Teik Guan Tan and Wynne Hsu
Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 119260
(65)-772-7944
{tanteikg, whsu}@iscs.nus.sg

Approximating Scheduling for Multimedia Applications under Overload Conditions

ABSTRACT

With the increased demands in multimedia applications, the need to provide better system support is greater than ever. Multimedia applications have an added dimension of time in their execution which results in stringent timing requirements. These timing requirements are usually "soft" constraints, that is, they are potentially relaxable human preference constraints. In this paper, we investigate an approximating scheduling algorithm that attempts to derive a "good" schedule for multimedia processes even under the overload condition. We will present a model which is able to quantify the tradeoff between conflicting goals in a multimedia application. A simulation of the performance of the approximating scheduling algorithm is presented. The results indicate that our proposed algorithm is able to adjust dynamically to transient overload situation without too much degradation in performance.

KEYWORDS: Approximating Scheduling, Predictable Overload Management, Real-time Multimedia

1. INTRODUCTION

Multimedia applications place different demands on computer systems with the added dimension of time in their execution. For instance, the playback of a video sequence has to follow a pre-established frame rate of display to achieve an optimal presentation. Adhering to the timing constraint in the frame rate is essential so that the viewers do not experience too much jitters or unnatural 'jerkiness' in the display. Such constraints imposed on the execution of the application are "soft" constraints where there are penalties incurred if the deadlines are missed but the consequences of missed deadlines are non-critical. We term such systems as soft real-time systems. Many researchers have recognized the need for real-time support for multimedia applications. While many have developed communication protocols [21, 20] and platforms [18] to support real-time transmission of data, scheduling the execution at the end-host computer has not received similar attention. Multimedia applications not only require large amounts of CPU time (to process complex video/audio images) but also demand that the CPU be allocated at pre-determined intervals (following the frame rate). As such, the scheduling mechanism in conventional general-purpose operating systems like UNIX, which emphasizes on fairness of distribution of CPU time (time-sharing) amongst all processes and not timeliness of delivery, cannot be used.

Yet, to naively apply the techniques used in hard real-time systems (where missing a deadline is considered fatal) to schedule multimedia applications cannot produce the desirable results. This is because while it may be possible to determine a priori the resource requirements of a multimedia application, such a priori determination results in an overly conversative schedule that does not take into consideration the the characteristics and demands of multimedia applications. Multimedia applications typically exhibit the following characteristics:

- Variable rate of execution. While most multimedia applications execute periodically, the execution rate does not need to strictly follow a fixed rate. For example, a video-on-demand application may have an optimal rate of 20 frames per second (fps). But if the system is overloaded, the execution may, for short spells of time, degrade to 10 fps without significant loss of quality.

- Non-static computation time. The execution time for a task may fluctuate depending on the data it is working on. Such cases happen due to the dynamic nature of multimedia data. In MPEG video data, frames can be encoded as either I-frames (intra-coded frame), P-frames (predictive-coded frame) or B-frames (bi-directionally predictive-coded frame). I-frames are coded without any reference to other images. P and B-frames requires information of the previous and/or future I/P-frames As a result, decoding I-frames require much more computation time as compared to decoding P or B-frames. This results in non-static computation time.

- Incomplete execution. Quite often, the quality of a video sequence is gradually improved through many phases of refinement. If there is time to spare, a frame can undergo several iterations to produce a high quality output. But if there is insufficient time, then a less-than-perfect output is achieved. This compromise in execution will produce a video sequence with a mix of high and low quality frames, something certainly more desirable than expecting high quality frames all the time, causing the task to miss all the deadlines in the process.

- Relaxed deadlines. If a multimedia task completes execution after its deadline, while the *value* obtained may not be as high as if the task is completed by the deadline, the outcome should not be totally abandoned. This is because most multimedia tasks do have a level of tolerance due to the variations in the level of human sensory perception. For example, in the case of an audio and video playback application, the synchronization points of the two streams are loosely bounded up to approximately 250ms [7]. Under such circumstances, even if one of the media stream is late at output, the jitter may not be easily detected by the user.

In this paper, we first propose a model that depicts these characteristics of multimedia applications. An approximating scheduling algorithm is then presented that can be used to schedule multimedia tasks effectively on a single processor.

2. Related Works

Much work has been done in the area of real-time scheduling algorithms. The Earliest Deadline (ED) algorithm [14] is an optimal dynamic scheduling algorithm that is able to schedule all task correctly provided the system is not overloaded. However, if the system was to be temporarily overloaded, due to a bigger demand, the ED algorithm fails miserably. Because of this, tasks having variable execution times may produce a transient overload which the system is not able to know in advance, thus causing undesirable execution.

Researchers have adopted different approaches to try to solve this problem. [19] models the varying computation times as a probability density function and then use existing scheduling algorithms (e.g. Rate Monotonic [14] with Sporadic Server [17]) to provide statistical guarantees on the execution. [16] models the computation times instead with a fixed repeating sequence and establishes a sufficient condition for optimal execution if the sequence is *accumulative monotonic*[1]. Besides trying to model the

[1][16] Given a task $T = ((c_1, c_2, \ldots, c_n), p)$, where c_i denotes the computation time and p denotes the period of the task, and c_1 is the largest. $\forall i, 1 \leq i \leq n, \forall k, 1 \leq k \leq n-i$,

$$\sum_{j=1}^{k+1} c_j \geq \sum_{j=i}^{k+i} c_j$$

execution times, others have concentrated on coping with overload conditions instead, and thus indirectly address the problem of variable execution times.

Many heuristics have been proposed for best-effort delivery under overload conditions [6, 9, 11]. They usually involve using an optimal scheduling algorithm under normal conditions, and then switching to a best-effort algorithm when the demand exceeds a certain threshold. A significant break-through in the study of overload scheduling is the proof by [2] that no on-line algorithm is able to guarantee a value of greater than $\frac{1}{4}th$ the value obtained by a clairvoyant[2] scheduler.

On the other hand, the approach taken by multimedia researchers is empirical and involves observing the performance of a particular multimedia application. In [4], an existing video-conferencing application is examined and modifications are proposed to its video segment to support CPU load adaptation. The mechanism involves a *Load Agent* which monitors the Inter-frame Display Time (IDT) and induces the application to discard frames if the variance of the IDT increases over a threshold. [8] implemented a programming model based on Rate-based Execution (RBE). The basis is a resource-allocation model which guarantees that an application will progress at at least its specified rate. When resources are scarce, a negotiation will take place to reduce the rate of progress. In similar fashion, the RT-Mach research for multimedia support [12] utilizes a reservation system for processor capacity. In overloaded conditions, the application can choose to lower the rate of execution or reduce the quantum of processor usage at every period. Experiments conducted in [5] on a video-playback application suggest that simply decreasing the desired frame-rate does not always guarantee a better quality result. These observations point to the fact that some systematic way of managing overload is necessary to ensure execution is maintained at the best possible level. In our work, the user is asked to specify a compromise criterion. Our scheduler will then regulate the overload condition based on the specified criterion. The specification of the compromise criterion is based on the concepts of mandatory and optional sub-tasks defined in the imprecise computation technique [13, 3]. Every task is divided into two parts, a mandatory sub-task which must be executed and will produce an acceptable result as well as an optional sub-task which will serve to refine and improve the result. The optional sub-task is assumed to be monotonically increasing, implying that more time spent on executing will guarantee a better result. Thus, in overloaded conditions, the optional sub-tasks are those that can be compromised to allow the mandatory sub-tasks to be completed.

---

[2] knowing the parameters of future events

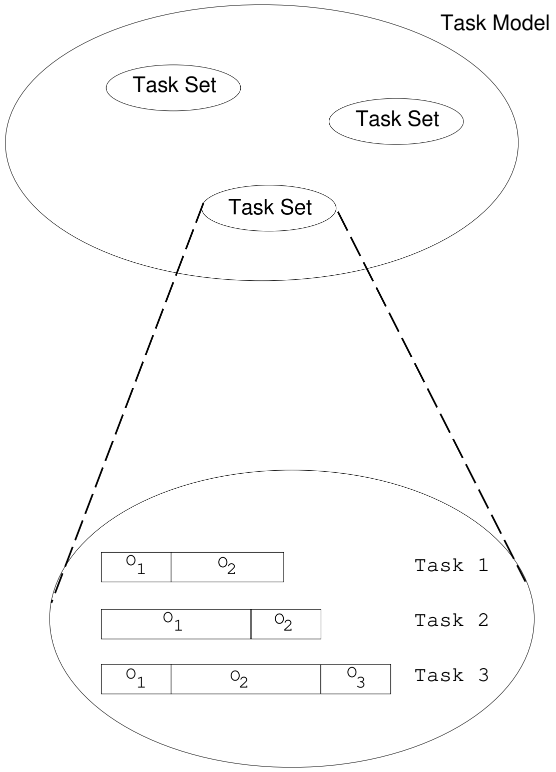

Figure 1. The Task Model

3. Multimedia Task Model

In the section, we propose a model for representing multimedia tasks. This model can then be used to perform the scheduling of tasks. Our task model is hierarchical in nature. At the top, the task model consists of a few task sets. Each task set is simply a collection of several independent tasks, competing with each other for processing time. Each task is, in turn, a sequence of operations. This is akin to a computer system (task model) having a few executing multimedia applications, (task sets) containing several media streams (tasks), and each stream executing its own subroutines (e.g. 3 operations, input $\rightarrow$ processing $\rightarrow$ output) (see Figure 1). Each task, T_i, has a start time, a period and a corresponding deadline for each release of the task at each period. Formally, the task set is defined as $\tau = \{T_1, T_2, \ldots, T_n\}$ where n is the total number of tasks and $T_i = (O_i, r_i, p_i, d_i)$ where $O_i = [O_i^1, O_i^2, \ldots, O_i^m]$, m is $|O_i|$, r_i is the

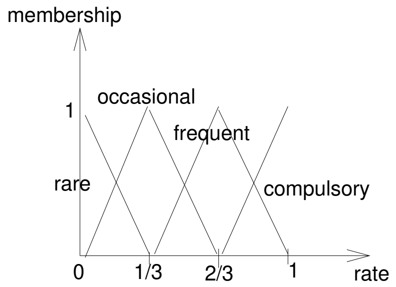

Figure 2. The Fuzzy Concepts Corresponding to the Linguistic Values of Rate_i^k

ready time or start time, p_i is the period and d_i is the relative deadline. Thus task T_i is released at times $r_i, r_i + p_i, r_i + 2p_i, \ldots$ and has deadlines at $r_i + d_i, r_i + p_i + d_i, r_i + 2p_i + d_i, \ldots$. The operation O_i^k is a 2-tuple $(oc_i^k, \mathrm{Rate}_i^k)$, where oc_i^k represents the computation time of the operation and Rate_i^k is the *desired rate* at which the particular operation is to run under normal load condition. Rate_i^k is a linguistic variable [10] with the values of "compulsory", "frequent", "occasional" and "rare". The universe of discourse is the interval $[0, 1]$. The membership curves of the linguistic values are shown in Figure 2. In addition, we also define c_i as the total computation time of task T_i, and $c_i = \sum_{j=1}^{m} oc_i^j$.

A point to note is that in a sequence of operations $O_i = [O_i^1, O_i^2, \ldots, O_i^m]$ in task T_i, the execution of operation O_i^k is not a pre-requisite for the execution of O_i^{k+1}. When scheduling T_i, the scheduling algorithm may, after the execution of O_i^{k-1}, discard O_i^k and immediately schedule O_i^{k+1}. In practical situations, the task T_i can be a video stream and the operation O_i^k is a dithering subroutine which improves the quality of the images. Under overloaded conditions, we may want to sacrifice the quality of the image (by discarding O_i^k) in exchange for on-time delivery of the video stream. This tradeoff between quality and time can be expressed using a set of rules. Assume Load denotes the system load at a particular instant of time, Rate is the user-specified desired rate for an operation O, and Z is the target rate for O. The set of rules are:

Rule1	If Load = heavy, Rate = compulsory then Z = compulsory
Rule2	If Load = normal, Rate = compulsory then Z = compulsory
Rule3	If Load = light, Rate = compulsory then Z = compulsory
Rule4	If Load = heavy, Rate = frequent then Z = occasional
Rule5	If Load = normal, Rate = frequent then Z = frequent
Rule6	If Load = light, Rate = frequent then Z = frequent
Rule7	If Load = heavy, Rate = occasional then Z = rare
Rule8	If Load = normal, Rate = occasional then Z = occasional
Rule9	If Load = light, Rate = occasional then Z = occasional

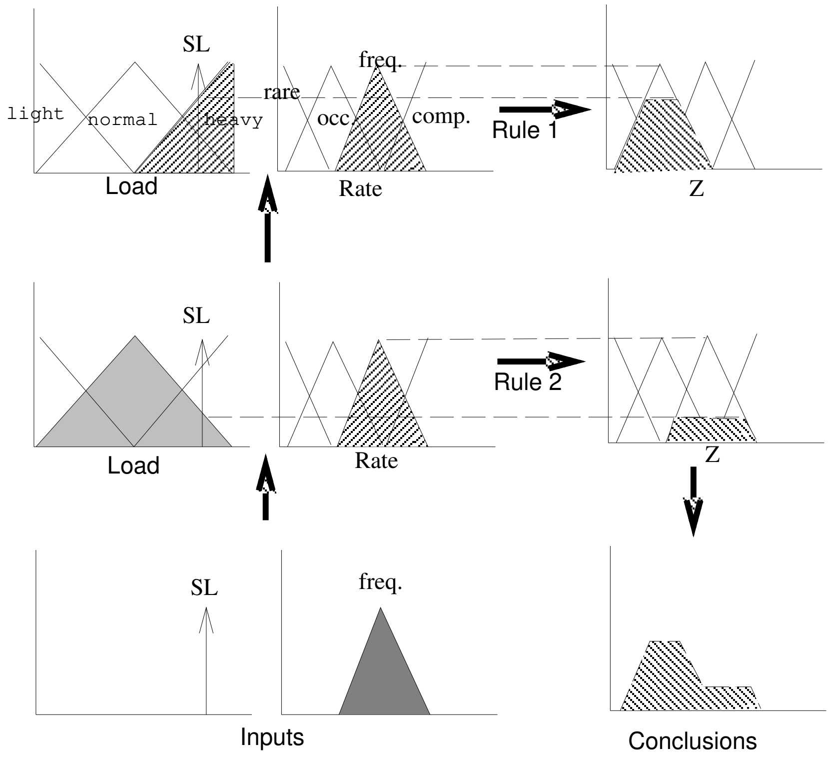

Figure 3. The Reasoning Process

Rule10	If Load = heavy, Rate = rare then Z = very rare
Rule11	If Load = normal, Rate = rare then Z = rare
Rule12	If Load = light, Rate = rare then Z = rare

To obtain the value of Z, we first compute the degree of match between our inputs and the antecedent of each if-then rule. The inputs are the load of the system at the given instant of time and the user-specified desired rate of the given operation. The calculation of the degree of match is based on standard fuzzy intersection. In the example (see Figure 3), the system load partially matches the antecedent of Rule 1 and Rule 2 while the user-specified desired rate for the operation is frequent. Hence, each rule's conclusion is truncated by the degree of match between the antecedent of the rule and the given system load. Finally, we take the union of the truncated sets to arrive at a conclusion (see Figure 3). This conclusion is a fuzzy set. To obtain a numerical value, we perform defuzzification of the conclusion using the center of maxima method [10]. The defuzzified value of Z, a numerical value between 0 and 1, is denoted by f. We call f the theoretical desired frequency of the operation. Suppose the value of f is $1/n$, then this implies that for every n number of operations, the user is satisfied with the quality obtained if only one out of the n is being executed. In the next section, we will describe a scheduling algorithm that makes use of f to dynamically adjust the performance of a multimedia application. Before that, it is useful to note that by allowing the user to specify different desired frequency rates at the operation level, we give the user greater control over the handling of different video frame types. For example, we may define one operation to handle the keyframes (I-frames), another operation to handle the P-frames and a third operation to handle the B-frames. Since dropping an I-frame degrades the quality of the video more as compared to dropping a P or B-frame, we specify a higher frequency for the operation handling the I-frame, while the operations handling P- or B-frame are given lower frequency values. With this, we can have a finer control over the handling of different video frame types thus allowing the quality of service be maintained to a satisfactory degree even in the overload conditions. Similarly, for audio samples, we can define one operation to handle the audio stream. A higher frequency is assigned to the audio operation over other operations. In so doing, we can prevent too many of the audio samples from being dropped which may degrades the performance substantially.

4. Approximating Scheduling Algorithm

A two-levels scheduling scheme is proposed to adapt to overload conditions. Figure 1 shows the organization of the scheduling scheme. The first level is a frequency filter (FF) agent which uses the f obtained from the previous section to decide whether a particular operation should be discarded or admitted into the system. If the operation is to be admitted, it is then sent to the second-level scheduler for further scheduling, else it is returned to the task to invoke the next operation.

Two strategies have been implemented to perform the filtering process:

- **Cumulative filtering**. This algorithm maintains a small counter for each operation. When the operation is invoked, f is added to the counter. If the value of the counter is more than or equal to 1, the operation passes through; otherwise, the operation is discarded. The algorithm depends on the feedback mechanism to find out if the operation is successfully executed. If so, the counter is decreased by 1. At the initialization phase, each counter starts with a randomly generated number between 0 and 1 so as to spread out the distribution.

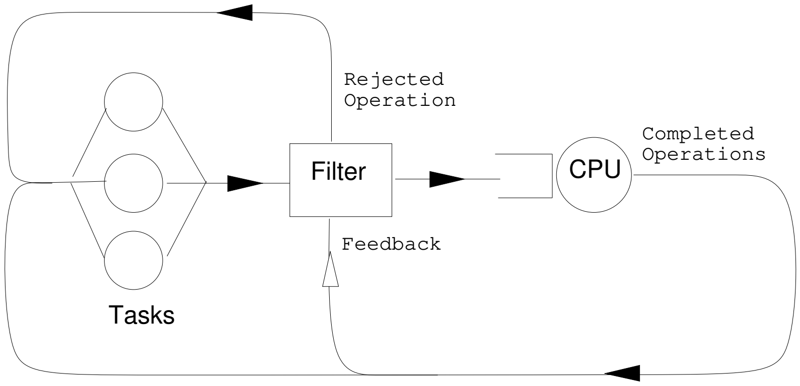

Figure 1. Role of the Frequency Filter Agent

This is to ensure that not all operations get accepted or rejected at the same time. This algorithm runs in $O(1)$ time.

- **Probabilistic filtering**. This algorithm relies on a random number generator to come up with a value between 0 and 1. Upon the arrival of an operation, f is compared with the random value. If f is equal or larger, the operation passes through; otherwise, it is discarded. The speed of this algorithm depends solely on the speed of the generator, which in this case, can be as simple as taking the nanosecond reading of the system clock, implying also a running time of $O(1)$.

With the FF agent, we effectively prevent the less important operations (lower frequencies) from entering the second-level scheduler if the system load is already high, thus avoiding overload conditions. As a result, many existing scheduling algorithms that have been proven to work well under light-normal conditions can be utilized at this second-level scheduler. Among the more well-known scheduling algorithms are:

1. **Earliest Deadline First (EDF)**. The tasks waiting for the CPU will be sorted in ascending order of their deadlines. The task with the earliest deadline will be executed first. The EDF algorithm has been proven to be optimal as long as utilization of the CPU is less than 1 [14].

2. **Least Slack (LS)**. This is also an optimal dynamic algorithm [15] for preemptive systems. It is based on the urgency of the tasks where the more urgent tasks will be scheduled first. Slack time is the "spare" time which a task can spend waiting for execution before it must be scheduled or miss the deadline.

Parameter	Value
Ready time of task	Exponential(20)
No. of operations / task	Uniform(2,5)
Compute time / operation	Exponential(5)
Overhead / operation	0.001
Period of task	Uniform(c_i,c_i*30)
Frequency	$\frac{\text{Random}(1,10)}{10}$

Table 1. Parameters for Soft Real-time Tasks

3. **Shortest Remaining Processing Time First (SRPT)**. This algorithm basically assigns the task with the shortest remaining processing time to execute first. This is adapted from the *shortest processing time* algorithm used in job-shop scheduling to minimize average lateness [1]. In this algorithm, tasks which can complete the job sooner are given higher priorities and tasks with extremely long execution times are usually sacrificed.

4. **First Come First Served (FCFS)**. This is the simplistic approach where the task that comes first will get to execute first. There is really no advantage in using this algorithm except that it does not need or depend on any information on the tasks it is scheduling.

We will test out the effectiveness of these two strategies in the experiment conducted below. The experiment is conducted on a Silicon Graphics Indy workstation running IRIX 5.3. A set of tasks is simulated with the following parameter values as shown in Table 4.

We run this set of tasks using the following scheduling strategies:

1. **With cumulative filtering**. This algorithm is used with Earliest Deadline First (EDF-CF), Least Slack (LS-CF), First Come First Served (FCFS-CF) and Shortest Remaining Processing Time (SRPT-CF).

2. **With probabilistic filtering**. This algorithm is also used with Earliest Deadline First (EDF-PF), Least Slack (LS-PF), First Come First Served (FCFS-PF) and Shortest Remaining Processing Time (SRPT-PF).

To evaluate the goodness of the proposed algorithm, we first distinguish between the theoretical desired frequency (f) versus the actual frequency (A) of an operation. The theoretical desired frequency of an operation is derived based on the system load and the user's specification as discussed in the previous section. However, in an actual implementation, this theoretical frequency usually differs from the actual frequency in which an operation is being invoked. We define the actual frequency (A) of an operation as the ratio of the number of times the operation is being executed to the total number of arrivals of the operation. Two measures are derived for evaluating the goodness of the proposed algorithm:

- **Operation Variance (v_o).** After every run, the individual variance of each of the operation within each task is computed by getting the square of the difference between the desired frequency (f) and the actual frequency (A), if the actual frequency is lower, and 0 if the actual frequency is equal or higher. v_o is obtained by getting the mean of all the individual variances. Formally, for each operation, individual frequency difference, FD, is defined as

  $$FD = \begin{cases} (f - A)^2 & (f > A) \\ 0 & (f \leq A) \end{cases}$$

  Operation Variance,

  $$v_o = \sum \frac{\text{individual frequency difference}}{\text{no. of runs * no. of operations}}$$

  The rational for this measure is that as long as an operation runs above the theoretical desired frequency, there is no penalty involved since the quality level is within what the user is willing to tolerate. Penalty is only incurred when $A < f$. Hence, with this definition, the larger the variance is, the greater is the amount of penalty. Therefore, this measure gives a good indication of the quality tradeoff incurred by different scheduling algorithms.

- **Task Variance (v_t).** After every run, for each task, the individual task variance is computed as follows:
  For each task, individual task variance

  $$= (\sum \frac{\text{individual frequency difference}}{\text{no. of operations}})^2$$

  Task Variance,

  $$v_t = \sum \frac{\text{individual task variance}}{\text{no. of runs * no. of tasks}}$$

The task variance measure is the normalized aggregate of individual operation variance. It is used to determine the total amount of penalty incurred per task. Ideally, we would like to keep this measure as small as possible.

We performed 50 runs for the experiments. The average results are presented below. In the graphs, the SRPT-CF and SRPT-PF algorithms

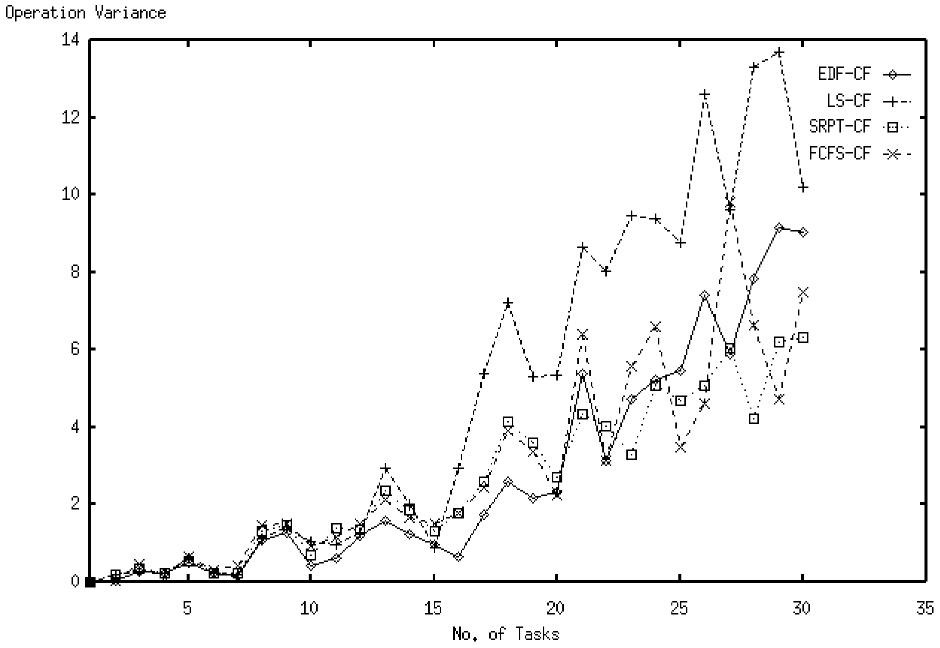

Figure 2. Graph of Operation Variance under Cumulative Filtering

are consistent in obtaining good results.

Next, we compare the SRPT-CF and SRPT-PF's performance with respect to the unfiltered algorithms (EDF, LS, FCFS, SRPT). In Figures 6 and 7, we see that a suitable first-level filter is able to help obtain the best performance while a poorly designed one can lead to undesirable results.

5. Conclusion

In this paper, we have proposed an approach to model multimedia applications to handle variable and overloaded situations. This is because in such situations, all the applications cannot be guaranteed to execute within their deadlines and hence, some form of degradation is necessary. Users are able to influence the schedule by specifying a frequency parameter associated with each operation of the task (or application). This parameter is a linguistic variable and it represents the *minimum desired rate* of the operation which the scheduling algorithm is expected to adhere to and hence achieve higher predictability in execution. A two-level scheduling strategy

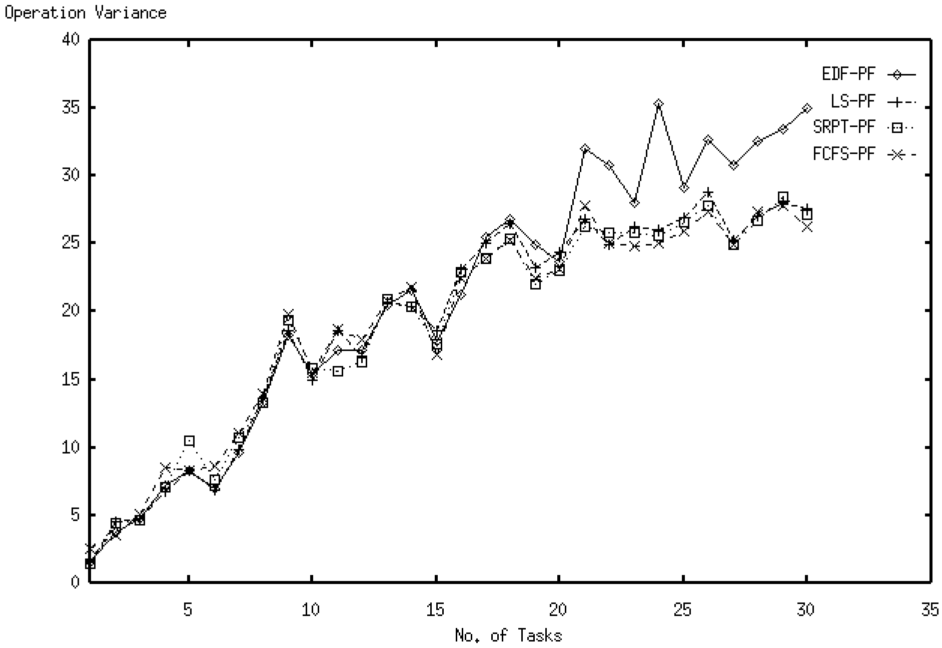

Figure 3. Graph of Operation Variance under Probabilistic Filtering

is proposed to incorporate the concept of *minimum desired rate* into the scheduling process. The simulation results indicate that the introduction of a simple frequency filtering agent, when used in conjunction with existing scheduling algorithms, can improve the schedule significantly under overloaded conditions. Further work is in progress to analyze the impact of the filtering agent so as to achieve better and more stable results.

References

1. K. R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, 1974.
2. S. Baruah, G. Koren, B. Mishra, D. Mao, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. In R. Werner, editor, *Proceedings of the Real-Time Systems Symposium - 1991*, pages 106–115. IEEE Computer Society Press, December 1991.

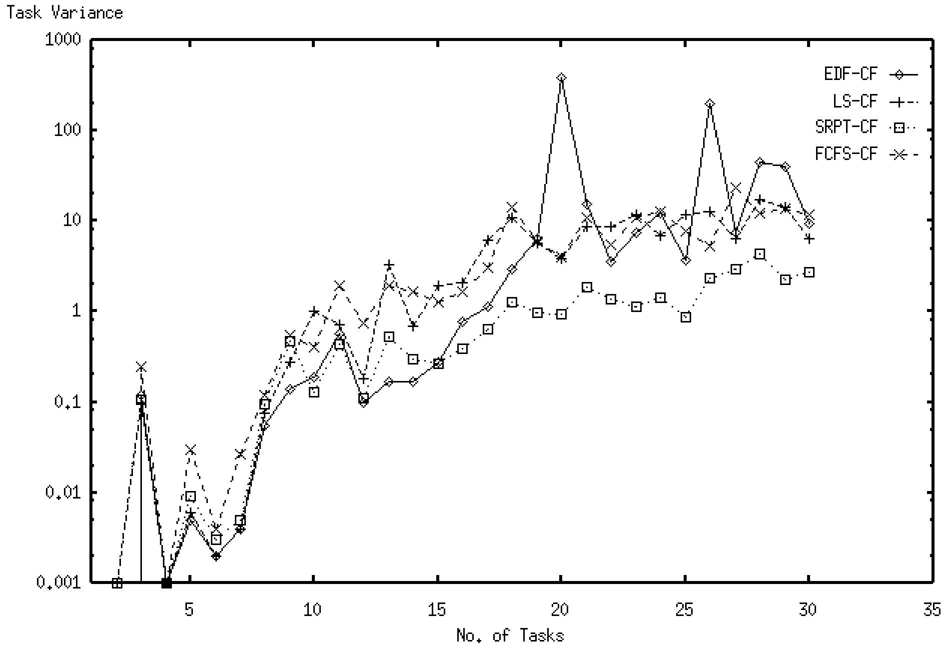

Figure 4. Graph of Task Variance under Cumulative Filtering

3. Jen-Yao Chung. Scheduling hard real-time jobs that allow imprecise results. Technical Report 1519, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1989.

4. Kevin Fall, Joseph Pasquale, and Steven McCanne. Workstation video playback performance with competitive process load. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1995.

5. Richard Gerber and Ladan Gharai. Experiments with digital video playback. Technical Report UMIACS TR 95-103, Department of Computer Science, University of Maryland, October 1995.

6. J. R. Haritsa, M. Livny, and M. J. Carey. Earliest deadline scheduling for real-time database systems. In R. Werner, editor, *Proceedings of the Real-Time Systems Symposium - 1991*, pages 232–243. IEEE Computer Society Press, December 1991.

7. B. B. Hehmann, M. G. Salmony, and H. J. Stuttgen. Transport services for multimedia applications on broadband networks. *Computer Communications*, 13(4):197–203, May 1990.

8. K. Jeffay and D. Bennett. A rate-based execution abstraction for multimedia computing. *Lecture Notes in Computer Science*, 1018:64–78, 1995.

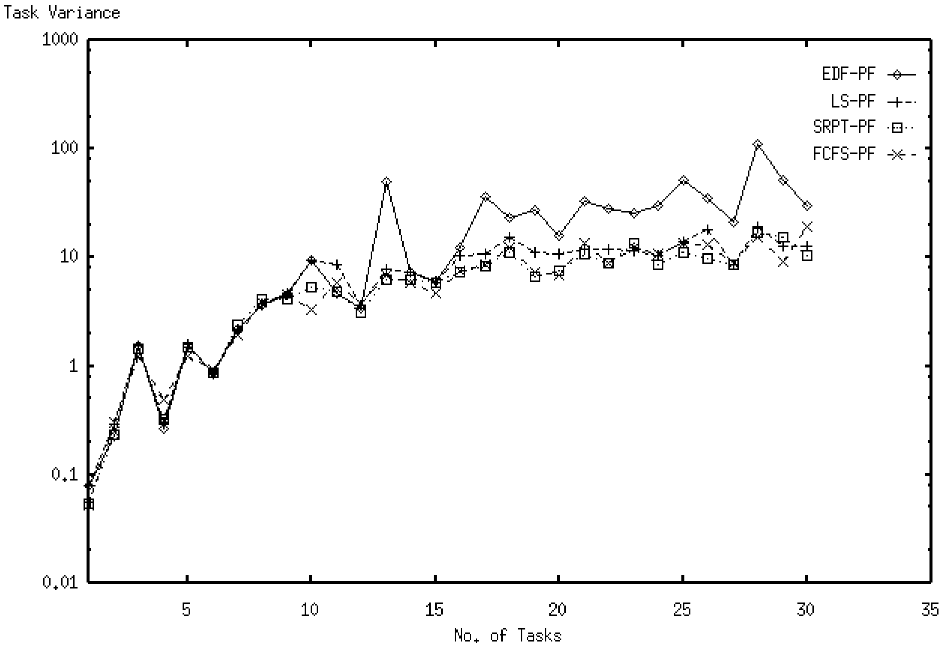

Figure 5. Graph of Task Variance under Probabilistic Filtering

9. H. Kaneko and J. Stankovic. A multimedia server on the spring real-time system. Technical Report UMass CS 96-11, Department of Computer Science, University of Massachusetts, Amherst, 1996.

10. G. J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall, 1995.

11. G. Koren, B. Mishra, A. Raghunathan, and D. Shasha. On-line schedulers for overloaded real-time systems. Technical Report TR1991-558-R246, New York University, May 1991.

12. C. Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic qos in real-time mach. *In the proceedings of Multimedia Japan 96*, April 1996.

13. K. J. Lin, S. Natarajan, and J. W. S. Liu. Scheduling real-time, periodic jobs using imprecise results. *Proc. IEEE Real-Time System Symp.*, 1987.

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

15. A. K. Mok and M. L. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proc. 7th IEEE Texas Conf. Comput. Symp.*, pages 5–1–5–12, November 1978.

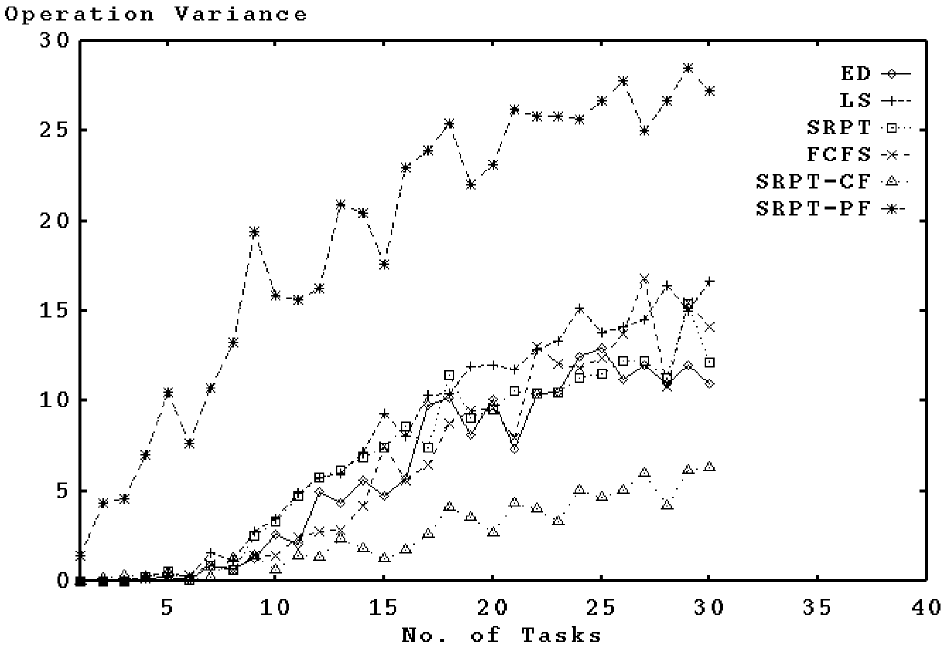

Figure 6. Graph of Operation Variance

16. Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. Technical Report CS-TR-96-07, Department of Computer Science, University of Texas, Austin, 1996.

17. B. Sprunt and L. Sha. Scheduling sporadic and aperiodic events in a hard real-time system. Technical Report CMU/SEI-89-TR-11 ADA211344, Software Engineering Institute (Carnegie Mellon University), 1989.

18. Peter Staunton. Tenet suite 1 and the continuous media toolkit. Technical Report TR-95-028, International Computer Science Institute, Berkeley, CA, June 1995.

19. T.-S. Tia, Z. Deng, M. Shankar, M. Storch, L.-C. Wu J. Sun, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 164–173. IEEE, May 1995.

20. Hui. Zhang and Domenico. Ferrari. Rate-controlled static priority queueing. Technical report, January 1992.

21. J. F. Gibbon and T.D.C. Little. Real-Time Data Delivery for Multimedia Networks. *Proc. 18th Annual Conference on Local Computer Networks,* Minneapolis, MN, September 1993, pp. 7-16.

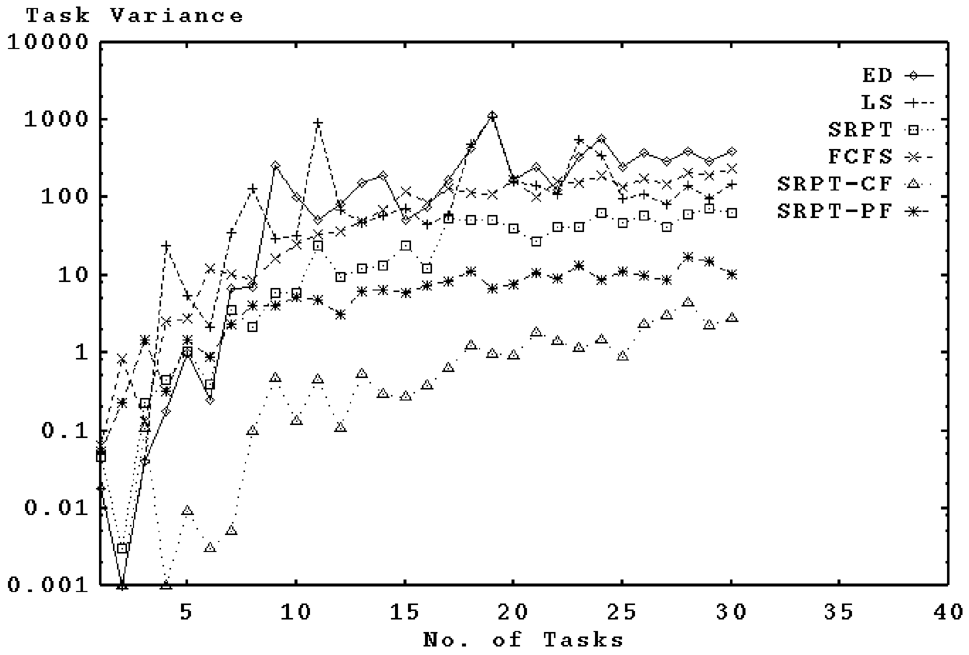

Figure 7. Graph of Task Variance