Lyra: A System Framework in Supporting Multimedia Applications

Paul C. H. Lee

Institute of Information Science
Academia Sinica, Taipei, TAIWAN

Abstract

Current systems, such as UNIX, lack necessary scheduling and resource management mechanisms in supporting specific timing requirements, thus multimedia applications cannot run smoothly on top of them. Aim on solving these problems, in this paper we propose a system framework, the Lyra, for supplying multimedia applications with sophisticate timing controls. Lyra enhances conventional systems in supplying firewall protections among multimedia applications, while also considering I/O interference, and aims on processing digital data on time. Empirical evaluations show that Lyra not only practically provides better QoS supports than conventional systems, but also performs better than several works that aim on similar problems.

1. Introduction

Advances in compression techniques [Mit96] have made digital data to be stored and processed more efficiently than the past decade, today that we see multimedia applications everywhere in entertainment, education, conferencing or in medical services. Unlike conventional applications, multimedia applications do not process discrete data only, but usually process continuous media (CM) data instead. CM data, such as video and audio streams, have strong temporal relationships that usually require timing control in playback and synchronization. However, we cannot find any commodity systems that support such kind of timing services.

Conventional systems cannot service the timing requirements of multimedia applications because these systems are originally designed for time-sharing applications. These so-called time-sharing applications only care their data whether to be processed correctly and efficiently, but care nothing about whether the data are processed on time. Multimedia applications, on the contrary, care more about whether the data streams are manipulated on time, but can tolerate a certain level errors in data manipulations, if these errors are not so evident to human. Though multimedia applications are so popular on commodity desktop systems, what we observed is that applications themselves need to do timing controls, but still depend on the unreliable best-effort system services. When the system is slightly overloaded, the behaviors of multimedia applications are out of controls [Lee98].

To meet the timing requirements, seemingly it is possible to apply the existing hard real time scheduling algorithms [But97, Liu73] into desktop systems [Mer93]. This kind of adoption has been experimental demonstrated to be unacceptable for servicing multimedia applications [Nie93]. This unacceptable result is caused because multimedia applications have different behaviors from those of hard real time applications. Missing deadlines is not a so serious failure for multimedia applications. In addition, they can also tolerant a certain level of data loss, thus multimedia applications can smoothly reduce their resource-consumption rates and adapt to an acceptable playback to reduce system loads [Nei97].

Another views to support multimedia applications are to make resource reservations to provide Quality of Service (QoS) guarantees for multimedia applications [Jon97, Mer94]. How fair about the resource allocations and what kind of delay-bound guarantees are usually the concerned points. Firewall protection is the basic result achieved via the proportional share reservation techniques [Jon97, Kim94]. The results in bounded-delay guarantee for resource availability are good but aren’t good enough for practically servicing multimedia applications. Multimedia applications do not really care how fair they consume the resources, but care about whether the data can be operated on time. In addition, multimedia applications require not only the processor resource. To practically meet the requirements of multimedia applications, a system should consider multiple resource scheduling at the same time [Raj98].

In this paper, the *Lyra* system is introduced to solve above problems. *Lyra* is a system framework that is designed and implemented for practically serving specific timing requirements for multimedia applications, which will be detailed illustrated in following sections. By isolating the resource management mechanisms from conventional machine core components, *Lyra* acts as a framework for defining the running model about program executions. *Lyra* enhances conventional systems in supplying firewall protections among multimedia applications, while also considering I/O interference, and aims on processing digital data on time. Empirical evaluations show that *Lyra* not only practically provides better QoS supports than conventional systems, but also performs better than several previous works that aim on similar problems.

The rest of the paper is organized as follows. In order to know what kind of services multimedia applications need, and which practical considerations should be taken into the system design, we first discuss characteristics of multimedia applications at Section 2. Section 3 introduces the programming model used in *Lyra*. Based on this model, the *Lyra* is designed and implemented. Section 4 illustrates its architecture. Follow on is the important, and maybe the most important resource scheduling algorithms used in *Lyra* in smoothly serving executions of multimedia applications. The performance evaluations are presented at Section 5 to highlight *Lyra*'s contributions. To compare with the works that aims on similar problems, a simulation environment based on *Lyra* is built for further inspections. Section 6 introduces previous related researches. At the last Section, we conclude this paper.

2. Multimedia Applications

To design a framework for running multimedia applications, we first need to know their characteristics that specially asking for system supports. Without losing the generality, we focus on applications that play digital video and audio data.

Multimedia data should be processed on time. This means playbacks for media data should bring users an acceptable quality. Slightly delays and data loss can be tolerated, but too largely variances in media playbacks will be perceived by human and cannot be accepted. To make use of this characteristic, a multimedia application can be modeled as a periodic task with constant period, but has variable processing time in each period. This is a different feature from conventional hard real time applications. Take video playbacks for example. To meet the requirements of human eyes, each video frame should play 30 frames per second. It means the period of the video player is 33.3 milliseconds, a fixed time period. However, the processing time within each period is different from period to period.

In addition, multimedia applications are data-intensive applications. The data that they consumed and manipulated are larger than conventional tasks. For example, an MPEG-1 video player usually needs about 1.5 Mega bits per second. This large data size will consume a lot of memory buffer and cause I/O subsystem to be busy in satisfying the consumption. Like the applications, I/O subsystem also needs processor time in feeding data to applications. However, the relationships among the I/O device capacity, its processor time consumption, the buffer consumption and the admission control mechanism are usually ignored in previous researches. A practical system should consider these in more detail to do more sophisticates services.

Since multimedia applications are resource-intensive, a system should provide firewall protections to avoid any resource monopolization and system service lock up. In addition, when an application is authorized to run on a system by passing the admission-control mechanism, it is better to provide executable guarantee for this application. Users will get crazy if a video clip is played smoothly at the initial time but later hanging there for any perceivable delay. Thus how to guarantee the shared resource among competing applications is also a necessity for framework design.

For most multimedia application designers, they probably know some timing or resource-consumption attributes, but it is hard for them to specify exact and complete attributes in application design or executions. For example, a MPEG-1 player designer probably knew that this player will consume about 1.5 Mega bits per second, and also knew that the player needs to play 30 frames per second, but he will never know how much processing time the player needs to run on a system. And so does the exact share that the player needs to allocate when it is executed on any possible platforms. A system framework should provide an easy-to-use model for programming, and an adaptive mechanism to change the parameters dynamically.

Multimedia applications usually will manipulate multiple streams at the same time. For example, a movie clip will need at least the audio and video playbacks at the same time. A very important requirement for this kind of services is how to do synchronization during playing multiple streams. Especially, when a system framework supports such kind of services, this service should not induce too much overhead in system performance, or introduce too much burden on application designers.

Another feature about multimedia applications is the dynamic characteristic. Unlike embedded hard real time applications, multimedia applications are not known in a prior, that a system scheduler can not make an optimal scheduling plan for them. No matter what kind the schedulability tests or resource reservations, each scheduling mechanism needs to consider the dynamic characteristic. This will cause

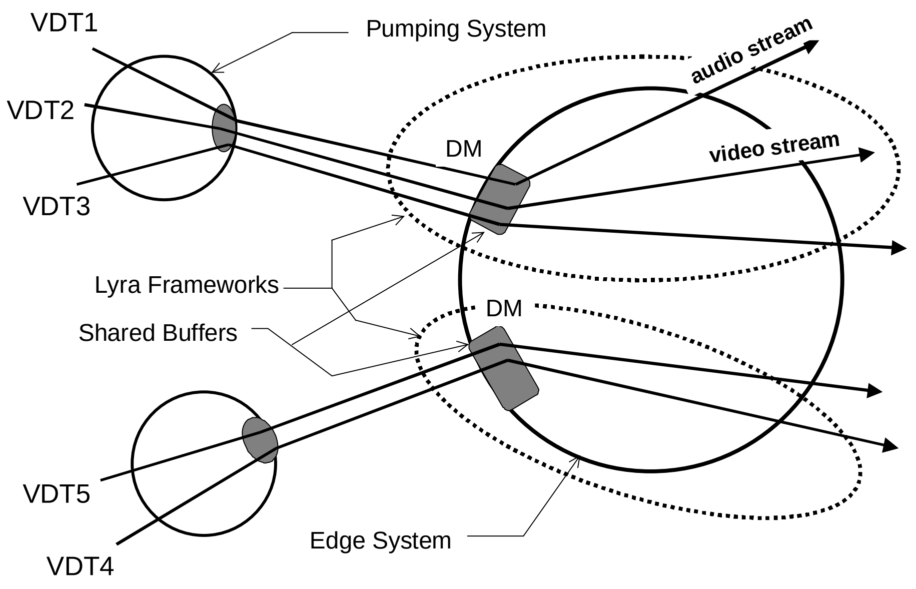


Figure 1: The Tunnel model used in *Lyra* environment

special difficulties in making any admission controls and scheduling plans, if multiple resources are taken into consideration. Fortunately, for each type of media data processing, it is found that the processing time is proportional to the data size that being processed [Bav98, Lee98]. If applying the linear regression model to analyze the relationship between processing time and data size, the coefficient of determination is close to 1. This means if we model multimedia applications as periodic tasks, their data consumption rates within each period will also be proportional to the processing time of that period. A system framework can make use of this information to add assumptions for reducing the difficulties.

3. Programming Model in *Lyra*

The Tunnel Model

Based on above observations and discussions, here we introduce the programming model used in *Lyra*, the tunnel model. As illustrated in Figure 1, *Lyra* assumes that all multimedia applications are executed at edge systems, which are the real resource collections for program executions. Each continuous media stream is pumped into the edge system from remote pumping systems trough the virtual data tunnels (VDT). Each stream will occupy one tunnel for sending data. It is assumed that each pumping system is powerful enough to pump enough data into the tunnel. Each tunnel can be physically constructed by a network link or by an I/O bus connecting the I/O devices.

On the edge systems, the hardware resources are partitioned and shared by several *Lyra* frameworks. And within each *Lyra* framework, there is a data manager (DM) that is responsible for receiving data from each tunnel and preparing the data on the shared buffers for each application. In addition, DM is also responsible for sending feedback information to the pumping systems, if the pumping systems send data too fast or too slow. The pumping system can be a video server in the remote site or a simple disk attached at the same machine *Lyra* operates. About how the DM talks with pumping system will not be further discussed in this paper. We will focus on how *Lyra* operates.

Definitions and Notations

Here we formally define the multimedia tasks, the data manager and introduce a remaining rate server (RRS) used in *Lyra*.

Definition 1: Each multimedia application is defined to be a task M_i with multiple threads $\{h_{ij} \mid j>0\}$ attached. Each thread is responsible for one data stream playback. Threads created for manipulating multimedia data are periodic When the thread h_{ij} is created, it needs to specify its release time τ_{ij} , the periodic T_{ij} , and the averaged data consumption rate γ_{ij} . The framework will assign a default processing time C_{ij} to the thread h_{ij}. Period is a constant but processing time is a variable that may change in each period.

Definition 2: A processing deadline d_{ij}^k is set to the end of each periodic T_{ij}^k, $\forall\ k$. If $\exists\ k$, ∍ $C_{ij}^k \geq T_{ij}^k$ for any $k \geq 1$, it is defined as a deadline miss.

Definition 3: Based on Definition 2, an averaged CPU share for each thread h_{ij} is s_{ij} , where $s_{ij} = C_{ij} / T_{ij}$.

Definition 4: A DM is a periodic thread. Its release time is set to 0, its maximum data production rate is R and its periodic is T_D , where T_D is set to $min\{\ T_{ij} \mid \forall\ i, j,\ if\ h_{ij}\ exists\}$, when admission control is performed. DM's processing time C_D in each periodic is defined to $C_D = \Sigma \gamma_{ij} \cdot T_D / R \cdot S_D$, when each time the admission control is performed.

Unlike the periodic thread definitions described in Definition 1, T_D is not a constant and so is the C_D. Their values are updated when admission control is performed. We will describe later why T_D and C_D are set to above values in the Section of scheduling algorithms. We also describe some assumptions in following paragraphs. We made these assumptions before we design the *Lyra* framework. These assumptions though are not 100% matching the real world cases, these assumptions are very close to practical situations.

Definition 5: Remaining Rate Sever is a virtual thread. It does not contain any executable codes. Once a thread exhausted its processor share, its remaining jobs within this period will queue to this server for possible executions. Once the RRS gets its CPU share, it will execute the queued jobs. The CPU rate assigned to this job is equal to $U = (1 - \Sigma(C_{ij} / T_{ij}))$. The virtual deadline V of RRS is defined to $v + Q / (1 - \Sigma(C_{ij} / T_{ij}))$, where v is the virtual time [Zha91] of RRS and Q is the system time quantum.

The RRS is created here for using the remaining rates to do the works that attached to it. The primary purpose is that if a thread has exhausted its CPU share, i.e., executed longer than the permitted C_{ij}^k time in period T_{ij}^k, the remaining jobs can be queued to the RRS for possible executions. If the extra jobs can be executed before the deadline d_{ij}^k , a deadline miss is reduced. Of course, if the jobs are not executed on time before the next period is issued, a deadline miss is created. RRS's goal is to do its best efforts to help multimedia applications that have exhaust their reserved rates but still remaining jobs to run. RRS will execute the remaining jobs in LCFS order because the last one may have the higher probability to be executed before its deadline.

Assumption 1: The data manager has a maximum data production rate R, and it is known to the *Lyra* system. The DM also needs processor time to service each data stream. The processor consumption rate for data manager to achieve its maximum data production rate is S_D, and is known to *Lyra* system.

Assumption 2: Each thread's real data consumption rate γ_{ij} is proportional to its real processing time C_{ij}.

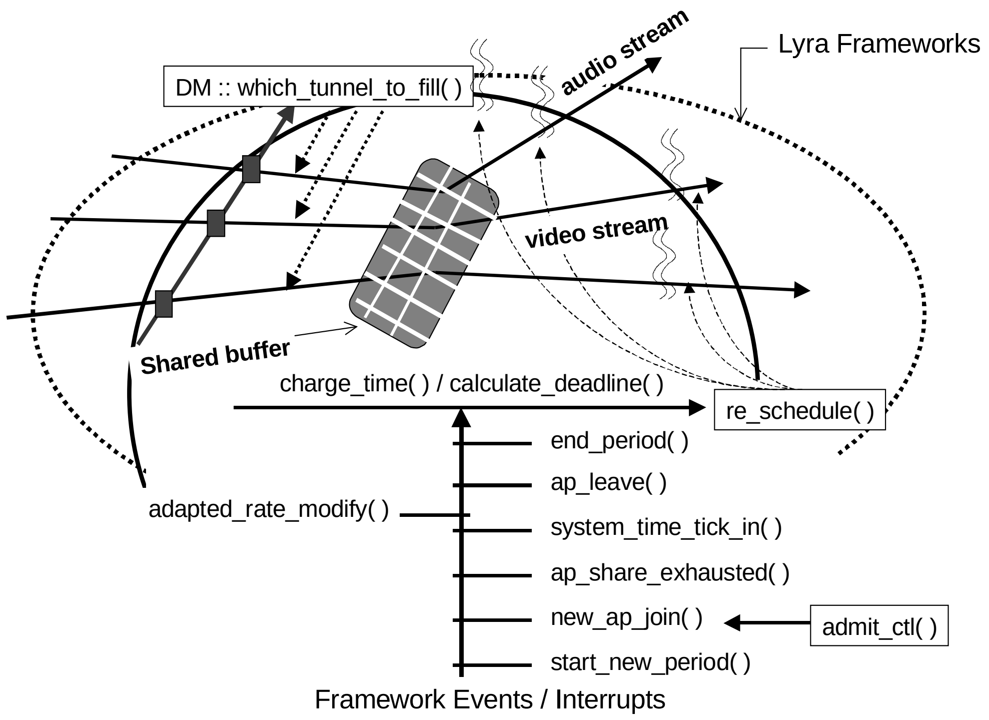


Figure 2: The *Lyra* architecture.

Since data manager is a thread running on the system, it is evident that the data manager also costs processor time to serve each data stream. The processor time it costs will be proportional to its data production rate. For example, if data manager runs as an asynchronous I/O server, it can prefetch data into the system greedily. When each time the data is ready for copying, data manager costs CPU time to move data into the shared buffer. Thus the assumptions match the real world cases.

Though applications specify their data consumption rates when they are ready to run, the real rate that applications consume in each period will not always be the same. But from the results of previous works [Bav98, Lee98], it makes sense to make the assumption 2.

4. *Lyra* Framework Architecture

Design goals

Lyra is designed under following expectation. First, it should has a robust admission control mechanism that when an application is authorized to execute on top of it, the application can run smoothly to its end.

For achieving this goal, we consider more resources at one time, than just thinking the CPU resource only.

The processing time C_{ij} of each thread is not a constant from period to period. If we use the averaged C_{ij} to do admission control, more deadline misses might be created. We expect that the deadline misses can be reduced as more as possible. Third, a thread's execution cannot influence other threads' execution, once the thread is admitted to get its share of the resources. This is the so-called firewall protection property.

The exact processing time within each periodic is not constant. Thus *Lyra* is expected to implement an adaptation mechanism to dynamic change the C_{ij} of each thread h_{ij} to a precise one. Of course, it is still under the constraint that the adaptations will not influence existing authorized applications' CPU share.

Motivations

Each thread runs periodically that they will get their CPU share s_{ij} if it does not need extra processor time. However, if it exhausts its reservation rate s_{ij}, *Lyra* still expects to help it to finish its jobs on time. So, *Lyra* creates RRS to help finishing the jobs. If finally the extra jobs can be finished on time, it is good. If not, than a deadline miss is created.

Because each thread's CPU reservation share s_{ij} is not exact as what they need, they may reserve too much or too little by setting the C_{ij} at the time the thread is created. In addition, after a longer phase, the share may also change from phase to phase. Thus an adaptation mechanism is needed.

Finally, because data manager also consumes processor rate in serving multimedia applications, the admission control mechanism needs to consider this extra rate when applications joining the system and leaving the system. In addition, whether the data consumption rate is beyond the data manager's maximum data production rate R also needs to take into consideration. And whether buffer size is enough is also important.

Based on these goals and motivations, *Lyra* is designed and implemented. The remaining problems are how the RRS to compete with all threads to help reducing the number of deadline misses, while still keep firewall protections among applications. We left this discussion at Section 5, the resource scheduling algorithms.

System Architecture

Lyra's architecture is shown in Figure 2. Since it is a framework, it means if any system that tries to support *Lyra*, this system should have the ability to export the system privilege and resource management to *Lyra*. This can be done by using extended operating systems, like *SPIN* [Ber95] or, *hipec* [Lee96], or just run *Lyra* on top of existing embedded kernels. We now implement *Lyra* on *Vega* kernel [Lee97], and plan to port it to real time Linux.

There are seven major parts defined in *Lyra*, the scheduling module, the event/interrupt module, the buffer management module, the admission control module, the adaptation module and the data manger. The whole system is controlled by the event/interrupt module, which will issue six major events to the system. Except the AP_LEAVE and AP_JOIN events, the other four events are the re-scheduling points for *Lyra* to re-calculate which thread to run next. Routines in adaptation module will be invoked to re-calculate the CPU share at the time the period is started, or ended, or when the share is exhausted. When each event happens, scheduling module is activated to select the next thread with earliest deadline to run.

5. Resource Scheduling Algorithms

The primary scheduling algorithms used in Lyra are illustrated in Figure 3 and 5. They are the virtual deadline algorithm and the admission control algorithm. The other one is the adaptation algorithm.

Virtual Deadline (VD) Algorithm

VD algorithm's purpose is to define RRS server's virtual deadline that we can use it to compete with normal thread. By calculating the CPU share of RRS and using it to calculating the virtual time of RRS, RRS is designed no to influence others. Furthermore, if virtual time is defined, we use it to get the virtual deadline of RRS.

VD algorithm is called when system clock interrupts the system. The primary purposes of VD algorithm are to check whether there is a deadline miss after this time quantum, to charge time to the running thread and update RRS's virtual deadline. In line 1 of Figure 3, if VD finds that this is an end of a period,

```
Algorithm VD( )
1.  if (deadline miss) handle it
2.  if (CPU is idle) $v$ = current time
3.  else
4.      if ( normal thread running) {
5.          charge time consumption $\varepsilon$ to the thread.
6.          if (this thread exhausts its share)
7.              Append the remaining jobs to the RRS
8.              if (RRS idle)
9.                  activate it.
10.                 $v = v + ( \varepsilon - C_{ij}) / U$
11.     } else if ( it is RRS running )
12.                 $v = v + \varepsilon / U$
13. }
14. $V = v + Q / U$.
15. Pick a thread with an earliest deadline to run
```

Figure 3: Virtual deadline algorithm

it means now is a deadline miss. VD handles that deadline miss by discarding any remaining jobs in this period, no matter this job is processed by the thread itself or by the RRS.

As illustrated in line 6, when VD finds that the running thread exhausts its share but still remains jobs to run in this period, VD appends the jobs to the RRS. After competing with other threads, RRS might be scheduled to run to finish the remaining jobs before the deadline.

Special tricks are used in VD at line 2, 10, 12, 14. At line 2, virtual time of RRS is reset to the current time to avoid the RRS saving a lot of credits. Without this reset, RRS will have a very small virtual time just because it is not executed for a long time. Once there are jobs attached to the RRS, RRS will soon be scheduled and run for a long time to increase its virtual time. This will cause other threads to miss their deadlines. Line 10 needs to be added because the system can not be preempted at any continuous time, but only in system time tick boundary. So there exist possibilities that a thread consumes more

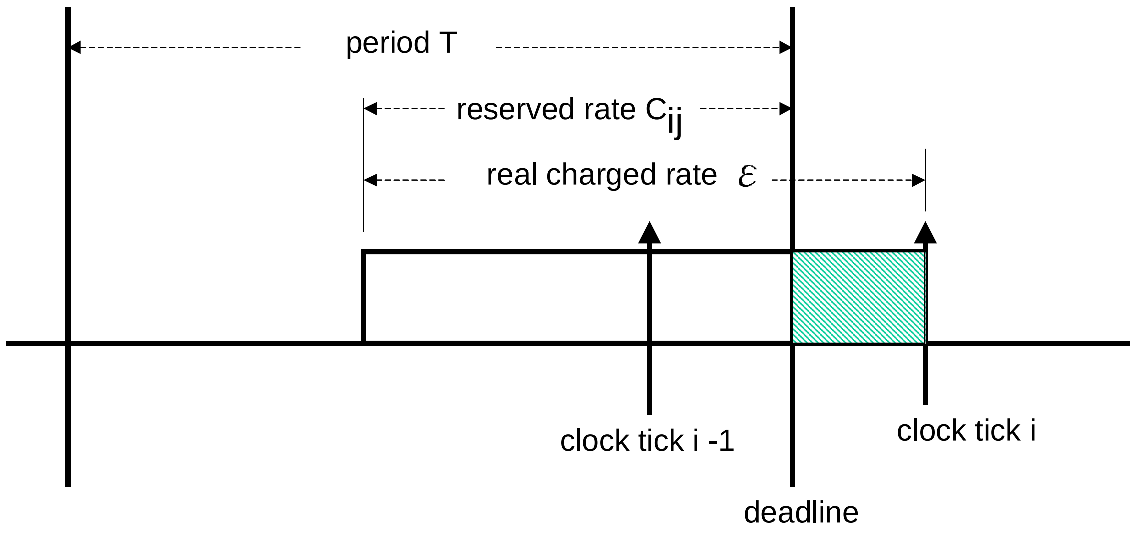


Figure 4: Illustration about over charges in VD.

processor time. Figure 4 illustrates this case. Even the extra consumption is small but should not happen. So, line 10's purpose is to re-charge the over-used time on RRS.

At line 12, if it is RRS running, then the time is charged to RRS by calculating its virtual time. Finally, at line 14 of Figure 3, virtual deadline is calculated. It is assumed that RRS will the exhaust next time quantum. So by fairly calculating its share of this time quantum, the virtual deadline should be the one illustrated at line 14 of Figure 3.

Admission Control Algorithm

The admission control algorithm is illustrated at Figure 5. It considers three resources at one time, which are illustrated at line 3, 4 and 5. We reserve β_1, β_2 and β_3 in the system because the system still has some system works to do, rather than just servicing multimedia applications. In addition, if an application requests less resource than it needs, this application cannot get help from RRS if the system remains too few resources to RRS. Using β_1, β_2 and β_3 in the admission control algorithm can reserve a percentage of resource in the system, that guarantees the RRS will not blocked anytime. Currently, β_1, β_2 and β_3 are set to 10%, which perform good enough in current implementation.

```
Algorithm admit_ctl( $h_{ij}$ )
1.  $T_D$ is set to $min\{ T_{ij} | \forall i, j, if\ h_{ij}\ exists\}$
2.  $C_d = \Sigma \gamma_{ij} \cdot T_D / R \cdot S_D$
3.  if ( ($C_D / T_D + \Sigma C_{ij} / T_{ij} \leqq 1 - \beta_1$ ) and
4.        ($\Sigma \gamma_{ij} \leqq R \cdot (1- \beta_2)$) and
5.        ($\Sigma (m \cdot \gamma_{ij} \cdot T_{ij}) \leqq B \cdot (1 - \beta_3 )$) )
6.  {
7.        add $h_{ij}$ to the system
8.        update $U = U - C_{ij} / T_{ij}$
9.        update $d_{ij}$ = current time $t + T_{ij}$
10.       update $V = v + Q / U$.
11.       Pick a thread with an earliest deadline to run
12. }
```

Figure 5: Admission control algorithm

At line 1 of Figure 5, T_D is set to the minimal period of all running threads. The purpose is to let the data manager (DM) appears frequently during each period of time. This will cause the DM not to appear at one time and then monopolize the resource for a long period.

Line 2 is about how to calculate the processing time of DM. We need to reset C_D here because the new job will ask new data rate from DM. DM consequently needs to get more processor time to achieve this requirement. By calculating the added processor time, the C_D is reassigned. Line 3 through line 5 are the filters to admit or reject the applications. Line 3 considers whether the total processor rates are enough to meet applications' need. It takes the data manager's processor rates into consideration too. At line 5, the B is the available buffer size used in *Lyra*. The m is an indicator to show how the buffer is allocated to the applications. For example, if data manager expects that the buffer should be 50% filled with data, then m is 2. If it is expected that only 25% of allocated buffer filled with data, then m is set to 4. If m is 1, it means the data producer (DM) and the data consumer (the application) should precisely produce and consume the data at the same rate without any noisy effects. The larger the m, the more buffer is allocated, thus applications can regulate their playback more smoothly.

Adaptation Algorithm

The adaptation scheme current is simple i *Lyra*. Via monitoring the C_{ij}, we set a window size ω to the number of frames played per second, and then use following formula to adapt the real rate that an application actually needs. Figure 6 illustrates the adaptive changes for C_{ij}.

$$C_{ij} = (\omega - 1) / \omega \bullet C_{ij} + C^{k}_{ij} / \omega \qquad (1)$$

Theoretic Results

There are some theoretic results that illustrated at following. Due to the size limitation, the proof is not presented in this paper. Two results are presented. The first result shows that introducing RRS into the system to do out-of-share jobs will not cause any more deadline misses, than without using RRS. The second result is about media synchronization. If two threads have the same period size and all are admitted into the system with the same release time, they are guaranteed to have synchronization skew bounded by their period. This means applications can use this feature to do media synchronization without too much efforts. The detailed theorem expressions are presented in following.

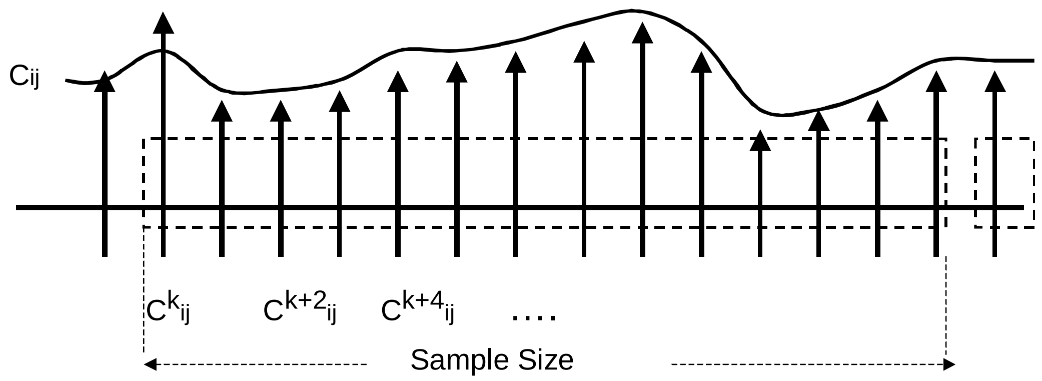


Figure 6: Illustration about how to do adaptation.

Theorem 1: Let D^t_{ij} be the total number of deadline misses that thread h_{ij} encountered from release time τ_{ij} to time t, if the RRS is introduced into the system and scheduled by the virtual deadline algorithm. And let D'^t_{ij} be the total number of deadlines misses thread h_{ij} encountered from release time τ_{ij} to time t, if the RRS is not introduced into the system. If there is a fix set of threads running on the system, then $D^t_i \leqq D'^t_{ij}, \forall t \geqq \tau_{ij}$.

Definition 6: f^n_{ij} is the finishing time that thread h_{ij} finishes its work at its nth period. Define the synchronization skew $Z^{(m,n)}(h_{ij}, h_{st}) = | f^m_{ij} - f^n_{st} | \forall i,j,s,t,n, \forall$ threads h_{ij} *and* h_{st} in manipulating different data streams at different periods.

Theorem 2: If threads h_{ij} *and* h_{st} have the same period T, and at their nth periods, no threads finish their works by getting help from RRS, then $Z^{(n,n)}(h_{ij}, h_{st}) \leqq T_{ij}$, where $T = T_{ij} = T_{st}, \forall i,j,s,t,n$.

6. Evaluations

We focus the evaluations on two points. The first is about the firewall protection property. If there are applications greedily consuming resources, they cannot consume the allocated resources of other applications. Second, we showed in the evaluations that once multiple resources are taken into considerations, deadline misses will be reduced, no matter whether applications are greedy in resource consumption. We also implemented other approaches proposed in previous published literature, and compare the deadline misses with them.

Firewall protection

The first experiment illustrates the firewall property of our scheduling algorithm. Table 1 summarizes the parameters used in our experiments. Table2 shows the characteristics of applications. The actual computation time distribution is normal distribution with standard deviation illustrated next to the computation time value. Application 4 is greedy. Whenever it is scheduled, it runs without relinquishing processor voluntarily. All applications are started at time 0. We collect data from time 0 to time 30000 (in ms).

We illustrate the share an application gets in Figure 8 to 11. Note that because accessing data also needs processor time, each application in other scheduling algorithms needs more processor time than requested. For our approach, ideally the calculated processor share should be close to C_{ij}/T_{ij}, while for

Table 1: Experimental parameters

Variable	Value
m	4
S_D	0.2
$\beta_1,\ \beta_2,\ \beta_3$	0.1
R	40 Mbps
B	16 MByte
System tick interval	1 ms

others should be equal to $C_{ij}/T_{ij} + \gamma_{ij}\ / R \cdot S_D$. We listed the actual processor share received in Table 3. From Table 3, we see that the greedy application does not use up all the resources. Instead, all applications execute at their estimated rates. Hence, a greedy task cannot disturb other applications that behave well. For other algorithms, RC > MTR-LS > EDF in view of fairness.

Table 4 lists the number of deadlines misses of each application. We could see that in EDF, the numbers are larger than the others. Hence, EDF is not good for firewall protection. In this case, our approach is better than other algorithms, because we only buffer four times data (m=4) that an application will use in each period. When the buffer is empty, the application is put to sleep by our assumption.

Admission Control

The admission control mechanism used in our approach considers processor and I/O simultaneously, while traditional EDF, and RC algorithm only take processor resource into consideration. The admission control mechanism used in MTR-LS uses a constant value for I/O access. We argue for the necessity. We believe that to re-compute I/O needed processor time in doing admission controls is necessary. The reasons are as follows. First, for different processor speeds, machine models, and I/O devices, the time needed for I/O are not the same. We could use a system micro benchmark to get the maximum needed processor time for any given devices and machines, i.e. we could get R and S_D. This means that a constant processor share reserved for doing I/O is not suitable for a dynamic environment, where their value might be too large. Second, if only processor time is considered, there may be cases where the sum of processor time is below 1. Many admission control mechanisms will admit the job. But adding this application could cause system overloaded, because the admitted application will request I/O services that also consume processor time.

Table 2: Application requested resources.

	Computation time (ms)	Period (ms)	Data rate (Mbps)	Requested Processor Share	Total Processor Share(plus I/O)
AP 1	5(0.2)	30	6	0.167	0.197
AP 2	10(1)	33	20	0.303	0.403
AP 3	22(1)	100	8	0.22	0.26
AP 4	1(greedy)	33	1(greedy)	1/33	1

Table 3: Effective average processor share received by each application.

	AP1	AP2	AP3	AP4
Our approach	0.165982	0.300386	0.218981	0.119014
EDF	0.154669	0.371780	0.146898	0.326738
MTR-LS	0.174820	0.375589	0.259753	0.189898
RC	0.196713	0.359160	0.260031	0.184152

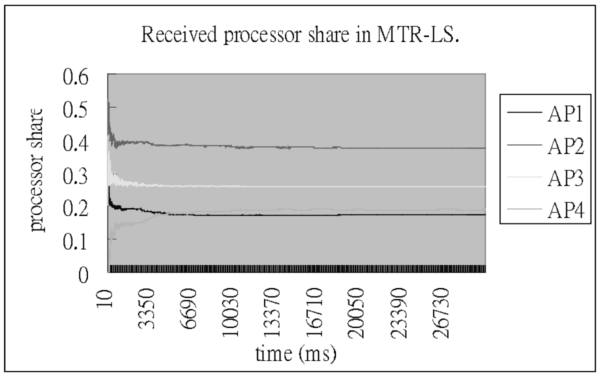
Table 4: Number of deadlines missed.

	AP1	AP2	AP3	AP4
Our approach	5	13	6	909
EDF	285	142	266	909
MTR-LS	163	148	4	909
RC	0	259	1	909

Table 5: Number of deadlines missed before and after adding the fourth application.

	AP1	AP2	AP3	AP4
Our approach	0, 0	0, 0	0, 0	-, -
EDF	0, 60	0, 80	0, 91	-, 103
MTR-LS	142, 158	82, 160	0, 12	-, 334
RC	0, 132	259, 349	0, 67	-, 83

We do an experiment in this section to show that our approach does not admit the fourth application, while others do. The characteristics of applications are the same as previous experiment except for the AP 4. AP 4 has computation time 3, period 15, and data rate 6. For algorithms only consider processor resource, the total processor time needed for these four applications is 89%. Hence, it is admitted while



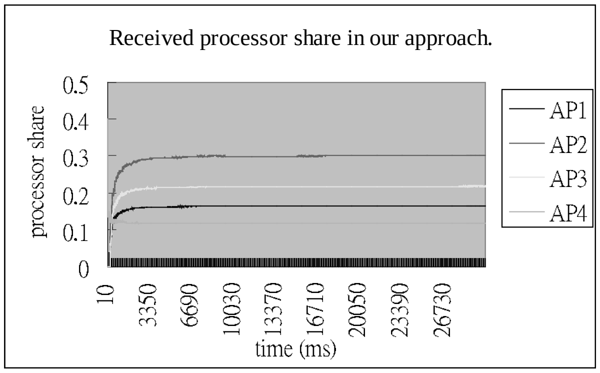
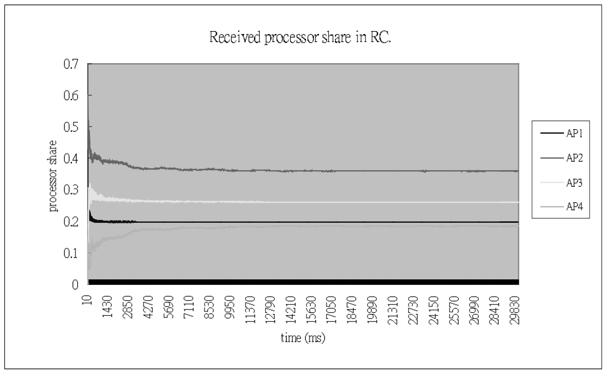


Figure 7-8: Received processors in MTR-LS and VD algorithms



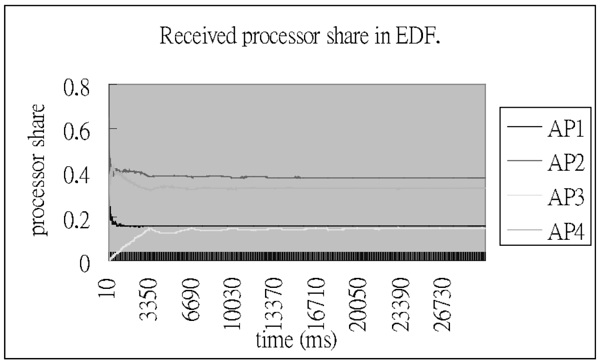


Figure 9-10: Received processors in RC and EDF algorithms

in our approach is not, because the calculated value is 109%. Table 5 shows the number of deadlines missed before and after adding the fourth application.

7. Related Work

Hard Real-Time Related Scheduling Algorithms

Hard real-time applications cannot tolerate miss deadlines. Hence, algorithms developed for this type of applications typically have strict admission control constraints. Applications also must have predictable behaviors in program executions. Rate-monotonic (RM) and earliest deadline first (EDF) scheduling algorithms are two famous algorithms developed for real-time systems [Liu73]. RM statically assigns priority to tasks according to their periods. The one with shorter period has higher priority. EDF dynamically assigns priority to tasks according to their deadlines. The one that has smaller deadline value, i.e. more urgent, has higher priority. Many researchers applied these two scheduling algorithms to support multimedia applications [Mer94,Raj98]. They considered the problem of processor capacity reserves [Mer94], and they use RM and EDF for scheduling CPU. Later, a resource-centric approach is proposed [Raj98]. In their work, they also consider other resources other than the processor, but only do reservations respectively. Since doing I/O also needs processor time, their reservations are not so precise, especially to the multimedia applications.

For handling aperiodical events in a real-time system, many approaches were proposed. Our approach is similar to the total bandwidth server [But97], which handles aperiodical events by using unreserved resources without interference to hard real-time tasks. The different points are as follows. First, the total bandwidth server requires knowing the aperiodical request's length in advance. We assign the request's length to one quantum time because to estimate the length of request is not easy. Second, the total bandwidth server's time-base point is set to *Max{current_time, time_base}*. The time when an event occurs is defined to the current time. In our approach, we use the virtual time to calculate the virtual deadline. The time base is earlier than the current time. Thus the RRS can be set in a earlier virtual deadline and to prevent applications from missing mode deadlines. Though the deadline of RRS is shortened, RRS will not violate the firewall protections.

Fair Queuing Related Scheduling Algorithms

Fair queuing related algorithms were designed to fairly distribute resources among competing tasks. For example, in network bandwidth allocation, the most concerned points are fairness, delay bounds and jitters. Many works are done in order to provide these guarantees [Ben96, Dem89, Goy96b, Zha91, and Zha95]. There is a difference between network and operating system scheduling. In the network case, when a packet arrives, the needed service time is known because we could see the packet's length. In the operating system case, it is harder to know exact computation time that a request needs. Hence in our approach, in calculating the virtual time of RRS, we only increases one quantum after the consumption length is known. That is to say, we use coarse-grained approximation in fairness because to meet deadlines is multimedia applications' primary concern. Multimedia applications care less about how fine-grained fairness could be achieved.

The rate-controlled (RC) [Yau96] and move-to-rear list scheduling (MTR-LS) [Bru97] are two proportional-share algorithms. The RC algorithm is similar to the VirtualClock [Zha91] and WFQ [Ben96], but it increases in integral multiple values of application's periods, i.e. it uses coarse-grained approximation. MTR-LS algorithm is similar to weighted-round-robin (WRR) algorithm with finer modifications. Application will stay at the original position in the ready queue when it is not ready for this type of resource.

Hierarchical Scheduling Algorithms

To mix various types of application in an operating system, a single algorithm to schedule all these types of application is not easy to develop. Hence, many hierarchical resource partition algorithms are proposed. In this way, each partitioned part can use different scheduling algorithms for different task sets. A hierarchical scheduling algorithm must be fair enough, i.e. fine-grained approximation to the ideal fluid flow system, in order to give the underlying part the illusion that they are run by a slower processor. For example, start time fair queuing (SFQ) [Goy96b] is such an algorithm. The SFQ algorithm is similar to SCFQ proposed in the network bandwidth allocation. A difference between SFQ and SCFQ is the way to charge resource usage. SCFQ needs to know requests' length in advance, while SFQ charges resource usage after the request is ended. Our approach also charges resource usage after the resource has used.

8. Conclusions

In this paper, a system framework is introduced to support multimedia applications running smoothly. Our purpose is to let applications run under their share allocations. By dividing their processing time in their periods, system can know the share allocations. If all applications do not use more time than what they are granted, applications can run smoothly that will not cause any deadline misses. However, real world multimedia applications do not have fixed processing time from time to time. This means that applications might want to consume more processor time in handling their extra jobs. For solving this problem, we introduce the RRS into the system to use the remaining rate of the system to help these applications. By using the virtual deadline algorithm, the RRS will not influence the other applications' normal executions, but still tries to reduce the number of deadline misses of the system.

Our contributions can be illustrated as follows. First, our work lets application designers easily to design and run their applications without guessing the difficult timing parameters. The framework tries to minimize the total number of deadline misses of the system, if there are enough resource share remained for such kind of jobs. Even so, it does not violate the firewall protection property. Through empirical evaluations, this property is proved.

In addition, by considering the practical situations, multiple resources are considered in doing admission control. This part though is simple and direct, but influences a lot to practical application implementations and program executions. However, it is usually ignored. We add this into the framework design, and from the simulation comparisons, *Lyra* does provide a better environment for multimedia applications than previous works. Besides, the introduced framework can support applications to do media synchronization easily. This property is proved in theoretic analysis.

Reference

[Bav98] Andy C. Bavier, A. Brandy Montz and Larry L. Peterson, "Predicting MPEG Execution Times", *In proceedings of the ACM SIGMETRICS'98*, Madison, USA, June 1998.

[Ber95] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers and Susan Eggers, "Extensibility, Safety and performance in the SPIN Operating System", In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, December 1995.

[Ben96] Jon C. R. Bennett and Hui Zhang, 'WF^2Q: Worst-case Fair Weighted Fair Queueing', *Proceedings of INFOCOM 96*, San Francisco, California, March 1996.

[Bru97] J. Bruno, E. Gabber, B. Ozden and A. Silberschatz, 'Move-to-Rear List Scheduling: a new scheduling algorithm for providing QoS guarantees', *Proceedings of ACM Multimedia 97*, Seattle, WA, November 1997.

[Bru98] J. Bruno, E. Gabber, B. Ozden and A. Silberschatz, 'The Eclipse Operating System: Providing Quality of Service via Resource Domains', *Proceedings of the USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[But97] Giorgio C. Buttazzo, Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers 1997.

[Dem89] Alan Demers, Srinivasan Keshav and Scott Shenker, 'Analysis and simulation of a fair queueing algorithm', *ACM SIGCOMM*, 19, (4), pp. 2-12(1989).

[Goy96a] Pawan Goyal, Xingang Guo and Harrick M. Vin, 'A Hierarchical CPU Scheduler for Multimedia Operating Systems', *Proceedings of Second Usenix Symposium on Operating System Design and Implementation*, Seattle, WA, October, 1996.

[Goy96b] Pawan Goyal, Harrick M. Vin and Haichen Cheng, 'Start-time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks', *Proceedings of SIGCOMM96*, California, 1996.

[Jon97] Michael B. Jones, Daniela Rosu and Marcel-Catalin Rosu, ‘CPU Reservations and Time Constraints: Efficient Predictable Scheduling of Independent Activities’, *Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.

[Kim94] Manhee Kim, Hyogun Lee and Joonwon Lee, ‘A Proportional-Share Scheduler for Multimedia Applications’, *Proceedings of the International Conference on Multimedia Computing and Systems*, Boston, Massachusetts, May 1994.

[Lee96] Paul C. H. Lee, Meng Chang Chen and Ruei-Chuan Chang, 1996, ‘*Hipec*: A System for Application-Customized Virtual-Memory Caching Management’, *Software: Practice and Experiences*, 27(5), 547-572.

[Lee97] Paul C. H. Lee, ‘From Micro Kernels to Micro Cores’, Technical Report, Institute of Information Science, Acamedia Sinica, Taiwan, 1998.

[Lee98] Paul C. H. Lee, Chi-Wei Yang, Mei-Ling Chiang, Yung-Li Jih and Ruei-Chuan Chang, ‘Performance Analysis of Software MPEG-II Audio/Video Players’, *In the proceedings of the IEEE International Symposium on Consumer Electronics (ISCE’98)*, 1998.

[Liu73] C. L. Liu and J. W. Layland, ‘Scheduling algorithms for Multiprogramming in a Hard-Real-Time Environment’, *Journal of ACM*, 20, (1), pp. 46-61(1973).

[Mer93] Clifford W. Mercer, Ragunathan Rajkumar and Hideyuki Tokuda, ‘Applying Hard Real-time Technology to Multimedia Systems’, *In the proceedings of the Workshop of the Role of Real-time in Multimedia/Interactive Computing Systems*, Raleign-Durham, NC, November 1993.

[Mer94] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda, ‘Processor Capacity Reserves: Operating System Support for Multimedia Applications’, *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Boston, Massachusetts, May 1994.

[Mit96] J. Mitchell, W. Pennebaker, C. Fogg and D. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, 1996.

[Nie93] J. Nieh, J. G. Hanko, J. D. Northcutt, G. A. Wall, ‘SVR4 UNIX Scheduler Unacceptable for Multimedia Applications’, *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster, U.K., November 1993.

[Nie97] J. Nieh and Monica S. Lam, ‘The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications’, *In proceedings of the Sixteenth ACM Symposium on*

Operating Systems Principles, Saint-Malo, France, October 1997.

[Par93] Abhay K. Parekh and Robert G. Gallager, ‘A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case’, *IEEE/ACM Transactions on Networking*, 1, (3), pp. 344-357(1993).

[Raj98] Raj Rajkumar, Kanaka Juvva, Anastasio Molano and Shui Oikawa, ‘Resource Kernels: A Resource-Centric Approach to Real-time Systems’, *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, San Jose, California, January 1998.

[Sto96] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke and G. Plaxton, ‘A Proportional Share Resource Allocation for Real-Time, Time-Shared Systems’, *Proceedings of 17th Real-Time Systems Symposium*, Washington DC, December 1996.

[Wal95] Carl A. Waldspurger and William E. Weihl, ‘Stride Scheduling: Deterministic Proportional-Share Resource Management’, Technical Memorandum MIT/LCS/TM-528, MIT Laboratory for Computer Science, June 1995.

[Wan98] Y.-C. Wang and K. J. Lin, “Enhancing the Real-Time Capability of the Linux Kernel “, Proceedings of the 5th International Conference on Real Time Computing Systems and Applications, October 1998.

[Xie95] Geoffrey G. Xie and Simon S. Lam, ‘Delay Guarantee of Virtual Clock Server’, *IEEE/ACM Transactions on Networking*, 3, (6), pp.683-689(1995).

[Yau96] David K. Y. Yau and Simon S. Lam, 'Adaptive Rate-Controlled Scheduling for Multimedia Applications', *Proceedings of ACM Multimedia 96*, Boston, MA, November 1996.

[Zha91] Lixia Zhang, ‘VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks’, *ACM Transactions on Computer Systems*, 9, (2), pp. 101-124(1991).

[Zha95] Hui Zhang, ‘Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks’, *Proceedings of the IEEE*, 83, (10), 1995.