocks $1, 2, ..., k - N + 1$ are written to disk. Consider $i = k + 1$, by assumption we have that

$$ T^{W}_{\text{blkReq,finish}}(k - N + 2) \leq T_{\text{recordStart}} + D_{\text{fill}}(S(1, k + 1)) \tag{48} $$

That is, block $k - N + 2$ is finished writing to disk at $t_{k+1}$. Accordingly, a new empty buffer become become available. Therefore, statement 3 is still hold when $i = k + 1$.

By Substep 1, 2, we have proved Step 2 is true. 

**A.4. Proof of Theorem 4.7**

**Theorem 4.7** With an N-buffering scheme, for a recording stream, to ensure no loss recording, the write request issue time $T^{W}_{\text{blkReq,issue}}(i)$ and its true deadline $T^{W}_{\text{blkReq,deadline}}(i)$ for block $i$, must satisfy the following three requirements:

**R1:** $T_{\text{recordStart}} + D_{\text{fill}}(S(1, i)) \leq T^{W}_{\text{blkReq,issue}}(i)$

**R2:** $T^{W}_{\text{blkReq,deadline}}(i) \geq T^{W}_{\text{blkReq,issue}}(i)$

**R3:** $T^{W}_{\text{blkReq,deadline}}(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i + N - 1))$

**Proof:**

**Step 1: proof of R1.** The block write request can not be issued before all the data in that block have been accumulated. Therefore, the request issue time for block $i$, $T^{W}_{\text{blkReq,issue}}(i)$ must satisfy

$$ T_{\text{recordStart}} + D_{\text{fill}}(S(1, i)) \leq T^{W}_{\text{blkReq,issue}}(i) \tag{49} $$
Step 2: proof of R2. This requirement is intuitively obvious because the true deadline of a disk block write request must be greater than its issue time.

Step 3: proof of R3. To ensure a hiccup-free display, with N-buffering scheme, the true deadline of a disk block write request must be earlier than the latest (maximum) required request finish time computed in Lemma 4.6.

By Steps 1, 2, and 3, we have proved the Theorem.

\[ \text{A.5. Proof of Proposition 4.1} \]

Proposition 4.1 With N-buffering scheme, RFD-DSP policy is a valid playback deadline setting policy (VP-DSP Proof: Proof by verify R1, R2 and R3 in the Definition 4.10.

According to the Definition 4.12, R1 and R2 are obviously true. Next, we verify R3.

By Definition 4.12, we have

\[ T^R_{\text{blkReq,issue}}(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N)) \]  

(50)

\[ T^V_{\text{blkReq,deadline}}(i) = T^R_{\text{blkReq,issue}}(i) + D_{\text{consume}}(S(i - N + 1, i - 1)) \]  

(51)

Thus, we obtain

\[ T^V_{\text{blkReq,deadline}}(i) = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - N)) + D_{\text{consume}}(S(i - N + 1, i - 1)) \]

\[ = T_{\text{movieStart}} + D_{\text{consume}}(S(1, i - 1)) \]

\[ = T^R_{\text{blkReq,deadline}}(i) \]  

by Theorem 4.4

\[ \text{A.6. Proof of Proposition 4.2} \]

Proposition 4.2 With N-buffering scheme, WFD-DSP policy is a valid recording deadline setting policy (VW-DSP)

Proof: Proof by verify R1, R2 and R3 in the Definition 4.11.

According to the Definition 4.11, R1 and R2 are obviously true. Next, we verify R3.

By Definition 4.13, we have

\[ T^W_{\text{blkReq,issue}}(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i)) \]  

(53)

\[ T^W_{\text{blkReq,deadline}}(i) = T^W_{\text{blkReq,issue}}(i) + D_{\text{fill}}(S(i + 1, i + N - 1)) \]  

(54)

Thus, we obtain

\[ T^W_{\text{blkReq,deadline}}(i) = T_{\text{recordStart}} + D_{\text{fill}}(S(1, i)) + D_{\text{fill}}(S(i + 1, i + N - 1)) \]

\[ = T_{\text{recordStart}} + D_{\text{fill}}(S(i + 1, i + N - 1)) \]

\[ = T^W_{\text{blkReq,deadline}}(i) \]  

by Theorem 4.7

\[ \text{A.7. Proof of Proposition 4.3} \]
**Proposition 4.3**  With N-buffering scheme, RDA-DSP policy is a valid playback deadline setting policy (VW-DSP).

**Proof:** The RDA-DSP is the same as RFD-DSP except the virtual deadline computation.

By Definition 4.14,

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) = T_{\text{BlkReq\_issue}}^{R}(i) + D_{\text{consume}}(S(i - N_{\text{available}}^{R}(T_{\text{BlkReq\_issue}}^{R}(i)), i - 1))
\]  

(56)

With N-buffering scheme, one of the buffer is used for displaying the movie content to client. Therefore,

\[
N_{\text{available}}^{R}(T_{\text{BlkReq\_issue}}^{R}(i)) < N - 1
\]

(57)

Thus, we obtain

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) < T_{\text{BlkReq\_issue}}^{R}(i) + D_{\text{consume}}(S(i - (N - 1)), i - 1))
\]

(58)

By Proposition 4.1, we have proved the proposition.

\[
\]

**A.8. Proof of Proposition 4.4**

**Proposition 4.4**  With N-buffering scheme, WDA-DSP policy is a valid recording deadline setting policy (VW-DSP).

**Proof:** The WDA-DSP is the same as WFD-DSP except the virtual deadline computation.

By Definition 4.15,

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) = T_{\text{BlkReq\_issue}}^{W}(i) + D_{\text{fill}}(S(i + 1, i + N_{\text{available}}^{W}(T_{\text{BlkReq\_issue}}^{W}(i))))
\]

(59)

With N-buffering scheme, one of the buffer is used for accumulating the recording stream. Therefore,

\[
N_{\text{available}}^{W}(T_{\text{BlkReq\_issue}}^{W}(i)) \leq N - 1
\]

(60)

Thus, we obtain

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) \leq T_{\text{BlkReq\_issue}}^{W}(i) + D_{\text{fill}}(S(i + 1, i + N - 1))
\]

(61)

By Proposition 4.2, we have proved the proposition.

\[
\]

**A.9. Proof of Proposition 4.5**

**Proposition 4.5**  With N-buffering scheme, RF-DSP policy is a valid playback deadline setting policy (VW-DSP).

**Proof:** The RF-DSP is the same as RFD-DSP except the virtual deadline computation.

By Definition 4.14,

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) = T_{\text{BlkReq\_issue}}^{R}(i) + D_{\text{consume}}(S(i - 1, i - 1))
\]

(62)

With N-buffering scheme, since the minimum number of buffers allocated for a stream is 2, i.e., \( N \geq 2 \). Thus, we have

\[
D_{\text{consume}}(S(i - (N - 1), i - 1)) \geq D_{\text{consume}}(S(i - 1, i - 1))
\]

(63)

Therefore, we obtain

\[
T_{\text{BlkReq\_deadline}}^{VR}(i) \leq T_{\text{BlkReq\_issue}}^{R}(i) + D_{\text{consume}}(S(i - (N - 1)), i - 1))
\]

(64)

By Proposition 4.1, we have proved the proposition.

\[
\]
A.10. Proof of Proposition 4.6

**Proposition 4.6**  With N-buffering scheme, WF-DSP policy is a valid recording deadline setting policy (VW-DSP).

**Proof:** The WF-DSP is the same as WFD-DSP except the virtual deadline computation.

By Definition 4.17,
\[ T^{VW}_{\text{blkReq,deadline}}(i) = T^{W}_{\text{blkReq,issue}}(i) + D_{\text{fill}}(S(i + 1, i + 2)) \]  \hspace{1cm} (65)

With N-buffering scheme, since the minimum number of buffers allocated for a stream is 2, i.e., \( N \geq 2 \). Thus, we have

\[ D_{\text{fill}}(S(i + 1, i + N - 1)) \geq D_{\text{fill}}(S(i + 1, i + 1)) \]  \hspace{1cm} (66)

Thus, we obtain
\[ T^{VW}_{\text{blkReq,deadline}}(i) \leq T^{W}_{\text{blkReq,issue}}(i) + D_{\text{fill}}(S(i + 1, i + N - 1)) \]  \hspace{1cm} (67)

By Proposition 4.2, we have proved the proposition.

[ ]