Scheduling Real-time Multimedia Tasks in Network Processors

Jingnan Yao, Jiani Guo, Laxmi Bhuyan and Zhiyong Xu
Department of Computer Science and Engineering
University of California, Riverside
{jyao,jiani,bhuyan,rodgexu}@cs.ucr.edu

***Abstract*— Several companies have introduced powerful network processors (NPs) that can be placed in active routers to execute application level tasks in the network. An NP consists of a number of on-chip processors to carry out packet level parallel processing operations. We propose to employ them for multimedia streaming (transcoding) to convert the incoming video streams to low bit-rate media units as per the requirements of the clients. To effectively schedule the parallel transcoding operations in an active router, we propose a Static Sequentialized Batch-CoScheduling (SSBC) scheme to meet both load balancing and real-time requirements for media streaming, based on divisible load theory (DLT). We first analyze the feasibility and optimality of the load distribution schemes from the theoretical perspectives, and then present separate solutions for non-delay-sensitive streams and delay-sensitive streams. Rigorous simulations and experiments have been carried out to evaluate the performance.**

I. Introduction

The transmission of multimedia information through networks has long been a research topic, and it is claimed that multimedia application is becoming one of the killer applications. Due to the receiver heterogeneity and varying network conditions, streaming media data is not as simple as just transmitting them through the network. Imagine that a media server sends a high-bit-rate video/audio stream to a low-bit-rate mobile client; the video/audio cannot be transferred as it is. It should be converted into low-bit-rate video stream to match the client's requirements. Besides, the media streams have to adapt to the variation of available bandwidth in the transmission channel.

A traditional way to solve the problem is to store multiple copies of the video streams on the media server, and select a copy according to some initial negotiation with the client. However, the reliability and bandwidth of a connection from various parts of the network to the client may change during a streaming session. It can also be the case that a mobile client moves physically from place to place, or some other clients join and leave. Multiple stored encoding versions of the streams, therefore, are not enough. The wide range of needs in video rates, sizes, and bandwidths can be met by real-time transcoding service. The multimedia streams can be transcoded to the appropriate format by active routers in the network before they reach the destination, as shown in Figure 1. Such a scheme was originally proposed in [1] with a cluster-based active router to accomplish this task.

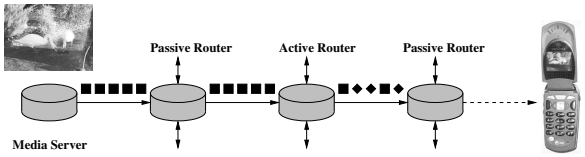

Fig. 1. An active router in network

With the advent of powerful network processors (NPs) in the market, this task can now be accomplished more easily without building a cluster of computers. Each NP consists of a number of on-chip processors that can provide high throughput for network protocol processing and application level tasks [2], [3], [4]. Media streaming for real-time playback is highly delay-sensitive, and the playback quality is the most important concern to the clients. Too much jitter may greatly deteriorate the playback quality. To reduce the jitter, media units of each stream must be processed timely and orderly in the NPs. Thus, taking into consideration both the workload characteristics of media streams and the real-time requirements, we have two design goals for our scheduling algorithm: 1) to achieve high throughput on the network processors and 2) to maintain the flow order of the outgoing media stream to reduce jitter.

Packet scheduling is a critical issue for NPs or multiprocessors, in general, to ensure fast processing and good utilization. Although a plethora of scheduling schemes have been proposed for multiprocessors, simple static policies, such as round robin or random distribution policy [1], [5], [6] are adopted in practice. However, these schemes do not consider the processing order or jitter problem. It was shown that round robin is simple and fast, but provides no guarantee to the playback quality of output streams because it causes out-of-order departure of processed media units [7]. Adaptive load sharing scheme, which we implemented from the literature [8], achieves better unit order in output streams, but involves higher overhead to map the media unit to an appropriate node. The aim of this paper is to derive a theoretical framework for load scheduling in a multimedia active router built with a network processor that comprises of a number of processors for packet processing.

We would like to apply Divisible Load Theory (DLT) [9] to schedule multimedia tasks. DLT is concerned with scheduling divisible loads in parallel and distributed systems. The loads are assumed to be large in size, homogeneous, and are arbitrarily divisible. Also, the communication cost in

collecting processed data is ignored. The primary objective in the research of DLT is to determine the *optimal fractions* (distribution) of the entire load for assignment to each of the processors such that the total processing time is minimized. This is assured by distributing tasks in such a way that all the processors finish their executions at the same time. Multiple-installment load distribution is proposed in [10] to further reduce the processing time for very large workload. DLT cannot be directly applied to multimedia processing because: 1) each task consists of media units that cannot be further divided; 2) delivering a processed media unit takes non-negligible communication time, for example, a normal MPEG GOP (Group of Pictures) consists of about 50 1KB packets [7]; and 3) the process order of consecutive media units in a media stream should be maintained. Media units are preferred to be finished in order instead of being finished at the same time.

In this paper, we propose an efficient load scheduling scheme, Static Sequentialized Batch CoScheduling(SSBC), to achieve load balance among the network processors and also maintain the flow order of the outgoing media stream to reduce jitter. The rest of the paper is organized as follows. In Section II, we investigate certain design issues that need to be addressed in an active router, and propose a novel scheduling scheme SSBC for real-time transcoding. In Section III, we present theoretical solutions of SSBC for both non-delay-sensitive and delay-sensitive tasks. Analytical and experimental results are presented in Section IV with varying number of processors and number of batches for processing. We present discussions for these results in this section as well. Finally, in Section V, we conclude the paper with future possible extensions related to this paper.

II. Static Sequentialized Co-Scheduling (SSBC)

Figure 2 illustrates the multiprocessor architecture model of an active router using a network processor (NP). The NP consists of one receiving processor P_0, a few worker processors, P_1 through P_m, and a transmitting processor P_x. Intel IXP NP divides its set of microengines this way for packet processing [2]. The receiving and transmitting processors communicate with the I/O ports sequentially. There are incoming queues to P_0, which also schedules packets in the processors in a sequential order. Similarly, the transmitting processor P_x receives packets from processors P_1 through P_m and sends them to the output port sequentially. The incoming media stream is a sequence of media units that need to be transcoded in the NP before their departure. Each media unit may correspond to a GOP of MPEG video. Each worker processor independently transcodes GOPs without coordinating with other processors. The aim of this paper is to derive a scheduling strategy for P_0 so that the flow order is maintained at the output, and as a result, the jitter is reduced.

We start by exploring how DLT works for the transcoding workload. As suggested by the DLT literature [9], in order to obtain an optimal processing time, it is *necessary and sufficient* that all the processors participating in the computation finish

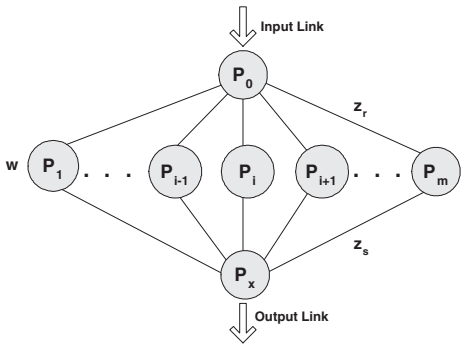

Fig. 2. Load Scheduling Model

computing at the same time instant. This condition is referred to as an *optimality principle* and serves as a baseline of the load distribution strategies. Therefore, as a first step, we apply this optimality principle to our transcoding workload by letting all processors complete transcoding at the same time. The transcoding workload can thus be divided into multiple load fractions with the granularity of one GOP, with each worker processor transcoding one fraction. All worker processors complete processing simultaneously and thus start delivering the transcoded media fractions to the output link simultaneously. In this way, the workload is well balanced and the worker processors are fully utilized. However, if we take a look at how such a scheme works by observing the packets at the output link, some problems are evident.

As illustrated in the first half of Figure 3, processor P_1 transcodes three GOPs, each fragmented into four packets while being transmitted to the output link. P_2 transcodes two GOPs. Both processors complete transcoding at the same time instant and start delivering GOPs to the output link simultaneously. Inevitably, during the delivery, they have to compete for the single output link. Consequently, packets of two different GOPs, GOP 1 from P_1 and GOP 4 from P_2, are interleaved while being transmitted. Note that interleaving among packets may not fall into such a regular pattern in the real life situation, as shown in Figure 3, which may further deteriorate the playback quality. Obviously, such a scheme introduces out-of-order departure of GOPs 1 and 4 on the output link, similarly for GOPs 2 and 5. This increases the jitter as well.

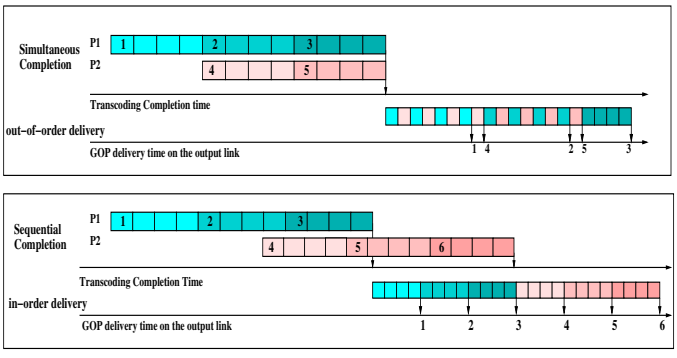

Fig. 3. Simultaneous Completion vs. Sequential Completion

From the above observation, we can conclude that: 1) because of the single shared output port, simultaneous completion causes concurrent data delivery; 2) concurrent data delivery incurs uncontrollable interleaving among packets and causes out-of-order departure of GOPs; 3) jitter increases as the number of worker processors grows.

Thus, instead of letting all the processors complete processing simultaneously, we propose a sequential completion pattern of the transcoding service rendered by different processors and thus a sequential data delivery on the output link. As shown at the bottom half of Figure 3, while P_1 is delivering data, P_2 is still busy transcoding its workload and the output link is only occupied by P_1. Therefore, GOPs 1, 2 and 3 can be delivered in order. Then, P_2 starts delivering data only after P_1 completes its data delivery. Thus, GOPs 4 and 5 can also depart in order. *The idea of such a sequential completion pattern forms the basis of our scheduling strategy*.

We propose a Static Sequentialized Batch-CoScheduling (SSBC) scheme for real-time transcoding. In our scheme, each worker processor processes a fraction of the total transcoding workload in parallel, and each processor works in the following three steps: 1) R-step: the worker processor receives media units from the receiving processor; 2) T-step: the worker processor transcodes all the media units it receives in R-step; 3) S-step: the worker processor sends to the transmitting processor all the processed media units it transcoded in T-step.

Assume that there are m worker processors in the router, the receiving processor P_0 statically divides the total workload into n batches, each consisting of m load fractions. It dispatches each load fraction to the worker processors sequentially. Each worker processor starts processing once it receives its assigned load fraction. As shown in Figure 4, the load is divided into 3 batches and scheduled in such a way that the worker processors complete transcoding sequentially. When a worker processor delivers media data to the output link, all other worker processors are busy processing their own loads, and will not compete for the output link.

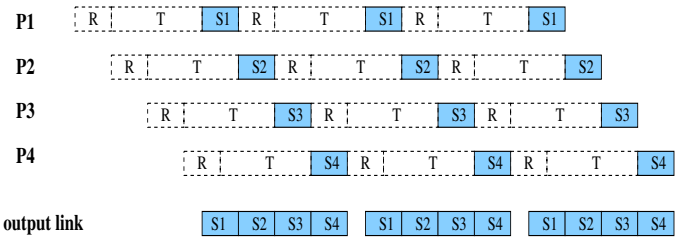

Fig. 4. Static Sequentialized Batch CoScheduling

Using the SSBC scheme, out-of-order data delivery is avoided, and the jitter can also be controlled systematically by choosing an appropriate load partition. Moreover, SSBC differs from DLT solutions in that it considers the time taken to deliver the processed results, which is ignored in DLT but cannot be neglected in multimedia service.

The key of the scheduling algorithm is to find optimal load partitions and number of batches to achieve good performance. We first categorize the workloads into two different types: delay-sensitive streaming and non delay-sensitive streaming. We define the initial delay for a media stream as the time duration between arrival of the first GOP and its departure. The initial delay must be as small as possible for delay-sensitive streams. For non-delay sensitive streams (also a "light" task), we can simply dispatch the requested media in a single batch. For delay-sensitive streams (also a "heavy" task), multiple batches are preferred to reduce both the idle time of the worker processors and the initial delay for the departing stream.

III. Theoretical Approach of SSBC

In this section, for simplicity, we present the theoretical approach of SSBC systematically for both delay-sensitive and non-delay-sensitive tasks in a homogeneours NP system. However, the proposed solutions are also appliable to a heterogeneous system, as presented in our technical report [11].

A. Terminology

We now introduce the necessary notations, definitions, and the terminology that will be used throughout the rest of this paper.

N	The total number of GOPs of a media stream.
m	The total number of worker processors in the router.
n	The number of batches of a load distribution.
$l_{r,i}$	Communication link connecting worker processor P_i with the receiving processor P_0.
$l_{s,i}$	Communication link connecting worker processor P_i with the transmitting processor P_x.
T_{cp}	Time taken to transcode a GOP by a standard processor.
T_{cm}	Time taken to communicate a GOP on a standard link.
β	The shrink rate of a GOP after it is transcoded.
z_r	A constant that is inversely proportional to the speed of communication link $l_{r,i}$. Thus, $z_r T_{cm}$ is the time P_i takes to receive a GOP from P_0.
w	A constant that is inversely proportional to the speed of processor P_i. Thus, wT_{cp} is the time P_i takes to transcode a GOP.
z_s	A constant that is inversely proportional to the speed of communication link $l_{s,i}$. Thus, $\beta z_s T_{cm}$ is the time P_i takes to send a GOP to the transmitting processor.
α	This refers to a load distribution of a media stream, i.e., $\alpha = (\alpha_{1,1},...,\alpha_{i,1}...,\alpha_{i,j},...,\alpha_{m,n})$, where $\alpha_{i,j}$ is the fraction of load assigned to P_i in the jth batch such that $0 \leq \alpha_{i,j} \leq 1$ and sum of all the above load fractions amounts to the total load to be processed, that is, $\alpha = \sum_{j=1}^{n} \sum_{i=1}^{m} \alpha_{i,j}$.
$L_{i,j}$	The number of GOPs scheduled to P_i in the jth batch. Thus, $L_{i,j} = \alpha_{i,j} \times N$. For a load distribution with a single batch, $\alpha_{i,j} = \alpha_i$, $L_{i,j} = L_i$.
$R_{i,j}$	Time duration P_0 takes to distribute $L_{i,j}$ to P_i.
$T_{i,j}$	Time duration P_i takes to transcode $L_{i,j}$.
$S_{i,j}$	Time duration P_i takes to send $L_{i,j}$ to P_x.
$T(\alpha)$	The total processing time of a media stream in the router under the distribution scheme α.

B. Mathematical Model

With the above mentioned router architecture in place, here we set up a mathematical model for the ease of a theoretical approach. As shown in Figure 2, there are $(m+2)$ network processors and $2m$ links inside the router. The worker processors P_1, P_2,...,P_m are connected to the receiving processor P_0 via links $l_{r,1}$, $l_{r,2}$,...,$l_{r,m}$. Meanwhile, each worker processor has a direct link $l_{s,i}$ to the transmitting processor P_x. The receiving processor receives the media streams from the input link, divides the total processing load into m parts and then distributes these load fractions to the corresponding worker processor. Each worker processor P_i starts transcoding immediately upon receiving its load fraction α_i and continues to do so until this fraction is finished. Finally, P_i sends the transcoded fraction to the transmitting processor P_x via link $l_{s,i}$. Without loss of generality, we assume that the load is distributed from the left hand side to the right hand side among the worker processors. Hence, we assume that the sequence of load distribution is fixed.

C. Optimal Solution for Non-delay-sensitive Tasks

In this section, we introduce an optimal solution for the non-delay-sensitive media streams. In this case, the incoming stream is less delay sensitive and would allow longer initial delay for the transcoding as long as the flow order of the stream is maintained. To meet this requirement, we distribute the media stream in a single batch and partition the total load such that all the media fractions depart from the router sequentially and continuously after transcoding.

As shown in the desired timing diagram (Figure 5), the receiving processor first partitions the load into m parts and distributes them to the corresponding worker processor sequentially. Each worker processor begins transcoding immediately upon receiving its load fraction and sends it to the output port right after the transcoding is over. The load fractions are divided such that all the worker processors stop transcoding sequentially and therefore, deliver the transcoded fractions to the output port sequentially. That is, worker processor P_{i+1} will start delivering the packets to the output port only after and right after P_i finishes its delivery. Thus, all the media fractions are transcoded in parallel and are sent to the output port in order without any break. In this way, we avoid the competition of the output port and eliminate the out-of-order problem of the media stream. Although there might be longer initial delay, (yet tolerable since the stream is less delay-sensitive and the load is light), once the first fraction of the media stream departs from the output link, the following parts departs one after another continuously. To achieve such a non-stopping sequential delivery pattern, we have the following recursive equations from the timing diagram:

$$\alpha_i w T_{cp} + \alpha_i \beta z_s T_{cm} = \alpha_{i+1} z_r T_{cm} + \alpha_{i+1} w T_{cp}, \quad (1)$$

$$\text{Thus, } \alpha_i = f\alpha_{i+1},\ i = 1, ..., m-1 \quad (2)$$

$$\text{where, } f = \frac{z_r T_{cm} + w T_{cp}}{w T_{cp} + \beta z_s T_{cm}}, \quad (3)$$

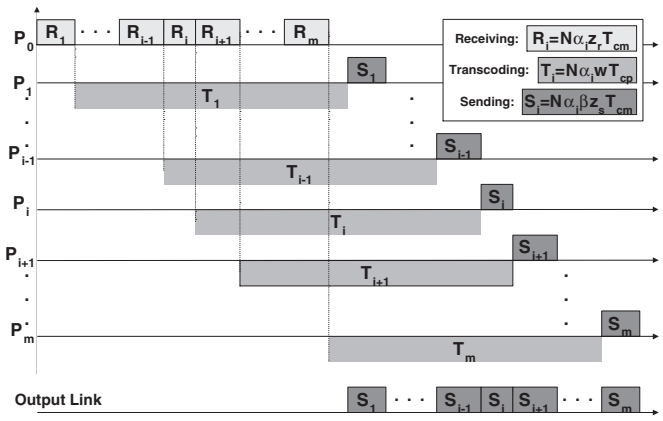

Fig. 5. Load Scheduling in One Batch

These recursive equations can be solved by expressing all the α_i (i=1,...,m-1) in terms of α_m and using the normalizing equation $\sum_{i=1}^{m} \alpha_i = 1$.

$$\alpha_i = \alpha_m f^{m-i},\ i = 1, ..., m \quad (4)$$

$$\text{where, } \alpha_m = \frac{1}{1 + \sum_{u=1}^{m-1} f^{m-u}}, \quad (5)$$

Thus, using the above closed-form solution, we can obtain the optimal load fractions for scheduling a stream. However, as the number of GOPs suggested by the optimal solution are not integers, we cannot use it directly in our scheduling. Integer approximation is needed to yield integer load fractions. In this paper, we shall use the algorithm proposed in [12] to carry out this approximation. Figure 6 presents the complete workings of this procedure.

Procedure: **Non-delay-sensitive Optimal Load Distribution**
Step 1: Using equations (4) and (5), determine the optimal load fractions α_i.
Step 2: Find out the number of GOPs that should be scheduled to each worker processor. /* $L_i = \alpha_i \times N$
Step 3: Carry out the integer approximation on all L_i.

Fig. 6. Non-delay-sensitive Optimal Load Distribution

D. Solutions for Delay-sensitive Tasks

In this section, we consider the problem of scheduling delay-sensitive tasks in the active router. In this case, it is essential to reduce the initial delay during transcoding without affecting the flow order of the stream. Thus, we dispatch the media stream to the worker processors in multiple batches instead of scheduling it in just one batch. To maintain the desired property of a sequential delivery to the output port, we shall follow the same scheduling pattern addressed in Section III-C within each batch. We first analyze the optimality constraints of an ideal solution and based on that, we propose a relaxed solution for a generic router configuration.

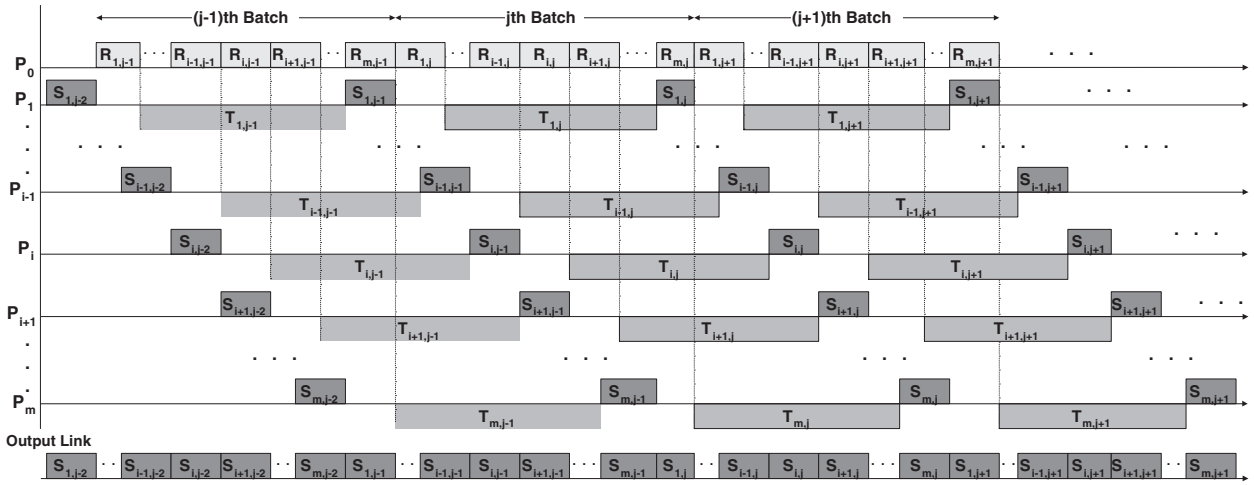

Fig. 7. Optimal Solution for Delay-sensitive Tasks

1) Optimality Analysis: So far in the design of our scheduling strategies, we mainly focused on reducing the out-of-order rate and departure jitter from a stream's perspective. However, it is also critical to guarantee the maximum throughput of the router. This is essentially true when the network traffic is very heavy. Thus, for an optimal solution, we shall partition the load such that there is no idle time for any of the processors in the router while still maintaining a continuous sequential delivery pattern for the passing streams. To illustrate these ideas, we show the desired timing diagram in Figure 7 for a load distribution of m worker processors and n batches. As can be observed from the diagram, multiple batches have been joined together seamlessly to carry out the entire transcoding task. There is no idle time on both receiving processor and any of the worker processors at any time instant. The entire load is therefore divided into $(m \times n)$ parts such that the $(j+1)$th batch arrives at a worker processor right after it finishes delivering the current batch to the output port. In this way, all the transcoded parts are sent to the output link timely and sequentially, and none of the processors is idle. To determine such an optimal load distribution, we can observe the following recursive equations from the timing diagram:

$$\text{for } (i = 1 \sim m-1, j = 1 \sim n)$$

$$\alpha_{i,j} w T_{cp} + \alpha_{i,j} \beta z_s T_{cm} = \alpha_{i+1,j} z_r T_{cm} + \alpha_{i+1,j} w T_{cp} \quad \text{(1-1)}$$

$$\text{for } (j = 1 \sim n-1)$$

$$\alpha_{m,j} w T_{cp} + \alpha_{m,j} \beta z_s T_{cm} = \alpha_{1,j+1} z_r T_{cm} + \alpha_{1,j+1} w T_{cp} \quad \text{(1-2)}$$

$$\text{for } (i = 2 \sim m, j = 1 \sim n-1)$$

$$\alpha_{i,j} \beta z_s T_{cm} = \alpha_{i-1,j+1} z_r T_{cm} \quad \text{(2-1)}$$

$$\text{for } (i = 1,\ j = 1 \sim n)$$

$$\alpha_{1,j} \beta z_s T_{cm} = \alpha_{m,j} z_r T_{cm} \quad \text{(2-2)}$$

The above two sets of equations are independent and can be solved uniquely by expressing all the $\alpha_{i,j}$ in terms of $\alpha_{m,n}$ and using the normalizing equation $\sum_{j=1}^{n} \sum_{i=1}^{m} \alpha_{i,j} = 1$. Thus, it is unlikely to obtain an α satisfying both the equations unless the system parameters are constrained by certain relationship. For a homogeneous network, we derive the following constraint for the optimal solution:

$$\frac{z_r}{z_s}\beta = \left(\frac{z_r T_{cm} + w T_{cp}}{w T_{cp} + \beta z_s T_{cm}}\right)^{m-1} \quad (6)$$

which holds true if $\beta z_s = z_r$.

We found that the practical implementation may not necessarily satisfy the optimality requirement of equation (6). Thus, there is no general optimal solution for a generic router configuration. The nonexistence of such a general optimal solution is due to the fact that we want to maintain the flow order of the incoming stream to eliminate the out-of-order problem and at the same time best utilize the computing power of the network processors. This incurs two sets of independent equations when we are to determine α, thus inevitably adding constraints on the system parameters to satisfy both of the design goals. Therefore, to achieve a feasible solution for a general NP setting, somehow we need to relax one of these constraints in our equations. Based on this observation, we propose a relaxed solution as follows.

2) Relaxed Solution: As shown in Figure 8, this solution is obtained by relaxing the requirement for a continuous delivery of the packets at the output link. Gaps may be allowed between the departure of two adjacent batches, but not within a batch. To do so, we first compute the optimal load fractions for a single batch and duplicate the same pattern for multiple times. However, we need to determine the best time to start despatching a new batch from the receiving processor. So, gaps will be introduced between the initialization of two batches with a care to minimize this gap. To identify such a minimum gap between two batches, we observe the following inequations. Note that time fractions R_i, T_i and S_i are obtained from the optimal load distribution α_i of a single batch.

$$\begin{aligned}
&\textstyle\sum_{k=1}^{m} R_k + Gap \geq R_1 + T_1 + S_1 \\
&\textstyle\sum_{k=1}^{m} R_k + Gap + R_1 \geq \sum_{k=1}^{2} R_k + T_2 + S_2 \\
&\textstyle\sum_{k=1}^{m} R_k + Gap + \sum_{k=1}^{i-1} R_k \geq \sum_{k=1}^{i} R_k + T_i + S_i \\
&\textstyle\sum_{k=1}^{m} R_k + Gap + \sum_{k=1}^{m-1} R_k \geq \sum_{k=1}^{m} R_k + T_m + S_m
\end{aligned}$$

Thus,
$$Gap_{min} = max(R_i + T_i + S_i) - \sum_{k=1}^{m} R_k \quad (7)$$

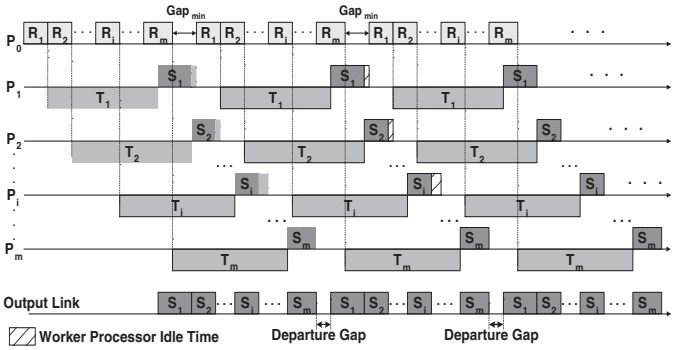

Fig. 8. Timing Diagram of Relaxed Solution

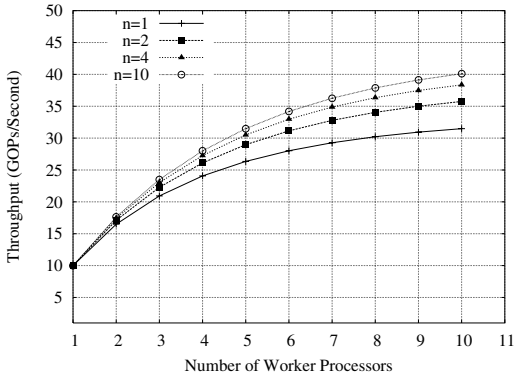

Fig. 10. Throughput of Relaxed Solution

Now that we have determined the proper time for the receiving processor to initiate a new batch, we can safely duplicate the dispatching for multiple times to reduce the total transcoding time. This solution is intuitive and it is very easy to implement. Figure 9 presents the corresponding pseudo-code for this procedure.

Procedure: **Relaxed Solution**
Step 1: Let $N = N/n$, determine the optimal load fractions L_i for a single batch. /* Figure 6
Step 2: Using equation (7), determine Gap_{min}.
Step 3: Let $L_{i,j} = L_i$ for $j = 1, ..., n$. Schedule a batch of load every Gap_{min} time for n times.

Fig. 9. Relaxed Solution

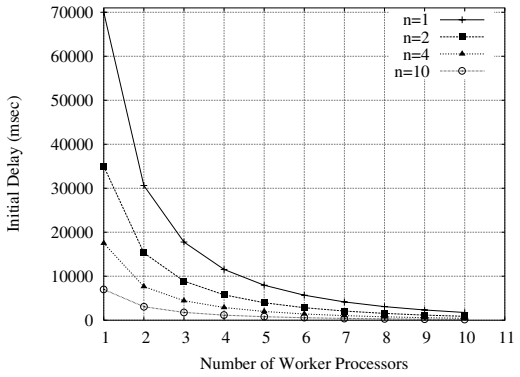

Fig. 11. Initial Delay of Relaxed Solution

IV. Results and Discussion

In this section, we study performance of the above strategies and carry out limited experimental verification. We highlight and discuss all the important issues. From our experiment, we obtained $z_r T_{cm} = 10ms$, $wT_{cp} = 60ms$, $\beta z_s T_{cm} = 30ms$, $N = 1000$. We use these values in the equations, derived for the relaxed solution, and obtain the results. We vary the number of batches n from 1 to 10 and the number of worker processors m from 1 to 10 to observe the performance. We evaluate the performance in terms of throughput (number of GOPs completed per second) and initial delay.

A. Analytical Results

Figure 10 shows the variation of the GOP processing rate (throughput) as a function of the number of processors, m. Clearly the performance improves as more and more processors are employed subject to saturation beyond a certain point. The curves are also plotted for various batch sizes n. Note that the values for $n = 1$ corresponds to the optimal solution for a non-delay-sensitive stream. However, it performs worse compared to the delay-sensitive situation when more batches are employed. As the number of batches increases, the throughput increases. Similarly, as shown in Figure 11, we can observe that the initial delay through the router reduces when m and n increase. Initial delay is the time taken by the first GOP from the time it was sent to the receiving processor until it comes out from the worker processor.

Thus, from these figures, we can observe a tradeoff between m and n. When the active router has fewer number of processors, dispatching the load in more batches is less effective than adding a worker processor. On the other hand, when there are a large number of processors in the router, adding more batches is more beneficial than further increasing the processors. Therefore, for the design of an active router and its scheduling scheme, we can pick a pair of (m, n) that achieves the most cost effective solution. Moreover, as stated earlier, our solutions eliminate the out-of-order problem and guarantees a smooth outgoing stream by reducing the departure jitter and the transcoding time.

B. Experimental Verification

We have a network processor laboratory consisting of IXP 2400s, donated by Intel. The programming is done in Micro C, which is like an assembly language. There is no compiler yet that can convert the transcoding C program [13] to Micro C for implementation in IXP. Hence, we tried to verify our theory by limited implementation in a PC cluster environment, described in detail in our earlier paper [7].

Figure 12 gives the timing diagrams for both theoretical expectation and experimental verification with the workload size of $N = 300$ GOPs and four processors. Out of these P_0 acts as receiving/scheduling processor and the other three

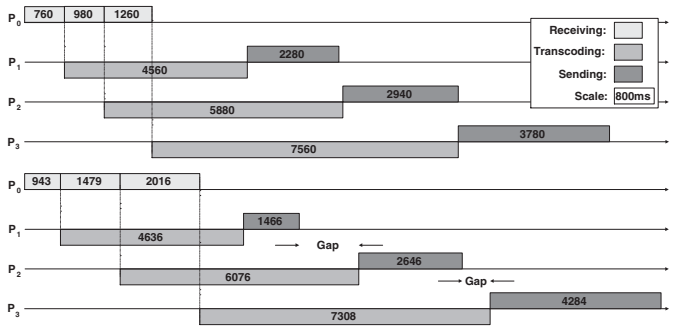

Fig. 12. Theoretical Expectation vs. Experimental Verification

act as worker processors. The theoretical figure was obtained from equations, derived for non-delay-sensitive (one batch) scheduling in Section III-C. The load distribution to each processor was set as per the theoretical derivation.

As we can see, the experimental results roughly match the theoretical expectation. The times to send the GOPs and compute them take longer than what was assumed in theory. There are some gaps between the sequential deliveries that appear in experimental results. These differences occur because we assumed fixed size for each GOP equal to the average GOP size of 50 Kbytes while deriving the theory. In practice, however, the size of different GOPs varies a lot in a real media stream, as shown in [7] while doing the experiment. Therefore, the times to send, transcode, and receive various-sized GOPs exhibit large variance in the experiment. Inspite of this, the output streams are sequentialized with minimal gap between their departures.

In the following table, we compare the theory and the experiment in terms of out-of-order departure rate of GOPs, average departure interval and the departure jitter. The experimental results show that out-of-order departure has been effectively eliminated. Although the departure jitter and average departure interval are higher than those predicted by theory, they are still small enough to achieve smooth playback of the multimedia stream for the potential clients. This is a large improvement compared to the results that we got in [7] for various scheduling strategies.

	Theory	Experiment
Out-of-order Rate	0	0
Average Departure Interval (ms)	30	61.9
Departure Jitter	0	0.0608

TABLE I
PERFORMANCE EVALUATION

V. CONCLUSION

In this paper, we investigated the problem of scheduling multimedia streams on a NP-based active router. We proposed a Static Sequentialized Batch-CoScheduling (SSBC) scheme to render an efficient transcoding service with guaranteed flow order. Following the central idea of SSBC, closed-form solutions are given for both delay-sensitive and non-delay-sensitive streams. The key concept of a sequential delivery pattern used in our algorithms helps to guarantee a high media quality at the client by reducing the jitter, interarrival time and out-of-order rate of the outgoing stream. Further, we evaluated the performance of our strategies in a homogeneous active router through analysis and experiments.

Several extensions seem to be possible at this stage for the problem addressed in this paper. In the design of the algorithms in this paper, we mainly focused on exploring the computation parallelism at a GOP level. However, this is not enough when there are multiple streams passing through the router at the same time. Thus, there is a need to explore and analyze the stream level parallelism for a heavily loaded network. Another possible concern is the sequence of the load distribution among the worker processors, particularly for heterogeneous processors. Instead of following a fixed left-to-right sequence during dispatching, it would be interesting to vary the dispatching sequence and identify an optimal sequence. Lastly, it will be useful to verify our theoretical results by implementing the techniques in a real network processor, like Intel IXP 2400/2800.

VI. ACKNOWLEDGEMENT

This research has been supported by NSF grants ACI-0233858, CCR-0220096 and a research grant from Intel Corporation.

REFERENCES

[1] G. Welling, M. Ott, and S. Mathur, "A cluster-based active router architecture," *IEEE Micro*, vol. 21, no. 1, January/February 2001.
[2] Intel, "Intel ixp2800 network processor," http://www.intel.com/design/network/products/npfamily/ixp2800.htm.
[3] IBM, "The network processor: Enabling technology for high-performance networking," 1999.
[4] Motorola, "Motorola c-port corporation : C-5 digital communications processor," 1999, http://www.cportcom.com/solutions/docs/c5brief.pdf.
[5] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building robust software based router using network processors," *Symposium on Operating Systems Principles (SOSP)*, pp. 216–229, November 2001.
[6] M. Satyanarayanan, "Scalable, secure, and highly available distributed file access," *IEEE Computer*, May 1990.
[7] J. Guo, F. Chen, L. Bhuyan, and R. Kumar, "A cluster-based active router architecture supporting video/audio stream transcoding services," *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France*, April 2003.
[8] L. Kencl and J. Y. L. Boudec, "Adaptive load sharing for network processors," *IEEE INFOCOM*, 2002.
[9] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, "Scheduling divisible loads in parallel and distributed systems," 1996.
[10] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Transactions on Aerospace and Electronic Systems*, April 1995.
[11] J. Yao, J. Guo, L. Bhuyan, and Z. Xu, "Scheduling multimedia real-time tasks in network processors," Dept. of CSE, UC, Riverside, Tech. Rep. TR-001-04, Feburary 2004. [Online]. Available: "http://www.cs.ucr.edu/ jyao/TRSSBC.ps"
[12] V. Bharadwaj and N. Viswanadham, "Suboptimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, November 2000.
[13] T. Ostreich, "Linux video stream processing," http://www.theorie.physik.uni-goettingen.de/˜ostreich/ transcode/.