Nonpreemptive Scheduling Algorithms for Multimedia Communication in Local Area Networks *

S. Eun

Dept. of Infor. Commu. Eng.
Han Nam University
133 Ojung-Dong, Taejon 300-791, Korea
E-mail: sbeun@hudice.hannam.ac.kr

J. Kim, B. Kim, H. Yoon, and S.R. Maeng

Dept. of Compu. Sci.
KAIST
373-1 Kusong-Dong, Taejon 305-701, Korea
E-mail: jckim@camars.kaist.ac.kr

Abstract

We consider a LAN-based multimedia information system like a Video On Demand(VOD) system that supports the retrieval of continuous media like motion video and sound. In the system, the server transmits the streams of continuous media on a shared communication channel while the continuity of multiple streams should be preserved. Several scheduling algorithms have been studied to guarantee the temporal constraints of time-critical messages, but there has been no study to schedule periodic transmission requests with variable bit rates(VBR), which results from the compression algorithms for motion video and sound. We suggest real-time scheduling algorithms that one is static and the other is dynamic, and an admission control algorithm to guarantee the delivery of continuous media. The characteristics of our algorithms are that it is nonpreemptive to save the overheads of preemption, and the static scheduling algorithm is proved to be optimal. It is shown through simulations that the performance of our dynamic scheduling algorithm is better than that of nonpreemptive Earliest Deadline First(EDF) algorithm, especially, under the assumption of variable bit rates.

1 Introduction

We consider a LAN-based multimedia information system like a Video On Demand(VOD) system that supports the retrieval of continuous media like motion video and sound. There have been several researches[1, 2, 3] to provide huge amount of concurrent accesses to many video titles contemporarily. Those systems should be equipped with broadband communication networks, a lot of processors, thousands of disks, and ATM switches. But, a small sized, dedicated, and rather inexpensive multimedia information system may be required for small organizations like elementary schools and small libraries, which supports only tens or twenties concurrent accesses in a Local Area Network(LAN) like Ethernet.

In most multimedia information systems, handling continuous media is a class of real-time engineering. For example, let us consider a motion video that is retrived from the file system where the stream should be retrieved at a constant rate, 30 frames/sec in NTSC video. The flow should be also transferred to a client at the same data rate, and finally, presented to the monitor in the client. The rate is up to 1.5Mbits/sec in MPEG1[4]. If the continuity of the data stream is not preserved even in a part of the flow from the hard disk to the presentation device, the presentation may suffer from the jitter. To guarantee the continuity of the stream, real-time scheduling techniques are usually employed in various levels such as disk head scheduling[5, 6], CPU scheduling[7], and communication channels[8, 9, 1].

We concentrate on the scheduling problem in the level of communication channel. As shown in Figure 1, clients make concurrent requests of the retrieval of multimedia information comprising continuous media. The server issues the continuous data stream on a shared communication channel like Ethernet. The server examines the current status of network traffic and approves the connection if the communication channel can accommodate the request. It is assumed that the continuous streams of motion video and sound are retrieved from the storage unit to the communication unit. It should be noticed that there have been several researches to guarantee the continuous retrieval[5, 6].

In this paper, we investigate the characteristics of communication patterns in a small and dedicated multimedia information system in which the transmission of continuous data is naturally periodic and nonpreemptive, the transmission requests are heterogeneous, and multimedia data have variable bit rates. We, at first, present a static algorithm to schedule periodic tasks and prove that the algorithm is optimal, that is, it can find a feasible schedule whenever there exists at least a feasible one. We also present a dynamic scheduling algorithm for scheduling communication requests dynamically and suggest an admission control algorithm to guarantee the delivery of continuous media for multimedia communication. The characteristics of our algorithms are that they are nonpreemptive to save the overheads of context switching

*This work was partially supported by *Center for Artificial Intelligence Research*of KAIST.

0-8186-7216-1/95 $04.00 © 1995 IEEE

and the static algorithm is proved to be optimal. The performance of our dynamic algorithm is compared with nonpreemptive Earliest Deadline First(EDF) algorithm through simulations, under the assumption of both fixed bit rates and variable bit rates.

In Section 2, the issues inherent in multimedia communication are described. In Section 3, a static scheduling algorithm is proposed and its optimality is proved. In Section 4, a dynamic scheduling algorithm and an addmission algorithm are presented for multimedia communication. In Section 5, performance evaluation is described, and finally, we conclude in Section 6.

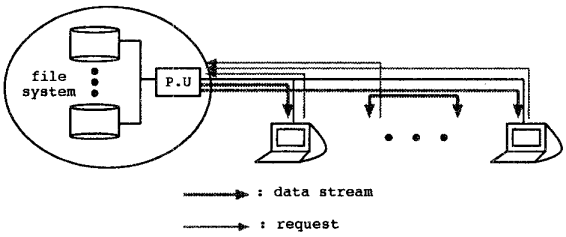

Figure 1: LAN-Based Video On Demand System

2 Backgrounds

In this section, we describe the issues inherent in multimedia communication, and also describe related work.

2.1 Issues in Multimedia Communication

2.1.1 Variable Bit Rates

In a multimedia information system, the server retrieves data blocks continuously from the storage unit, and transmits them continuously via communication channels. From the nature of motion video and sound, the process of data delivery is *periodic*. For example, data blocks for motion video should be handled at the constant rate of 30 frames/sec.

In conventional real-time communication, the communication overhead in each period is often assumed to be constant through all periods , for example the period is 10 and the communication overhead is 5. Then, it means that the communication overhead remains equal to 5 through all periods. But, it has been reported in a literature[10] that compressed multimedia data have variable bit rates(VBR). For example, although the average is 5, the communication overhead may be 6 or 4 in a period. This is mainly caused by compression algorithms like MPEG since the static scenes are more compressed than dynamic scenes.

2.1.2 Heterogeneous Transmission Requests

In a simple multimedia information system with only homogeneous video sources, the frequency transmitting such videos is equal to a certain rate, for example, 30 frames/sec in NTSC videos. In such a system, it is sufficient that multiple streams of videos are serviced in the manner of round-robin since they have the same period of data request and the period is corresponding to one round. In [1], data requests are assumed to be serviced in the round-robin manner, and Baek[9] developed an admission control scheme under the assumption of the same period.

However, even in a simple multimedia information system, communication requests with variable frequencies should be served. For example, there may be a request that a series of still images is presented consecutively at the period of once per second. In a teleconferencing application, a motion video captured in a client at a lower rate, 10 frames/second, may be transmitted spontaneously to other clients.

2.1.3 Nonpreemptive Scheduling

Preemptive scheduling allows the preemption of current transmission of a data stream. Preemption is required when urgent transmission requests with higher priorities are arrived. For implementing the preemption in communication systems, data blocks should be packetized, which causes the overhead of packetization and context switching.

However, in a small and dedicated multimedia information system, unpredictable preemption is not necessary since the rigid requirements of scheduling the transmission can be made in advance. Just keeping the requirements makes the system satisfy the timing constraints of itself. Nonpreemptive scheduling also has the advantage that it is simple to implement. From this point of view, nonpreemptive scheduling is considered in our multimedia communication system.

2.2 Related Work

In the scheduling of transmission requests in communication channels, we can divide related work into two groups, opportunistic and contract scheduling, according to their characteristics. The opprotunistic scheduling tries to adapt itself to the immediate needs of processing continuous media, which has no knowledge in advance about the future behavior of its environment. It serves the most important process in the current jobs, which is often determined by its fixed priority. This kind of scheduling is appropriate for complex, less predictable, and general systems as it was noticed that fixed priority scheduling algorithms had been often used in conventional real-time systems[11, 12, 13].

On the contrary, the contract scheduling algorithm makes contracts on several parts of a multimedia system. Timing requirements should be satisfied in each part of the system as rigidly as possible, which makes the whole system more predictable and stable. Although some fluctuations happen due to transient overload, the expansion of the fluctuations can possibly be minimized. The contract scheduling is adequate when the prediction of behaviors of handling continuous media is possible in small, dedicated systems. As a contract scheduling algorithm, Zheng[8] proposed EDF algorithm to schedule periodic requests in a point-to-point channel and admission control condition to determine whether a newly arrived request can be serviced satisfying its timing constraints. Baek[9]

also considered nonpreemptive EDF algorithm and suggested admission control conditions in two limited cases. One is that all computation times are the same and the other is that all periods are the same. With the assumption of homogeneous requests, the round-robin algorithm was proposed in [1]. Among the researches on this type of scheduling, homogeneous requests are assumed in [9] and [1], and EDF algorithm adopted by Zheng[8] is not optimal to schedule the set of nonperiodic tasks.

While preemptive scheduling of periodic tasks has been studied actively to find optimal and efficient scheduling algorithms [14, 15, 16], related work on nonpreemptive schedulings has been found less frequently, especially for periodic tasks. Jeffay[17] showed that scheduling nonpreemptive periodic tasks is NP-hard and proposed a set of conditions that a nonpreemptive Earliest Deadline First(EDF) algorithm would be optimal. There was also a research [18] on the efficient and optimal scheduling of the tasks with the specific periods satisfying $p_{i+1} = k \times p_i, k \geq 2$ where p_i is the period of task,t_i, and k is an arbitrary integer. However, the constraint on the periods is not general.

if we assume the small and dedicated multimedia information system in which the prediction of behaviors of handling continuous media is possible, the contract scheduling is preferred to the oppertunistic one since it can optimize resources like buffers and performance.

3 Static Scheduling Algorithm

In this section, we propose an optimal scheduling algorithm in a static way where a set of periodic tasks is given and scheduled before actual running. A set of n tasks, $\mathcal{T}$, is defined as $\{t_1, t_2, \cdots, t_n\}$, each of which is characterized by two components, (p_i, c_i), where p_i is the period of t_i and c_i is the computation time for an invocation of t_i. Tasks are sorted in the ascending order according to their periods, so t_1 has the shortest period. We call each invocation of tasks an *instance* and denote it as t_{ik} that means the kth invocation of t_i. Since a task is nonpreemptive, every instance of it should run to completion once started. The jth instance of t_i should start after $(j-1) \times p_i$ form the origin, and should be completed no later than $j \times p_i$. Our algorithm is optimal in a sense that it can always produce a schedule if there exists at least a feasible schedule.

Our scheduling algorithm is based on the tasks with the periods which satisfy the following constraint to be optimal.

$$p_{i+1} = \frac{m_i}{m_i - 1} LCM(p_1, p_2, \cdots, p_i) \qquad (1)$$

m_i, called the *concatenation factor*, is an arbitrary integer greater than 1, which represents the number of partial schedules to be concatenated to provide the gaps for the instances of the next task. For example, when $p_1 = 2$ and the concatenation factors, $< m_1, m_2, m_3 > = < 3, 3, 2 >$ are given, then the set of corresponding tasks$\{t_1, t_2, t_3, t_4\}$ is $\{(2,-),(3,-),(9,-),(36,-)\}$. Our constraint is much more general than one for the optimal scheduling algorithm in [18], and we believe that the constraint is reasonalble since the periods of tasks which are communication requests in the multimedia communication in LAN environment are not so various, and so, they are grouped into a small number of subsets. For example, a motion video has the capture rates of 10 frames/sec, 20 frames/sec, and 30 frames/sec.

We call our algorithm Largest Gap First(LGF) algorithm in which each instance of t_{i+1} is scheduled in the largest gaps in the partial schedule of i tasks. We show that LGF algorithm is optimal, that is, a feasible schedule can be found whenever there exists at least a feasible one.

3.1 LGF algorithm

LGF algorithm is similar to those with the greedy method[19]. At first, it schedules t_1 and if it is feasible, t_2 is considered to be scheduled with the partial schedule of t_1. It goes on iteratively until t_n is scheduled. Let $\mathcal{S}_i$ be a subset of $\mathcal{T}$ with i tasks, $\{t_1, t_2, \cdots, t_i\}$ and σ_i be the schedule of $\mathcal{S}_i$ from origin to $LCM(p_1, p_2, \cdots, p_i)$. In the algorithm, m_i σ_i's are concatenated and then $m_i - 1$ instances of t_{i+1} are inserted into the appropriate gaps as inferred from Eq. (1).

LGF algorithm investigates the marginal gaps in each σ_i, and determines where $m_i - 1$ instances of t_{i+1} are inserted among them. We illustrate the various marginal gaps possible in Figure 2, where a schedule for a task set, $\{(1,-),(2,-),(4,-)\}$, is considered as an example. Black filled boxes represent the instances of t_1 and white boxes represent the instances of a following task. In Figure 2-a), we center the schedule, σ_i to make as large margins as possible at the both ends of σ_i and this schedule is called *centered* σ_i. We then call each of two margins *an external gap* that appears at both left and right ends of the centered σ_i, and denote them together as λ_i whose size is as large as the sum of two margins. An instance is scheduled into one of gaps at the both ends or is often scheduled into the gaps between two concatenated σ_is in LGF algorithm.

In some cases as shown in Figure 2-b), there may be some gaps within the centered σ_i, which are unavoidable due to the existence of ready times and deadlines of instances. We denote the sum of all such margins within σ_i as ϕ_i. The centered σ_i can be moved to the left or right. We denote the maximum movement of it as μ_i shown in Figure 2-c). The movement is restricted by the ready times and deadlines of instances. In the case of the centered σ_i, we can make a margin by widening the gap between two adjacent instances within the boundary of ready time and deadline. There can be as many such gaps as the number of instances in σ_i. Among them, the largest one is called the *internal gap* and denoted as ρ_i as shown in Figure 2-d).

The LGF algorithm consists of three steps as depicted in Figure 3. In the algorithm, we use the above marginal gaps as the parameters to find a feasible schedule. In Step 1, the parameters for σ_i and γ_i, the location of ρ_i, are calculated. Every instance in σ_i is then justified to the left and right, and also cen-

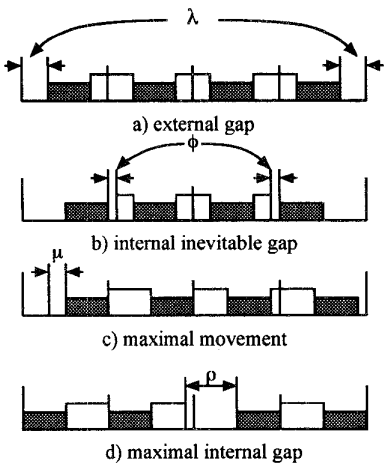

Figure 2: Marginal Gaps within Partial Schedules

tered to make partial schedules τ_1, τ_2, and τ_3. Finally, three parameters, λ_i, μ_i and ρ_i, are obtained in the functions, **center**(), **mu**(), and **rho**() respectively.

In the next steps, the algorithm tries to insert the instances of t_{i+1} into the bigger gap between λ_i and ρ_i to increase the probability to be scheduled. The steps are divided into Step 2 and Step 3 according to whether $\lambda_i \geq \rho_i$ or not. In each step, an adequate feasibility condition is checked, and if the condition is not satisfied, then return the failure of scheduling. Otherwise, in Step 2, σ_i is duplicated m_i times and then they are concatenated. $m_i - 1$ instances of t_{i+1} are finally inserted into the external gaps between two adjacent σ_i's. Similarly, in Step 3, σ_i is also duplicated, and concatenated. Of course, it should be satisfied that $t_{(i+1)k}$ is in the ready state at the time of insertion. The difference from Step 2 is that $t_{(i+1)k}$ is inserted into the internal gap within σ_i, rather than an external gap. The σ_{i+1} is then generated and the above three steps are iterated until all tasks are examined.

We now analyze the time complexity of LGF algorithm. Let N_i be the number of instances in $\mathcal{S}_i$ and N be the number of all instances in $\mathcal{T}$.

Theorem 1 *The time complexity of LGF algorithm is at most $O(nN)$.*

Proof The Step 1 executes five functions and each one requires only one examination of all instances, N_i. In Step 2 and Step 3, N_{i+1} instances are handled. It means that at most $\xi \times N_i, \xi \leq C_0$ instances are examined in each iteration, where C_0 is a certain constant. Since N_i is always less than N and n iterations are executed, the complexity is less than $\xi \times n \times N$. Hence, proved. □

3.2 Optimality of LGF Algorithm

In this section, we show that the LGF algorithm is optimal, that is, the algorithm always finds a feasible schedule whenever there exists at least a feasible one.

```
Schedule( T ) /* T is a set of n tasks */
{
  for i ← 1, n − 1 do {
      /* Step 1 : calculate μ_i, λ_i, ρ_i, and γ_i */
      Calculate(σ_i);

      /* Step 2 */
      if (λ_i ≥ ρ_i) {
        if (c_{i+1} × (m_i − 1) > λ_i × m_i + 2 × μ_i)
          return No feasible schedule;
        σ_{i+1} ← σ_i||t_{(i+1)1}||σ_i|| ··· ||t_{(i+1)k}||σ_i;
        /* k is m_i − 1 */
      }

      /* Step 3 */
      else {
        if (c_{i+1} > ρ_i)
          return No feasible schedule;
        τ ← φ; k ← 1; ct ← 0;
        /* φ is a null schedule */
        for j ← 1, m_i do {
            τ ← τ||σ_i;
            if (γ_i + ct > p_{i+1} × (k − 1)) {
            /* if t_{(i+1)k} is ready */
              insert(t_{(i+1)k}, γ_i + ct);
              k ← k + 1;
            }
            ct ← ct + LCM_i;
        }
        σ_{i+1} ← τ;
      } /* end of else */
  } /* end of for loop */
  return σ_n
}


Calculate( σ_i ) /* σ is a schedule */
{
  if (i = 1) {
    Initialize μ_1 ← (p_1 − c_1)/2; λ_1 ← ρ_1 ← 2 × μ_1;
    return;
  }

  τ_1 ← left-justify(σ_i);
  /* for all instances in σ do {···} */
  τ_2 ← right-justify(σ_i); /*
  for all instances in σ do {···} */
  τ_3 ← center(σ_i); /* obtain λ */
  mu(τ_3, τ_1); /* Min(∀i, j, τ_3.ct_ij − τ_1.ct_ij) */
  rho(τ_2, τ_1); /* obtain ρ */
}
```

Figure 3: Largest Gap First(LGF) Algorithm

Lemma 1 *LGF algorithm always schedules the instances of t_{i+1} into the largest gaps in σ_i.*

Proof When $\lambda_i \geq \rho_i$, the instances are scheduled into the external gaps that are the largest in Step 2. On the contrary, if $\rho_i > \lambda_i$, the internal gaps are the largest in which the instances are scheduled in Step 3. Hence, proved. □

One might think that it would be better to schedule instances at smaller gaps to leave larger gaps for a task with larger computation time. However, we contradict such a heuristic by the following lemmas. Let G_{i+1} be the largest gap in σ_{i+1} generated from σ_i by LGF algorithm and G'_{i+1} be that by an arbitrary scheduling algorithm except LGF algorithm. Similarly, we distinguish the parameters of schedules made by an arbitrary algorithm from those by LGF algorithm by putting primes on them.

Lemma 2 *Given σ_i with $\lambda_i \geq \rho_i$, $G_{i+1} \geq G'_{i+1}$ holds.*

Proof It should be noted that any scheduling algorithm can make a successful schedule only if $c_{i+1} \leq \lambda_i$ holds.

LGF algorithm schedules instances into the external gaps between two adjacent σ_i's when $\lambda_i \geq \rho_i$. Hence, there remain margins at both ends of the centered σ_{i+1}, which are as large as those of σ_i or even larger when μ_i is not 0. It means that $\lambda_{i+1} \geq \lambda_i$ holds. In addition to this, it is clear that $\rho_{i+1} \leq \lambda_i$ holds. Therefore, $G_{i+1} = \lambda_{i+1} \geq \lambda_i$ holds.

An arbitrary algorithm inserts one or more instances into the internal gaps of σ_i and concatenates them to form σ_{i+1}. If an instance is inserted into σ_i, the margins at both ends get less than or equal to the original ones regardless of the size of c_{i+1}. Even though only one instance is inserted into one σ_i, it is clear that $\lambda'_{i+1} \leq \lambda_{i+1}$ holds. ρ'_{i+1} is at most equal to λ_i as well as ρ_{i+1}.

By both cases, $G_{i+1} \geq G'_{i+1}$ holds. Hence, proved. □

Lemma 3 *Given σ_i with $\lambda_i < \rho_i$, $G_{i+1} \geq G'_{i+1}$ holds.*

Proof LGF algorithm schedules instances into the internal gaps within σ_i's. Since the number of internal gaps in σ_{i+1} is bigger than the number of instances by 1, there remains one internal gap unoccupied. It is clear that the size of this internal gap is larger than any other gaps in σ_{i+1} , and therefore, G_{i+1} is equal to the internal gap, ρ_i. If an arbitrary algorithm schedules instances to other locations rather than the internal gap, it results in reducing the size of the internal gap. G'_{i+1} is at most less than or equal to ρ_i. Therefore, $G_{i+1} \geq G'_{i+1}$ holds.

Hence, proved. □

Notice that the proofs of Lemma 2 and 3 are independent of which algorithm has been used to schedule σ_i.

Theorem 2 *LGF algorithm is optimal.*

Proof We show that the contraposition of the theorem is true. Assume that $\mathcal{S}_i$ has been scheduled successfully by LGF algorithm but σ_{i+1} is not schedulable. If we try to schedule instances of t_{i+1} in other locations, it will also fail since the other locations have smaller gaps than previous one as shown by Lemma 1. If we try to backtrack to the previous schedulings to make enough room for t_{i+1}, it will also fail since larger gaps are not made by any trials as shown by Lemma 2 and 3. Therefore, no algorithm can schedule σ_{i+1}. Hence, proved. □

4 Dynamic Scheduling Algorithm and Its Admission Condition

We propose a scheduling algorithm to operate dynamically for multimedia communication. Note that the terms, "request" and "communication time" for the dynamic scheduling algorithm are analogous to the terms, "task" and "computation time" used for the static scheduling algorithm respectively. The dynamic scheduling algorithm operates in the same manner as Earliest Deadline First Algorithm. When a task completes its execution, the dynamic algorithm selects one among the ready requests and dispatch it. When a new request is arrived while a set of requests are serviced, the server should decide whether it admits the request or not. Admission control should be rigid enough to guarantee the timing constraints of all requests once they are admitted.

4.1 Request Model

Let $\mathcal{R} = \{r_1, r_2, \cdots, r_i\}$ be a set of periodic requests. Each request is characterized by $< p_i, c_i >$, where p_i is the period of request r_i and c_i is the communication time of it. In $\mathcal{R}$, requests are sorted according to the period. So, r_1 has the shortest period. Each p_i is assumed to satisfy the same constraint in Eq.(1) used for the static algorithm.

$\mathcal{R}$ is the same as the set of tasks defined in Section 3. Therefore, LGF algorithm is optimal to schedule $\mathcal{R}$. But, we should consider the scheduling in the situation where the traffic of the communication channel is not so busy. Let $\mathcal{U}$ be a subset of $\mathcal{R}$. For example, $\{< 1, - >, < 3, - >\}$ is a subset of $\{< 1, - >, < 2, - >, < 3, - >\}$ and $\{< 1, - >, < 2, - >, < 8, - >\}$ is a subset of $\{< 1, - >, < 2, - >, < 4, - >, < 8, - >\}$. Notice that r_1 should not be discarded in $\mathcal{U}$. We let $\mathcal{U}^*$ be the superset of $\mathcal{U}$ by adding some dummy requests with 0 communication overhead. For example, if $\mathcal{U} = \{< 1, - >, < 3, - >\}$, then $\mathcal{U}^* = \{< 1, - >, < 2, 0 >, < 3, - >\}$. If a set of requests, $\mathcal{U}$, is arrived, we examine whether the set is a subset of $\mathcal{R}$ or not. If not, the requests are rejected. Otherwise, $\mathcal{U}^*$ is made from $\mathcal{U}$. In turn, $\mathcal{U}^*$ is analyzed to be admissible by the admission algorithm defined later and then the requests are scheduled by the dynamic algorithm.

When a periodic request is serviced, the initial service point is important. In our dynamic algorithm, the initial service points of all requests should be set to the origin. Therefore, the service of a new request should be delayed until the start of the next round of the other requests, if the request is not issued at the exact start. For example, while two requests with the

periads of $\frac{1}{30}$ seconds and $\frac{2}{30}$ seconds are served, if a new request with $\frac{3}{30}$ seconds arrives, then the request should wait for at most $\frac{2}{30}$ seconds.

4.2 Dynamic Algorithm

In the communication unit, the buffers for data retrieved from the file system are preserved. The file system fills the buffers satisfying the timing constraints of requests. In our system, the scheduling algorithm is called when the transmission of current data is completed. Then, the scheduling algorithm looks into a list of blocks of data that are ready to be transmitted and selects one of them.

The dynamic algorithm is different from LGF algorithm in some points. The static algorithm decides the schedule of the given tasks after scheduling them from 0 to the LCM of their periods. On the contrary, the dynamic algorithm decides the schedule of the given requests according to the status of the current traffic, the result from the admission test, and the status of the ready queue. Moreover, the dynamic algorithm schedules requests in the ready state as well as EDF algorithm.

```
Flag     turn = ON;
/* ON means to consider the instances of r1 */

Dynamic Scheduling(R)
Set of Tasks     R;
/* set of ready blocks */
{
    int i;

    If (turn == ON) {
        If (the instance of r1 is not ready) Wait;
        Schedule the instance of r1;
        turn = OFF;
        /* OFF means to consider the instance of
           other requests except r1 */
    }
    else {
        If (NONE is ready) Wait;
        i = An instance with Earliest Deadline
            and Earliest Ready-time first;
        If (i != DUMMY)
        /* it is not a dummy task */
            Schedule i;
        turn = ON;
    }
}
```

Figure 4: Dynamic Scheduling Algorithm

Figure 4 shows our dynamic scheduling algorithm. The algorithm schedules an instance of r_1 and then change the turn for the next requests and selects an instance with Earliest Deadline and Earliest Ready-time. Then the turn is changed for r_1 and an instance of it is again scheduled. These procedures are iterated until all the requests are scheduled. By using this recursive operation, the instances of r_1 get enough chances for schedule not to miss the deadline.

Lemma 4 *The dynamic algorithm schedules $\mathcal{U}^*$ to make a schedule equivalent to LGF algorithm.*

Proof Recall that the set of tasks which LGF schedules is a member of $\mathcal{U}$ where each instance of the next task of the member has only one appropriate gap that is determined by its ready time and deadline. The dynamic algorithm also schedules first such requests with earliest deadline, and then, earliest ready time. Hence, proved. □

4.3 Admission Control Algorithm

While a set of requests is being served, a new request can be asked. In that case, the server examines the status of the traffic for the communication channel, and determines whether it admits the request or not. It is required that the service quality for the requests including the new request should be kept once the admission is approved. In our algorithm, it admits the new request if the dynamic algorithm can schedule the set of requests including the new request. It means that the admission control algorithm can be viewed as the schedulability analysis of the dynamic algorithm.

Let $\mathcal{U}^+$ be the set of requests including the new request. Our admission control algorithm receives $\mathcal{U}^+$ and returns yes or no.

```
Admit            (U+)
Set of Tasks     U+; /* set of requests */
{
    int λ, μ, m;
    int i;

    λ=p1 − c1;
    μ=λ;
    m=2;
    For i = 2, n {
        If((μ! = 0)&&((m − 1) × ci) >
              (m − 1) × λ × 2 + 2 × μ)
                  return(no);
        If((μ==0)&&(ci > λ × 2))
                  return(no);
        μ=μ − |λ − ci|;
        λ=λ + sign(|λ − ci|) × μ;
        m=p_{i+1} / (p_{i+1} − LCM(p1,···,pi));
    }

    return(yes);
}
```

Figure 5: Admission Control Algorithm

Let n be the number of requests in $\mathcal{U}^+$.

Theorem 3 *The time complexity of our admission control algorithm is $\mathcal{O}(n)$.*

Proof The algorithm is composed of only one loop bounded by n. Hence, proved. □

Theorem 4 *$\mathcal{U}^+$ is schedulable if it is admitted by the algorithm.*

Proof Recall that μ and λ are the spatial parameters defined in Section 3. μ is the maximal movement of the partial schedule and λ is the maximal gap made at both ends of the partial schedule.

When μ is not 0, the size of the gap where an instance is inserted is $(m_i - 1) \times \lambda \times 2 + 2 \times \mu$ since the partial schedule can be expanded at the both ends. At this time, $m_i - 1$ instances of next requests with the computation time, c_i, should be inserted into the corresponding gap. Therefore, if the first condition is not satisfied, then $\mathcal{U}^+$ can not be scheduled by the dynamic algorithm.

When μ is 0, an instance should be inserted into a gap with the size of $\lambda \times 2$, and then no expansion is possible. Therefore, if the second condition is not satisfied, $\mathcal{U}^+$ can not be scheduled by dynamic algorithm, either.
The values of μ and λ are modified as defined in the algorithm. μ is monotonically decreasing. λ is on decrease when $\lambda - c_i$ is positive and is on increase in the opposite case.

So, if the algorithm admits, $\mathcal{U}^+$ is schedulable since the dynamic algorithm can schedule it. Hence, proved. □

4.4 Adaptation to Various Requests

In practical applications, multiple requests may have the same period. In such a case, the dynamic algorithm can not be applied directly. We propose a grouping strategy to overcome the problem. Given $\mathcal{U}$, multiple requests with the same periods are grouped into a new request to form a new set of requests, $\mathcal{U}^-$, and then it is analyzed by our admission control algorithm. If it is admitted, $\mathcal{U}^-$ can be scheduled by the dynamic algorithm successfully. For example, given $\{< 1, 10 >, < 2, 10 >, < 2, 10 >\}$, $\mathcal{U}^-$ is $\{< 1, 10 >, < 2, 20 >\}$. This is also directly applicable to the static scheduling algorithm.

Theorem 5 *IF the nonpreemptive EDF algorithm schedules $\mathcal{U}^-$, the dynamic algorithm always schedules it.*

Proof In $\mathcal{U}^-$, all requests with the same period have the same ready time and deadline. Since the nonpreemptive EDF algorithm schedules according to deadlines, it schedules the requests contiguously. That is the same with our grouping strategy. Hence, proved. □
This theorem means that the dynamic algorithm shows better schedulability than the nonpreemptive EDF algorithm.

In practical applications, some aperiodic requests should be serviced. For example, clients send their commands sporadically and the server may send discrete media like still images to clients according to the requests of clients. These events are the sources of aperiodic traffic.

In conventional real-time systems, there have been researches to schedule both periodic tasks and sporadic tasks simultaneously. One of them is the polling server method[20], which reserves the periodic task for servicing the sporadic tasks. If a client wants to send an aperiodic data, it waits until a polling server task is ready. This method may cause either some delay of service or waste of resources, but it has the advantage that the implementation is rather simple. In addition, it can achieve good performance if the system is tuned to avoid the waste as much as possible.

5 Performance Evaluation

We have already proved that the LGF algorithm is optimal. For the performance evaluation of our dynamic scheduling algorithm, it is important to evaluate the schedulability of the algorithm, that is, how many sets of tasks the algorithm can schedule. We make a comparison between our dynamic scheduling algorithm and the nonpreemptive EDF algorithm[17, 8] through an extensive simulation. Simulation is done in two cases, one is for the requests with the fixed bit rates, the other is for the requests with the variable bit rates.

A set of requests is characterized by the three parameters which are the number of requests in a set, the period of each request, and the communication time of each request. We evaluate the algorithms with two sets of requests where the number of requests in each of them is 3 and 5 respectively. It is clear that these two sets are enough to evaluate the performance of our dynamic algorithm since the periods of requests in reality are not various. In real applications, there may be many requests with the same period and they are grouped to be scheduled.

For the simulation, we generated sets of periodic requests under the assumption that the communication time of each request is uniformly distributed. In each case, ten thousands sets of requests are generated, and we schedule them with both our dynamic algorithm and the nonpreemptive EDF.

The schedulability of scheduling are affected by two parameters, the number of requests and the utilization factor. When the number of requests in a set is small, it is easy to schedule the set of requests. The more requests are given, the harder the scheduling is. In the case of the utilization factor which specifies the usage of the communication channel, if it is 100(%), there is no space remained in the communication channel. Hence, it is very difficult to schedule such sets of requests. If the utilization factor is less, the scheduling gets easier.

The results of simulation with the fixed bit rates are shown in Figures 6 and 7. The communication time for each request is selected randomly. As shown in the figures, the nonpreemptive EDF algorithm can schedule less sets of requests than our dynamic algorithm in all cases. The difference in the schedulability between two algorithms becomes larger with the increase in the number of requests and the utilization factor. In Figure 7, it is remarkable that the nonpreemptive EDF algorithm can schedule only 20(%) of sets in the case of 100(%) utilization and 5 requests while the dy-

namic algorithm shows 60(%). We can argue that our dynamic algorithm is more efficient than the nonpreemptive EDF algorithm by twice on the average with the fixed bit rates.

Figures 8 and 9 show the results of a simulation for the continuous streams with variable bit rates. A number of request sets are generated with variable communication overheads. For example, when the communication overhead of a request is 10, the communication overhead of each period is variable such as 8, 11, 10, 9, and so on, but, the average is 10. As depicted in the figures, our dynamic algorithm with variable bit rates shows the similar schedulability compared to one with constant bit rates and the schedulability is quite better than that of EDF. This tells us that our dynamic algorithm is effiecient to handle the requests with variable bit rates as fixed bit rates. On the contarary, EDF algorithm makes much less feasible schedules, which tells that EDF algorithm is not stable to schedule continuous streams with variable bit rates.

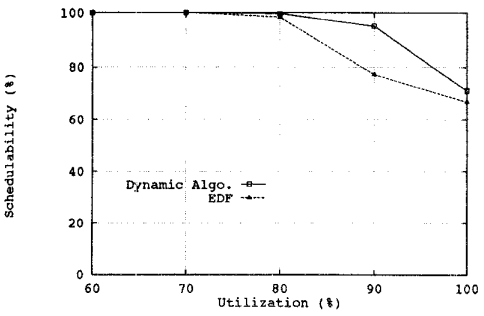

Figure 6: Schedulability for the Request Sets, $\{(1,-),(2,-),(3,-)\}$ with Fixed Bit Rates

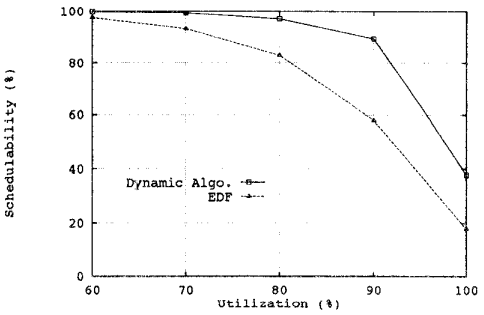

Figure 7: Schedulability for the Request Sets, $\{(1,-),(2,-),(3,-),(8,-),(24,-)\}$ with Fixed Bit Rates

6 Conclusion and Further Research

We have suggested scheduling algorithms and an admission control algorithm to schedule continuous

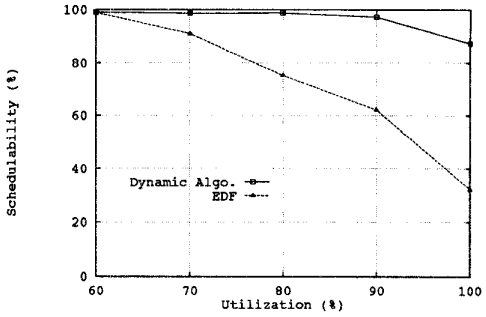

Figure 8: Schedulability for the Request Sets, $\{(1,-),(2,-),(3,-)\}$ with Variable Bit Rates

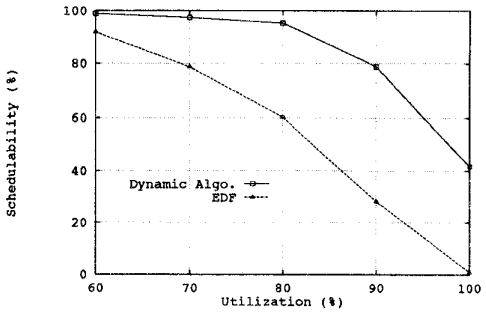

Figure 9: Schedulability for the Request Sets, $\{(1,-),(2,-),(3,-),(8,-),(24,-)\}$ with Variable Bit Rates

media like motion videos and sounds in a small and dedicated LAN-based multimedia information system. The proposed algorithms are nonpreemptive and periodic real-time scheduling algorithms which satisfy the issues in the multimedia communication; variable bit rates, heterogeneous transmission, and nonpreemptive scheduling.

The LGF algorithm which is a static one is proven to be optimal. It is also shown through simulation that our dynamic scheduling algorithm can achieve much better schedulability than EDF algorithm for the requests with fixed bit rates as well as variable bit rates by approximately twice on the average. This justifies that our dynamic algorithm can provide more efficient and stable scheduling than EDF algorithm in multimedia communication.

We believe that our scheduling algorithms and admission control algorithm are appropriate in a small and dedicated multimedia information system like a small VOD system. Therefore, more detailed design of such a system is required as a further research.

References

[1] Kenchammana-Hosekote D.R. and J. Srivastava. Scheduling continuous media in a video-on-

demand server. In *Proc. of IEEE International Conference on Multimedia Computing and System*, pages 19–29, May 1994.

[2] L. Lamont and N.D. Georganas. Synchronization architecture and protocols for a multimedia news service application. In *Proc. of IEEE International Conference on Multimedia Computing and System*, pages 3–8, May 1994.

[3] H.M. Vin, Al. Goyal, An. Goyal, and P. Goyal. An observation-based admission control algorithm for multimedia servers. In *Proc. of IEEE International Conference on Multimedia Computing and System*, pages 234–244, May 1994.

[4] D.L. Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, Apr. 1991.

[5] A.L. N. Reddy and J. Wyllie. Disk scheduling in a multimedia i/o system. In *Proc. of ACM Multimedia'93*, pages 225–234, Aug. 1993.

[6] M.-S. Chen, D. D. Kandlur, and S. Y. Yu. Optimization of the grouped sweeping scheduling(gss) with heterogeneous multimedia streams. In *Proc. of ACM Multimedia'93*, pages 235–242, Aug. 1993.

[7] C.W. Mercer, S. Savage, and H. Tokuda. Process capacity reserves: Operating system support for multimedia applications. In *Proc. of IEEE International Conference on Multimedia Computing and System*, pages 90–99, May 1994.

[8] Zheng. Q. *Real-time Fault-tolerant Communication in Computer Networks.* Phd thesis, University of Michigan, 1993.

[9] Y. Baek and K. Koh. Schedulability analysis on non-preemptive real-time tasks for continuous media retrieval. *Journal of the Korea Information Science Society*, 21(7):1252–1260, Jul. 1994.

[10] Aurel A. Lazar, Giovanni Pacifici, and Dimitrios E. Pendarakis. Modeling video sources for real-time scheduling. In *Proc. of ICC'93*, pages 835–839, 1993.

[11] J. K. Strosnider. *Highly Responsive Real-Time Token Ring.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Melon University, 1988.

[12] L. Sha, R. Rajkumar, and J. Lehoczky. Real-time scheduling support in futurebus+. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 188–197, 1990.

[13] Lui Sha, Mark H. Klein, and John B. Goodenough. Rate monotonic analysis for real-time systems. In *Foundations of Real-TIme Systems: Scheduling and Resource Management*, pages 129–156. Kluwer Academic Publishers, 1991.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[15] Joseph Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letter*, 11(3):115–118, 1980.

[16] Eugene L. Lawler and Charles U. Martel. Scheduling periodically occurring tasks on processors. *Information Processing Letter*, 12(1):9–12, 1981.

[17] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of Real-Time Systems Symposium*, pages 129–139, 1991.

[18] Jr. Mario J. Gonzalez and Jin W. Soh. Periodic job scheduling in a distributed processor system. *IEEE Transactions on Aerospace and Electronic Systems*, AES-12(5):530–536, 1976.

[19] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press Inc., 1978.

[20] L. Sha, B. Sprunt, and J.P. Lehoczky. Aperiodic task scheduling for hard real-time system. *Journal of Real-Time System*, 1:27–60, 1989.