SCHEDULING HARD REAL-TIME MULTI-MEDIA DISK TRAFFIC

K. Tindell[1], A. Burns
Real-Time Systems Research Group,
Department of Computer Science,
University of York, England

ABSTRACT

In this report we show how existing real-time scheduling theory, developed to analyse the scheduling of processors and network communications, can also be applied to the problem of guaranteeing the performance of multi-media information streams read from a disk drive. We develop simple analysis that is independent of disk layout information; a small example is analysed, using a simple tool embodying the analysis. The analysis is then extended to take account of disk layout information; the example is then re-analysed using this less pessimistic model. One of the advantages of the proposed approach is that it gives a precise means of determining the impact of buffer size on disk utilisation.

1. INTRODUCTION

The technological problems associated with multi-media computing has received much attention recently. One of the critical difficulties in multi-media is the real-time scheduling problem of ensuring that multiple multi-media data streams from a disk drive (or a collection of disk drives) are processed within deadlines at a sufficiently high rate (the quality of service of a stream dictates the data rate, and size of the buffer in the computer dictates the deadline for the data).

Co-incidentally, much attention has also been focused on the problem of scheduling multiple tasks on a processor (for example, the work of Liu and Layland [5]). Recently, Tindell *et al* [11] have developed analysis for fixed priority pre-emptive scheduling that permits tight bounds on the longest time taken to satisfy certain resource requirements (for example, completing an amount of computation time, or transmitting a message on a shared communications bus).

In this paper we take the analysis of Tindell and apply it to the problem of scheduling disk traffic in a multi-media context.

---

[1]This author can be contacted via e-mail as ken@minster.york.ac.uk

2. COMPUTATIONAL MODEL

We assume that a multi-media system comprises of a main processor (for example, a workstation) connected to a disk drive that stores multi-media data in the form of blocks on the disk. The disk drive may consists internally of multiple disks, where blocks are *striped* (for bandwidth reasons [7]). A *stream* consists of a set of requests for disk blocks that are to be read[2] from the disk and processed by the main processor (for example, decompressed and played as video on the screen). A given stream i is characterised by the following:

R_i	The number of disk blocks that are requested together
T_i	The lower bound on the time between successive disk requests

Thus, in the worst-case, every time T_i the processor requests R_i disk blocks to be read.

We assign a unique static priority to each stream such that requests for disk blocks are placed in a single queue ordered by this priority. When processing these requests, the disk scheduler examines the head of the queue, and fetches up to the first n_i disk blocks required (where $n_i \leq R_i$). Once R_i blocks have been fetched, for a given request of stream i, the request is satisfied and removed from the queue. The queuing of requests are *pre-emptive* such that the arrival of a higher priority request is queued ahead of lower priority requests.

Multi-media is stored on the disk in such a manner that reading successive x disk blocks from the stream i results in a worst-case number of track-to-track seeks of $t_i(x)$. The worst-case track-to-adjacent-track seek time is denoted s. The time S represents the worst-case time to move the disk drive read head from any one track on the disk to any other track on the disk. This time S is analogous to the context switch time of a task in a multi-tasking operating system. Lougher and Shepherd assert that current real-time work assumes that these costs are nominal; the work of Burns and Wellings [3], among others, realise that context switch costs can be large, and derive specific analysis to bound these overheads.

Disk blocks corresponding to a stream i, read from the disk drive, are stored in a buffer space allocated to the stream, before being retrieved by the main processor. No flow control is implemented between the main processor and the disk drive: data is read out of the buffer by the main processor as fast as it is written into the buffer by the disk drive; the buffer is not permitted to overflow or underflow. This is ensured by the timing characteristics of a stream: before a stream is activated the buffer space allocated to the stream is pre-loaded from the disk. Thus, as long as the worst-case time to satisfy a request is less than the time taken to empty the buffer space allocated to the stream, and as long as the data is removed at the same rate as being added, then the buffer will never

[2]Following the approach of Lougher and Shepherd we consider only the reading of disk blocks, since the writing of blocks is conceptually the same problem.

overflow or empty [7]. This means that access to the buffer is non-blocking. When a given stream is opened it is allocated a defined buffer space, which is pre-loaded with stream data before the stream is played back.

The following section derives mathematical analysis bounding the timing and buffer requirements of the system. The rôle of the analysis is, as stated by Lougher and Shepherd [7], to provide a test that is able to determine whether a new stream can be accepted or not. Also, the analysis is able to provide metrics by which disk layout, stream request size, stream buffer size, *etc* can be evaluated.

3. ANALYSIS OF THE SYSTEM

The deadline for a request of a given stream may be longer than the time between stream requests, and thus neither the rate monotonic nor the deadline monotonic approaches are appropriate. Recently, Tindell *et al* [11] formulated analysis for a real-time processor with multiple tasks, permitting tasks with arbitrary deadlines to be analysed. The following equation, from Tindell *et al*, gives the worst-case response time of a task i, scheduled on a processor by fixed priority pre-emptive despatching:

$$r_i = \max_{q=0,1,2,3,\cdots} \left(w_i - qT_i \right) \tag{1}$$

where w_i is given by:

$$w_i = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i}{T_j} \right\rceil C_j \tag{2}$$

where C_i is the worst-case computation time requirement of task i, T_i is the period of task i (*i.e.* the lower-bound on the time between successive releases of the task), $hp(i)$ is the set of tasks of higher priority than task i, and B_i is the worst-case time task i can be deferred from executing by a lower priority task holding a resource needed by task i. Resources are modelled as critical sections guarded by semaphores.

In the above equation, the term:

$$\left\lceil \frac{w_i}{T_j} \right\rceil C_j \tag{3}$$

is the *computational interference* from a task j on a lower priority task i, and represents the worst-case time that a task j can pre-empt and prevent task i from running. The worst-case computation time C_j is defined to include the costs of two context switches: one to take the processor to a state where it can execute task j, and one to take it back to a state where it can resume the execution of task i.

In equation 2 the term:

$$(q+1)C_i \tag{4}$$

represents the total amount of processing of priority i that needs to be done within the time w_i (this time w_i represents a so-called *level i busy perio*d [6,11]). Within w_i no work of lower priority than task i can take place after task i has commenced execution.

Equation 1 specifies an infinite sequence of values of q; however, the sequence can be terminated when the following condition holds:

$$w_i < (q+1)T_i \tag{5}$$

Furthermore, equation 2 has w_i appearing on the left and right hand sides. To solve this a recurrence relation is formed:

$$w_i^{n+1} = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \tag{6}$$

where $w_i^0 = 0$, and where the equation has converged when $w_i^{n+1} = w_i^n$. The equation is guaranteed if the utilisation is less than or equal to 100%, *i.e.*:

$$\sum_{\forall i \in tasks} \frac{C_i}{T_i} \leq 1$$

where *tasks* is the set of all tasks in the system. Note that equation 6 is pseudo-polynomial. Generally, it is quick to evaluate, but when utilisation approaches 100% the equation takes a time proportional to the least common multiple of the task periods. The least common multiple of the task periods can (if the periods are prime numbers) be equal to the multiple of all the task periods, and thus the analysis can take a considerable time to evaluate.

The analysis permits tasks to lock and unlock semaphores according to the priority ceiling protocol [8], whereby a task i can be blocked at most once by a lower priority task, and thus B_i can be computed.

We can apply this task scheduling analysis to the problem of scheduling disk drive traffic. Equation 4 gives the computation time from a given task i that contributes towards computing the level i busy period of width w_i (equation 2). To derive analysis for the disk scheduling problem we need to find the time taken to process $q + 1$ stream i disk requests (not including the interference from higher priority streams, or blocking from lower priority streams). To do this we use the seek time function, $t_i(x)$. This indicates the total time the disk spends moving heads to read any given x sequential blocks of stream i

from the disk (assuming that no other higher priority stream pre-empts). So, for example, if x is less than the number of blocks per track, and all the blocks of stream i are stored contiguously, then $t_i(x)$ is equal to the worst-case track-to-track seek time (including any rotational latency[3]). Thus the placement of blocks on the disk is important, and the timing ramifications of any given layout can be taken into account by this analysis. The time taken to read a disk block is defined as b, and hence the total time, in the absence of interference and blocking from other streams, to read x blocks is:

$$C_i(x) = t_i(x) + xb \tag{7}$$

We now address the costs of blocking of a request from stream i. When a request of a given stream i is queued it cannot be processed immediately, since a request from a lower priority stream k may have just attained access to the disk drive. Thus, stream k behaves as if it were a task locking a semaphore guarding access to the disk drive. A stream k holds the semaphore for at most time:

$$S_{\max} + C_k(n_k) \tag{8}$$

since up to n_k blocks can be processed by the disk drive before it is made available to a higher priority streams. $S_{\max}$ is the worst-case time taken to move the disk head from any position on the disk to anywhere else on the disk. The value of $S_{\max}$ is large compared to other parameters. In section 5 we show how knowledge of disk layout can be used to give more accurate values of S. It takes a further bounded time $S_{\max}$ to position the disk drive head to be in a position to read blocks to satisfy the request of stream i.

By applying the priority ceiling protocol analysis [8], we can say that the blocking factor of a given stream i, B_i, is thus given by:

$$B_i = \max_{\forall k \in lp(i)} \left(S_{\max} + C_k(n_k) + S_{\max} \right) \tag{9}$$

Where $lp(i)$ is the set of streams of lower priority than i. Note that we may wish to create a server stream with lowest priority to serve disk requests from non-real-time applications (*e.g.* general file access); in this report we assume that there is an idle stream (*i.e.* a stream which makes no disk requests, but may have left the read head in any position on the disk) of lower priority than all other streams. This would give a blocking time for all multimedia streams of at least $S_{\max}$.

The computational interference of a task (equation 3) can be used to determine the interference a higher priority stream j causes on a lower priority task i:

[3]By 'skewing' data on the disk the impact of rotational latency can be reduced.

$$I_{i,j} = \left\lceil \frac{w_i}{T_j} \right\rceil \left(S_{\max} + C_j\left(R_j\right) + S_{\max} \right) \tag{10}$$

Notice how the costs of moving the disk head from reading stream k to stream i, and back again are included — pre-emption behaves very much like an interrupt in a processor: the old context is saved, the interrupt processing done, and then the old state restored. Here 'state' is the position of the disk head.

The work of Burns *et al* [2] shows how using non-pre-emptable sections of code can lead to increased schedulability. Following this work we determine the largest number of blocks read in the final non-preemptable section of servicing a request of a given stream i. This last batch of blocks is of size:

$$l_i = R_i - \left(\left\lceil \frac{R_i}{n_i} \right\rceil - 1 \right) n_i \tag{11}$$

We will complete the last batch of block reads without being pre-empted, and thus the time taken to read these blocks can be added on to the time computed using w_i thus:

$$r_i = \max_{q=0,1,2,3,\cdots} \left(w_i - qT_i \right) + C_i\left(l_i\right) \tag{12}$$

where $C_i\left(l_i\right)$ is the time taken to read the l_i disk blocks (equation 7), and where w_i is given by:

$$w_i = B_i + C_i\left(qR_i + R_i - l_i\right) + \sum_{\forall j \in hp(i)} I_{i,j} \tag{13}$$

Note that to evaluate w_i we form a recurrence relation, as for equation 2, we must modify the recurrence thus:

$$I_{i,j} = \left\lceil \frac{w_i^{n+1} + \varepsilon}{T_j} \right\rceil \left(S_{\max} + C_j\left(R_j\right) + S_{\max} \right) \tag{14}$$

where ε is the granularity of time (thus if all times are whole numbers of milliseconds, ε is equal to 1ms). This is done to ensure that if the pre-emptable computation is zero (*i.e.* $l_i = R_i$), then equation 14 computes the correct time at which the non-pre-emptable computation starts.

From equation 12, the worst-case time between a request for R_i disk blocks is made and satisfied is r_i. Lougher and Shepherd indicate that the deadline of a given stream is the

time taken to empty the buffer space assigned to the stream, and thus the deadline of a given stream i, D_i, is given by:

$$D_i = \left\lfloor \frac{\beta_i}{R_i} \right\rfloor T_i \tag{15}$$

where β_i is the buffer space assigned to the stream i (note that a value of β_i not a multiple of R_i would be wasteful of buffer space, since data is removed from the buffer in units of size R_i).

The worst-case response time of a stream i, r_i can be trivially compared to the deadline, D_i, to see if the stream is schedulable:

$$r_i \leq D_i \tag{16}$$

And thus we have the basis of an on-line test, where we ask the question "can the new stream i be opened?". We assume a given buffer space requirement for the new stream, and thus obtain a deadline D_i. This deadline can then be used to obtain the priority of the new stream i. Finding a suitable value for n_i allows the evaluation of r_j for all streams j. If the condition in equation 16 holds for all streams then the new stream can be opened.

Generally the worst-case response time r_i will be considerably less than the deadline, and thus the buffer space required is generally lower. Given that a buffer for a stream i is pre-loaded, every time T_i the main processor removes R_i blocks from the buffer, and makes a request for another R_i blocks. Given that it may take up to time r_i to satisfy the request for R_i blocks, the number of blocks that may have been read by the main processor is given by:

$$\left\lceil \frac{r_i}{T_i} \right\rceil R_i \tag{17}$$

and thus this is the minimum amount of pre-loaded buffer space required by stream i.

Hence the minimum total buffer space required by all streams is given by:

$$\sum_{\forall i \in streams} \left\lceil \frac{r_i}{T_i} \right\rceil R_i \tag{18}$$

where *streams* is the set of multi-media streams being processed.

We could use equation 18 to reclaim surplus buffer space from existing streams — if there is insufficient space for a new stream then some of the buffer space can potentially be reclaimed from existing streams (in effect, shortening the 'pipeline' of data for a given

stream) and given to the new stream. However, if buffer space is taken from an existing stream i then stream i must be assigned a new deadline D_i commensurate with its new amount of buffer space (equation 15), which could entail changing the priority. Such a change requires further analysis, since priorities are assumed to be fixed. Note that with this approach there is no way of returning buffer space to a stream, since this would require additional data transfers to 'lengthen the pipeline' again. For simplicity, therefore, we adopt the approach of Lougher and Shepherd — buffer space is assigned statically — and therefore assume that a new stream cannot be added unless there is sufficient free buffer space.

The closure of a stream does not entail the instant removal of the effects of that stream. The buffer space of the stream can be re-claimed immediately, but the disk bandwidth cannot be reclaimed immediately. This is because the timing effects of the removed stream are not guaranteed to be over until an idle period in the disk drive occurs (the work of Tindell [12] shows how a naive approach to adding and removing real-time tasks — so-called *mode changes* — can lead to flawed analysis). Therefore we say that a stream is not completely removed until the disk drive scheduler idles the disk (or, at least, does not process multi-media disk traffic[4]).

As has been mentioned earlier, determining w_i (equations 2 and 6) can, in pathological cases, take a long time to compute. However, this rarely occurs, and hence we can in practice simply apply a limit to the amount of time we are prepared to spend before getting an answer.

Finally, we mention the configuration problem for the test. The analysis presented here has three main parameters: the non-preemptable section size of each stream i, n_i, the priority of the stream, and the size of each stream request, R_i. The choice of these for each stream is an optimisation exercise: for example, setting the value of n_i too small results in large overheads for stream i and streams of lower priority than i; setting the value of n_i too large results in a large blocking factor for streams of higher priority than i. Note that it is not possible to change the value of these parameters once a stream i has been accepted, since this would again entail a mode change. These parameters must be chosen not only to accept a given stream, but also to leave the system in a state such that future streams are likely to be accepted. This latter point is important, since it implies that to optimally choose the parameters requires a clairvoyant algorithm.

4. EXAMPLE MULTI-MEDIA PROBLEM

We now give a simple example which illustrates the use of the above analysis. The table below gives details of the set of multi-media files stored on a disk system:

[4]In general, multi-media data on the disk is shared with general purpose computing data, which must be processed in-between multi-media requests.

Filename	*Description*	*start block*	*size*
`sttng_1_vid`	TV quality video	998	108000
`sttng_1_aud`	TV quality sound	108998	4800
`rtstng_tut_vid`	Medium quality video	113798	36000
`rtstng_tut_aud`	TV quality sound	149798	2400
`cd_audio`	CD quality sound	152198	108000
`audio_1`	TV quality sound	312998	14400
`audio_2`	TV quality sound	260198	52800

Block sizes are assumed to be 1K bytes. We also assume that the files are stored on a disk with 606 tracks, each of 1098 blocks (making a total disk capacity of 665388 blocks, or approximately 650Mb). The track-to-track seek time is 350μs, and the time taken to read a block (including rotational latency) is 1100μs. We assume that the worst-case head movement time to access a block on the disk, S_{max}, is 212ms (*i.e.* 606 multiplied by 350μs). The worst-case rotational latency on the disk is 8.33ms (equal to a disk rotation speed of 7200 r.p.m.).

We wish to open the following streams in the given order:

file	*T*	*R*	*n*	β	*D*
`sttng_1_aud`	1.0s	4	4	52	13.0s
`sttng_1_vid`	0.3s	30	5	210	2.1s
`rtstng_tut_aud`	1.0s	4	4	52	13.0s
`rtstng_tut_vid`	0.1s	6	3	162	2.7s
`cd_sound`	0.5s	44	22	220	2.5s
`audio_1`	1.0s	4	4	52	13.0s
`audio_2`	1.0s	4	4	52	13.0s

Note that the period (*T*) and the number of blocks to read per period (*R*) effectively define the bandwidth required. We observe that by setting *T* to 1 second for all streams, choosing *R* correspondingly, and setting $n = R$, the analysis is equivalent to the 'round robin' approach of Lougher and Shepherd.

It is straightforward to embody the analysis developed so far in a small tool. We use this tool to illustrate the example, and start by trying to open the first stream:

```
command: open sttng_1_aud 52 1000000 4 4 25
```

This command requests that a stream corresponding to the file `sttng_1_aud` be opened with a buffer of 52 blocks, a period of 1 second, and where 4 disk blocks are read per request, and where the request cannot be pre-empted until 4 blocks have been read (*i.e.* the request is non-pre-emptable). The priority of the stream is 25 (where a smaller number indicates a higher priority). The stream can be opened successfully:

```
stream 0 response time would be 225183(13000000)
stream 0 opened
```

The stream response time is 0.225 seconds (*i.e.* a request for 4 disk blocks of this stream will be satisfied within 0.225 seconds); the deadline on reading the stream data is 13 seconds, and hence is schedulable. Thus we permit the stream to be opened.

We go on to open a video stream:

```
command: open sttng_1_vid 210 300000 30 5 100
stream 0 response time would be 451466(13000000)
stream 1 response time would be 699749(2100000)
stream 1 opened
```

This stream can be opened. Notice how the worst-case response time of stream 0 increases, since a request from this stream can be delayed (so-called blocking) by a request from the lower priority stream. We go on to try to open a third stream:

```
command: open rtstng_tut_aud 52 1000000 4 4 35
stream 0 response time would be 450366(13000000)
stream 1 response time would be 2927829(2100000)
Open failed!
```

Opening the third stream would entail the second stream being unschedulable (with a potentially unbounded response time), and thus we deny the open request.

5. USING DISK LAYOUT INFORMATION

The analysis so far has assumed no knowledge of absolute layout of stream data on the disk — the function $t_i(x)$ gives the head movement costs when sequentially reading x blocks of stream i, but this information is independent of where on the disk the stream is located. It is possible to take advantage of absolute stream layout information when determining the costs of pre-emption. Clearly, if a stream j pre-empts a stream i, but the file opened by stream j is located close to the file opened by stream i, then the full $S_{\max}$ disk head movement costs are not incurred. Further, if the file opened by stream i is located in the middle of the disk then the worst-case time to move the disk head to service requests from stream i must be lower than $S_{\max}$, since the disk head in the worst-case need only traverse half-way across the disk surface.

Thus we can significantly reduce the pessimism of the analysis, entailing a more efficient use of the disk (*i.e.* analysis with pessimism reduced could guarantee more concurrent streams). We now define the following seek times:

$S_{i,j}$	The worst-case time to move the disk head from accessing data of stream i to accessing data of j; this time includes both head movement time and rotational latency
$S_{any,i}$	The worst-case time to move the disk head from accessing data anywhere on the disk to accessing data of stream i

Using this seek-time information we can update the analysis derived earlier. Firstly, the time a given stream k can exclusively hold the disk drive for, from equation 8 can be re-written:

$$S_{any,k} + C_k(n_k) \tag{19}$$

And hence equation 9 (giving the blocking time of a given stream i, B_i) can be re-written:

$$B_i = \max_{\forall k \in lp(i)} \left(S_{any,k} + C_k(n_k) + S_{k,i} \right) \tag{20}$$

The interference from a higher priority stream j on a lower priority stream i, given by equation 10, can be updated to:

$$I_{i,j} = \left\lceil \frac{w_i}{T_j} \right\rceil \left(S_{i,j} + C_j(R_j) + S_{j,i} \right) \tag{21}$$

Note that $S_{i,j}$ is equal to $S_{j,i}$.

The term $S_{i,j}$ includes a rotational latency delay, since the block requested by stream j could have any arbitrary position within a track relative to the last block read by stream i.

We can now embody this updated analysis into a tool and resume the example of the previous section. Knowing about the disk layout we can open the first four streams:

```
command: open sttng_1_aud 52 1000000 4 4 25
stream 0 response time would be 190533(13000000)
stream 0 opened
command: open sttng_1_vid 210 300000 30 5 100
stream 0 response time would be 275416(13000000)
stream 1 response time would be 347649(2100000)
stream 1 opened
command: open rtstng_tut_aud 52 1000000 4 4 35
stream 0 response time would be 204316(13000000)
stream 1 response time would be 465665(2100000)
stream 2 response time would be 336382(13000000)
stream 2 opened
command: open rtstng_tut_vid 162 100000 6 3 150
stream 0 response time would be 214066(13000000)
stream 1 response time would be 489198(2100000)
stream 2 response time would be 262082(13000000)
stream 3 response time would be 575347(2700000)
```

```
stream 3 opened
```

When we come to opening the fifth stream we find the following:

```
command: open cd_sound 220 500000 44 22 50
stream 0 response time would be 257716(13000000)
stream 1 response time would be 950430(2100000)
stream 2 response time would be 293482(13000000)
stream 3 response time would be 2874635(2700000)
Open failed!
```

Stream 3 (corresponding to file `rtstng_tut_vid`) becomes unschedulable, and hence the open request for `cd_sound` must be denied. However, we can still open a further audio stream:

```
command: open audio_1 52 1000000 4 4 45
stream 0 response time would be 220766(13000000)
stream 1 response time would be 710114(2100000)
stream 2 response time would be 256532(13000000)
stream 3 response time would be 869379(2700000)
stream 4 response time would be 542148(13000000)
stream 4 opened
```

Now that the worst-case response times of the streams have been calculated we note that the buffer spaces defined far exceed the actual worst-case buffer usage. Using equation 18 we compute the buffer space needed; the results of this are shown below:

stream	*r*	*T*	*R*	β
0	220776µs	1000000µs	4	4
1	710114µs	300000µs	30	90
2	256532µs	1000000µs	4	4
3	869379µs	100000µs	6	54
4	542148µs	1000000µs	4	4
			total	156

Returning to the problem of opening streams we find that opening a sixth stream to read `audio_2` fails:

```
command: open audio_2 52 1000000 4 4 55
stream 0 response time would be 208166(13000000)
stream 1 response time would be 1068594(2100000)
stream 2 response time would be 243932(13000000)
stream 3 response time would be 2750335(2700000)
Open failed!
```

Thus we can see that by taking account of disk layout information we can obtain a very much less pessimistic assessment of whether a given stream can be opened or not.

6. DISCUSSION

In this section we undertake some general discussion of the problem of hard real-time multi-media disk traffic.

Buffer space and available disk bandwidth are inextricably linked: an infinite amount of buffer space permits 100% utilisation of the disk drive. As has been mentioned earlier, a given static buffer allocation determines the deadline of a stream, which in turn determines the schedulability of the disk traffic. Thus a small allocation of buffer space results in a short deadline, which in turn entails a stream that is difficult to schedule. Hence the choice of the static buffer allocation important. We leave it to other work to examine this issue in detail.

The layout of multi-media files on the disk is crucial to real-time performance. The analysis in this paper takes into account layout information (to compute $S_{i,j}$, for example). Thus the analysis can be used as a metric for laying out data on the disk. As Lougher and Shepherd state, when placing multi-media data on a CD-ROM much time can be spent ensuring that the disk layout is optimal. Further, they point out that observations made on the Lancaster Multi-media test-bed [13] indicate that streams that are recorded together tend to be played back together. The analysis reflects the timing benefits obtained when placing the files opened by such concurrently played streams close together on the disk, since for any two streams i and j (where j is of higher priority than i), the value of $S_{i,j}$ and $S_{j,i}$ will be much smaller than if the files were placed far apart, and hence the disk head movement overheads will be smaller when requests from stream j pre-empt those from stream i. Since the analysis can be used as a metric to evaluate the performance of a given stream, this can form part of a 'fitness function', and general purpose combinatorial optimisation techniques (such as simulated annealing [4]) can then be used to solve the problem of placing files on the disk[5].

We argue that a sensible approach is to require file accesses to be sequential, since this permits less pessimistic bounds on disk head movement when satisfying stream requests (most importantly, reducing rotational latency time). However, this prohibits "rewind" or "skip to" requests within an open stream. The solution to this problem is to make the position for stream playback to be a parameter of the "open stream" request; a "rewind" operation on an open stream is accomplished by closing the stream and re-opening it at the appropriate starting place.

There is potential for further reduction of pessimism in the head movement costs of pre-empting one stream by another. Consider two streams i and j, where j is of higher priority. If these two streams are replayed concurrently and at the same rate, such that the streams start and finish at the same relative position, then the cost of switching to j from i can be lower than the cost of moving the head from anywhere in the file opened by stream i to anywhere in the file opened by stream j. Thus knowledge of playback

[5]Tindell [9] *et al* use simulated annealing to solve an analogous problem by using simple schedulability analysis to place tasks across processors in a distributed system.

characteristics can permit a less pessimistic bounding of $S_{i,j}$. This concept of 'linked' streams has been discussed by Venkat Rangan and Vin [13] where the term *rope* is used to denote a set of streams synchronised and played together[6]. If such characteristics are introduced into the model then previous work by Tindell [9] on scheduling tasks with fixed phasing relationships can be integrated into the theory developed here.

7. CONCLUSIONS

This short report has applied existing fixed priority pre-emptive scheduling theory to the problem of scheduling hard real-time multi-media disk traffic and has provided an on-line test of whether a given stream can be opened. An example using the analysis has been given.

Our motivation is to provide a framework in which various parameters for disk scheduling can be evaluated. Given a configured system our analysis shows how the worst-case behaviour can be predicted. The forms the basis for investigating optimal disk performance for multi-media applications. The analysis presented here shows that there are a number of interdependent parameters:

- the priority of each stream
- the buffer size allocated to each stream
- the size of the non-preemptable read operations

It is unlikely that a simple optimisation of these parameters is possible. Different application scenarios will give rise to alternative configurations. Anticipation of future needs when deciding up the priority and buffer size for a new request are key issues. Quality of service is also important: it is possible to analyse situations in which a new stream can only be opened if an existing stream suffers a temporary reduction in quality following, for example, priority or buffer re-assignment.

We have also shown that the layout of the disk is critically significant in reducing the 'context switch' times between streams. Although this general result was known before, we now have a metric to precisely evaluate different schemes.

8. ACKNOWLEDGEMENTS

The authors would like to thank Robert Davis for constructive comments on an earlier draft of this report.

9. REFERENCES

[1] Audsley, N., Burns, A., Richardson, M., Tindell, K. and Wellings, A., "*Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,*" Software Engineering Journal (July 1993).

[6]The term 'rope' comes about because they refer to streams as 'strands', and thus a rope is a number of interwoven strands.

[2] Burns, A., Nicholson, M., Tindell, K., Zhang, N., "*Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System*", Proceedings of the The Workshop on Parallel and Distributed Real-Time System, April 13-15 1993, pp. 11-20

[3] Burns, A., Wellings, A.J., and Hutcheon, A.D., "*The Impact of an Ada Run-time System's Performance Characteristics on Scheduling Models,*", Ada sans frontieres: Proceedings of the 12th Ada-Europe Conference, Lecture Notes in Computer Science 688, Springer-Verlag (1993).

[4] Kirkpatrick, S., Gelatt, C. and Vecchi, M., "*Optimisation by Simulated Annealing,*" Science (1983), No. 220, pp.671-680

[5] Liu, C. L. and Layland, J. W., "*Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment,*" Journal of the ACM 20(**1**) (1973) pp.46-61.

[6] Lehoczky, J. P., "*Fixed Priority Scheduling of Periodic Task Sets With Arbitrary Deadlines,*" Proceedings 11th IEEE Real-Time Systems Symposium (5-7 December 1990) pp.201-209.

[7] Lougher, P. and Shepherd, D., "*The Design of a Storage Server for Continuous Media,*" Computer Journal 36(**1**) (1993) pp.32-42.

[8] Sha, L., Rajkumar, R. and Lehoczky, J., "*Priority Inheritance Protocols: An Approach to Real-Time Synchronisation,*" IEEE Transactions on Computers 39(**9**) (September 1990) pp.1175-1185.

[9] Tindell, K., "*Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets,*" YCS 182, Department of Computer Science, University of York (1992).

[10] Tindell, K., Burns, A. and Wellings, A., "*Allocating Real-Time Tasks (An NP-Hard Problem Made Easy),*" Real Time Systems 4(**2**) (June 1992) pp.145-165.

[11] Tindell, K., Burns, A. and Wellings, A., "*An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks,*" YCS 189, Department of Computer Science, University of York (1992).

[12] Tindell, K., Burns, A. and Wellings, A., "*Mode Changes in Priority Pre-emptively Scheduled Systems,*" Proceedings of the Real Time Systems Symposium, Phoenix, Arizona (1992).

[13] Venkat Rangan, P. and Vin, Harrick M. , "*Designing File Systems for Digital Video and Audio,*" Operating Systems Review 25(**5**) (1991) pp.81-94. Thirteenth ACM Symposium on Operating Systems Principles.

APPENDIX A — ANALYSIS PROGRAM

A small program implementing the analysis in this paper was used to analyse the example given earlier. The source code to this program is available via anonymous FTP from the following site:

`minster.york.ac.uk` (IP address `144.32.128.41`)

in the directory:

`/pub/realtime/programs/src/disksched`

Additionally, many of the papers and reports written at York are available in compressed postscript format in the directory:

`/pub/realtime/papers`

The file `INDEX` lists the available documents.