An Efficient Packet Scheduling Algorithm in Network Processors

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

Abstract— **Several companies have introduced powerful network processors (NPs) that can be placed in routers to execute various tasks in the network. These tasks can range from IP level table lookup algorithm to application level multimedia transcoding applications. An NP consists of a number of on-chip processors to carry out packet level parallel processing operations. Ensuring good load balancing among the processors increases throughput. However, such multiprocessing also gives rise to increased out-of-order departure of processed packets. In this paper, we first propose a Dynamic Batch Co-Scheduling (DBCS) scheme to schedule packets in a heterogeneous network processor assuming that the workload is perfectly divisible. The processed loads from the processors are ordered perfectly. We analyze the throughput and derive expressions for the batch size, scheduling time and maximum number of schedulable processors. To effectively schedule variable length packets in an NP, we propose a Packetized Dynamic Batch-CoScheduling (P-DBCS) scheme by applying a combination of deficit round robin (DRR) and surplus round robin (SRR) schemes. We extend the algorithm to handle multiple flows based on a fair scheduling of flows depending on their reservations. Extensive sensitivity results are provided through analysis and simulation to show that the proposed algorithms satisfy both the load balancing and in-order requirements in packet processing.**

I. Introduction

With the advent of powerful network processors (NPs) in the market, many computation-intensive tasks such as routing table look-up, classification, IPSec, and multimedia transcoding can now be accomplished more easily in a router. Such an NP-based router permits sophisticated computations within the network by allowing their users to inject customized programs into the nodes of the network [1]. An NP provides the speed of an ASIC and at the same time is programmable. Each NP consists of a number of on-chip processors that can provide high throughput for network packet processing and application level tasks [2], [3], [4]. However, processing of packets belonging to the same flow by different processors gives rise to out-of-order departure of the packets from the NP and incurs high delay jitter for the outgoing traffic. For TCP, it has been proved that out-of-order transmission of packets is inimical to the end-to-end performance. For many applications like multimedia transcoding [5], it is imperative to minimize this out-of-order effect because the receiver may not be able to reorder them easily to tolerate high delay jitter.

This research has been supported by NSF grants CCF-0220096, CCF-0311437 and research grants from UC Micro and Intel Corporation.

Today's receivers vary widely from palm devices, PDAs to desktops that may or may not have enough storage and reordering capabilities. Examples of multimedia transcoding in an active router are found in the MeGa project [6] of the University of California, Berkeley, and the Journey network model [7] at the NEC-USA, where routers provide customizable services according to packet requests. Efficient packet scheduling is necessary in order to guarantee both high throughput and minimal out-of-order departures of packets. However, these two goals are contradictory to each other because scheduling on more number of processors increases throughput but also increases out-of-order departure of packets.

Packet processing in an NP can be considered as similar to link aggregation techniques that employ multiple physical links from a source to the same destination. Link aggregation provides increased bandwidth and reliability between the two devices (switch-to-switch or switch-to-station) as more channels are added. Implementations include the Cisco Etherchannel in the CISCO ONS 1500 Series based on the proprietary Inter-Switch trunking (ISL), Adaptec's Duralink port aggregation, 3COM, Bay Networks, Extreme Networks, Hewlett Packard and Sun, etc. A practical link stripping protocol, called Surplus Round Robin (SRR), is proposed by Adiseshu [8] to schedule variable length packets over multiple links with different capacities. They demonstrate that stripping is equivalent to the classic load-balancing problem over multiple channels. They solve the variable packet size problem by transforming a class of fair queuing algorithms into load sharing algorithms at the sender. Although their solution is elegant and efficient, it requires the receiver to run a corresponding resequencing algorithm to ensure in-order delivery of packets.

The aim of this paper is to derive an efficient packet-scheduling algorithm in a network processor that comprises of a number of processors (or channels) for packet processing. It should provide 1) load balancing for processing variable length packets using a group of heterogeneous processors, and 2) in-order delivery of packets without considering receiver's rearranging capability. A plethora of scheduling schemes for multiprocessors have been proposed in parallel processing community. Examples of such schemes vary from simple static policies, such as round robin or random distribution policy [1], [9], [10] to sophisticated adaptive load-sharing policies [11], [5], [12]. However, only simple policies such as round robin are employed in practice because adaptive schemes are difficult to implement and involve considerable overhead. We have shown that round robin is simple and fast, but provides no guarantee to the playback quality of output streams of video signals because it causes large out-of-order departure of the processed media units [5]. Adaptive load sharing scheme, which we implemented from the literature [11], achieves better unit order in output streams, but involves higher overhead to map the media unit to an appropriate node. Recently, we proposed a scheduling algorithm called Static Sequentialized Batch CoScheduling for video transcoding [13] using homogeneous network processors. It considered a single multimedia stream with all the packets available in the input queue at the beginning of the scheduling.

In this paper we derive a Dynamic Batch Co-Scheduling (DBCS) algorithm that considers a backlogged queue, and can be applied to dynamic arrival of packets in an overload situation. Expressions for load distribution in heterogeneous network processors are derived first by assuming that the schedulable workload is perfectly divisible in terms of bytes. The divisible load theory (DLT) for parallel processing has been suggested in [14], [15]. However, they did not consider sequential ordering of the processor execution times, as required for packet transmission over the output link. Our algorithm schedules the packets in batches by computing the optimal batch size, scheduling time, and number of schedulable processors given the maximum packet size and network processor parameters. A batch is similar to the concept of time epochs when scheduling is done. Several interesting results are derived regarding scalability of our algorithm.

Because the arriving packets cannot be distributed to processors in bytes, we also derive a packetized version of the DBCS algorithm by applying a combined version of DRR and SRR algorithms [16], [8]. The P-DBCS algorithm produces better results in terms of throughput and out-of-order rate compared to round robin and pure SRR schemes. We then extend the P-DBCS algorithm to handle multiple flows having reservations. When applying P-DBCS algorithm to multiple flows, both load balancing and fair scheduling requirements should be satisfied. Hence, we revise the expression of the batch size and packet dispatching condition to reflect the new requirements. Finally, we perform a number of simulations and sensitivity studies to verify the accuracies of our theory and obtain performance over wide-ranging input parameters.

The rest of the paper is organized as follows. In section II, we present the preliminaries and certain design issues in a Network Processor. In section III, we design the Dynamic Batch CoScheduling algorithm (DBCS) by doing theoretical derivation and analysis. In section IV, we propose and design a packetized version of the DBCS algorithm called Packetized-DBCS (P-DBCS) to deal with variable length packets. In section V, we present how to achieve fairness among multiple network flows using P-DBCS. Simulation results are presented in section VI in comparison with several other schemes. Finally, in section VII, we conclude the paper with future possible extensions related to this paper.

II. Model for Packet Scheduling

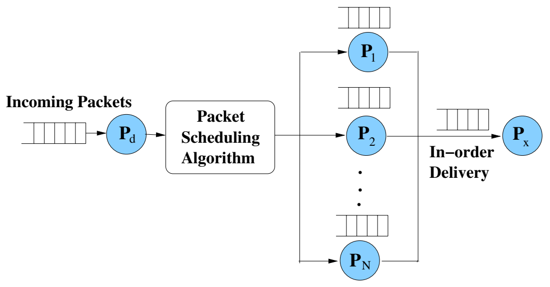


Fig. 1. Packet Scheduling in Network Processors

Figure 1 illustrates the multiprocessor architecture model of a router using a network processor (NP). The NP consists of one dispatching processor P_d, a few worker processors, P_1 through P_N, and a transmitting processor P_x. Intel IXP NP divides its set of microengines this way for packet processing [2]. The dispatching and transmitting processors communicate with the I/O ports sequentially. The dispatching processor P_d schedules incoming packets among the worker processors for packet processing. The transmitting processor P_x receives packets from processors P_1 through P_N and sends them to the output port. The aim of the packet scheduling algorithm is two fold: 1) the input load is balanced among the processors P_1 through P_N; 2) the flow order is maintained when the packets are transmitted to the transmitting processor P_x.

A similar problem called channel stripping has been addressed in the literature by Adiseshu [8]. There are N channels between the sender and the receiver. The sender implements the stripping algorithm SRR to strip incoming traffic across the N channels, and the receiver implements a resequencing algorithm to combine the traffic into a single stream. The stripping algorithm aims to provide load sharing among multiple channels. It does not consider the transmission order among the packets in different channels. Hence, the receiver needs to run a resequencing algorithm to restore the packet order in the original flow. A strict synchronization between the sender and the receiver is difficult to implement.

Although the packet scheduling problem looks different from the channel stripping problem, there are many similarities. First, in channel stripping, the packets are transmitted in the channels. In NPs, the packets are processed on the worker processors. These two times are equivalent and proportional to the load size in bytes. Second, in channel stripping, the time to move packets from the single input port to different channels is assumed to be negligible in [8]. Actually, there should be a time overhead in executing SRR at the IP level processing. As for packet scheduling in an NP, the time to move packets from the dispatching processor to a worker processor cannot be ignored because of the time taken by the dispatching processor and the transmission between the two processors. Finally, the transmitting processor in an NP removes the packets from the worker processors on an FCFS basis, whereas the receiving processor in the stripping model executes a resequencing algorithm. Hence the models developed in this paper are applicable both to packet processing in an NP or packet transmission over multiple channels.

For the rest of the paper, we will explain our models/algorithms just in terms of NPs without loss of generality. To describe the whole procedure experienced by one packet when it is processed in an NP, there are three steps: 1) D-step: the dispatching processor dispatches the packet to a worker processor; 2) P-step: the worker processor processes the packet; 3) T-step: the worker processor sends the packet to the transmitting processor. Correspondingly, in the channel stripping problem, only one step, namely P-step, is modeled, if we take transmission of a packet as a type of processing. Hence, the packet scheduling problem in an NP is more complicated.

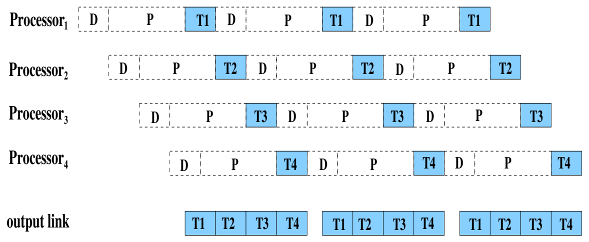


Fig. 2. Dynamic Batch CoScheduling

We propose a sequential completion pattern of the packet processing and thus a sequential data delivery by the worker processors, as illustrated in Figure 2. Let there be N worker processors in the router, each worker processor P_i $\forall i$, first receives some packets from the dispatching processor P_d (D-step), then processes these packets (P-step), and finally sends the packets to the transmitting processor P_x (T-step) sequentially. We propose to let the dispatching processor P_d distribute the packets among the N worker processors from P_1 through P_N in such a way that each worker processor completes processing and transmission sequentially. In another word, the worker processor P_i for (i=2 $\sim$ N) starts delivering the packets to the P_x immediately after P_{i-1} completes its T-step. Therefore, sequential packet delivery is ensured, as well as the load is balanced among multiple processors. We call a scheduling algorithm that produces such a scheduling pattern Dynamic Batch CoScheduling (DBCS). A nice property of the above model is that sequential data delivery is achieved without any additional control. Simply by arranging the computation and communication phases, a scheduling algorithm is obtained.

However, to design a practical algorithm, there are many issues to be addressed. First, how many packets should be dispatched to each worker processor to produce the desired pattern? Second, what if the packets are of variable lengths? Thirdly, given a network processor configuration and any packet arrival rate, can we always find a way to schedule packets in such a sequential delivery pattern? Fourthly, how to ensure fair scheduling among multiple flows having different reservations? Rest of the paper attempts to analyze the situations and provide answers to the above questions.

Table I defines the notations that are used throughout the rest of the paper. First, we devise a DBCS algorithm assuming that the workload is perfectly divisible. In the theoretical approach, a load distribution α = (α_1,...,α_i ,..., α_N) is first determined to produce the sequential scheduling pattern in one scheduling round. To further enable dynamic scheduling, we derive expressions for the *minimal batch size* I and the *Batch Size* B based on the maximal possible packet length L. Then, the complete DBCS algorithm is designed to schedule packets over multiple scheduling rounds with a care to always keep the sequential delivery of multiple batches of data. The scalability of the DBCS algorithm is analyzed and important conclusion reaches. To schedule variable length packets, a packetized version of the DBCS algorithm (P-DBCS) is devised based on the combination of DRR and SRR. In this way, we always minimize the absolute difference between the actual load distribution and the ideal load distribution. Finally, the P-DBCS algorithm is extended to handle multiple flows having reservations.

III. Dynamic Batch CoScheduling

In this section, we build a theoretical model for the general packet scheduling problem in an NP, and develop a load scheduling algorithm Dynamic Batch CoScheduling (DBCS) to produce a scheduling pattern described in Figure 2. There are two design goals of DBCS. The first is to ensure load balancing for a group of heterogeneous processors. The second is to ensure strict in-order delivery of data. In the following derivation, we assume that the input load is divisible at the granularity of one byte.

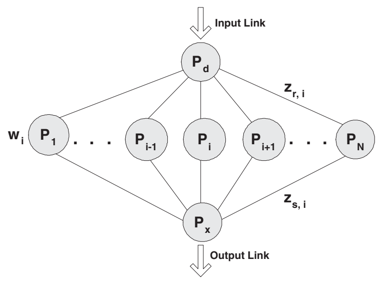


Fig. 3. Load Scheduling Model

A. Load Distribution in A Single Batch

For an NP, we set up the following mathematical model for the ease of theoretical analysis. As shown in Figure 3, there are $(N + 2)$ processors and $2N$ links inside the router. The worker processors P_1, P_2,...,P_N are connected to the dispatching processor P_d via links $l_{r,1}$, $l_{r,2}$,...,$l_{r,N}$. Meanwhile, each worker processor has a direct link $l_{s,i}$ to the transmitting processor P_x. The dispatching processor receives packets from the input link, divides the input load into N parts and then distributes these load fractions to the corresponding worker processor. Each worker processor P_i starts processing immediately upon receiving its load fraction α_i and continues to do so until this fraction is finished. Finally, P_i sends the

TABLE I
NOTATIONS AND TERMINOLOGY

Symbol	Definition of the Symbol
I	The minimal schedulable batch size measured in terms of bytes.
L	The maximal possible packet length that may arrive at an NP.
B	The batch size, i.e., the number of bytes that can be scheduled in a round. It is a multiple of the minimal schedulable batch size. Thus, $B = mI$, where m is a positive integer.
m	The batch granularity, which represents the ratio of B/I. m is a positive integer. The bigger m is, the coarser the batch granularity.
$N_{saturate}$	The maximal number of worker processors that can be supported by one dispatching processor.
N	The total number of worker processors.
T_{cp}	The processing speed of a standard processor which is measured in terms bytes per second.
T_{cm}	The bandwidth of a standard link which is measured in terms of bytes per second.
$z_{r,i}$	A constant that is inversely proportional to the speed of communication link $l_{r,i}$. Thus, $z_{r,i}T_{cm}$ is the bandwidth of the link $l_{r,i}$.
w_i	A constant that is inversely proportional to the speed of processor P_i. Thus, w_iT_{cp} is the processing speed of P_i.
$z_{s,i}$	A constant that is inversely proportional to the speed of communication link $l_{s,i}$. Thus, $z_{s,i}T_{cm}$ is the bandwidth of the link $l_{s,i}$.
α	This refers to a load distribution of the input load, i.e., α = (α_1, ... ,α_i , ... , α_N), where α_i is the fraction of load assigned to P_i such that $0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{N} \alpha_i = 1$.
B_i	The load scheduled to P_i. Thus, $B_i = \alpha_i B$.
D_i	Time duration P_d takes to dispatch B_i to P_i.
P_i	Time duration P_i takes to process B_i.
T_i	Time duration P_i takes to send B_i to P_x.

processed fraction to the transmitting processor P_x via link $l_{s,i}$.

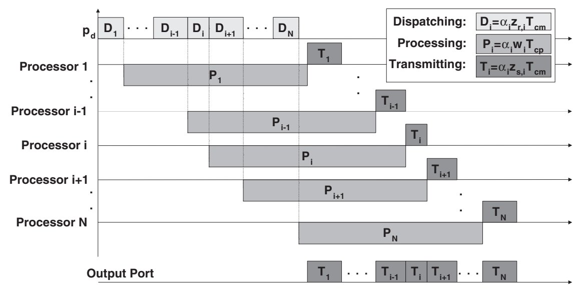


Fig. 4. Load Distribution in A Single Batch

As demonstrated in Figure 4, the load is partitioned among processors such that all the worker processors stop processing sequentially and therefore, deliver the processed fractions to the transmitting processor sequentially. That is, worker processor P_{i+1} starts delivering the packets to the output port only after and right after P_i completes its delivery. Thus, all the packets are processed in parallel and are sent to the output port in order without any break. In this way, we eliminate the out-of-order packet delivery. To achieve such a sequential delivery pattern, we obtain the following recursive equations for $i = 1, ..., N-1$ from the timing diagram:

$$\alpha_i w_i T_{cp} + \alpha_i z_{s,i} T_{cm} = \alpha_{i+1} z_{r,i+1} T_{cm} + \alpha_{i+1} w_{i+1} T_{cp} \tag{1}$$

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

$$\alpha_i = \alpha_N \prod_{v=i}^{N-1} f_v, \ i = 1, ..., N \tag{2}$$

$$\text{where, } \alpha_N = \frac{1}{1 + \sum_{u=1}^{N-1} \prod_{v=u}^{N-1} f_v}, f_v = \frac{z_{r,v+1} T_{cm} + w_{v+1} T_{cp}}{w_v T_{cp} + z_{s,v} T_{cm}}, \tag{3}$$

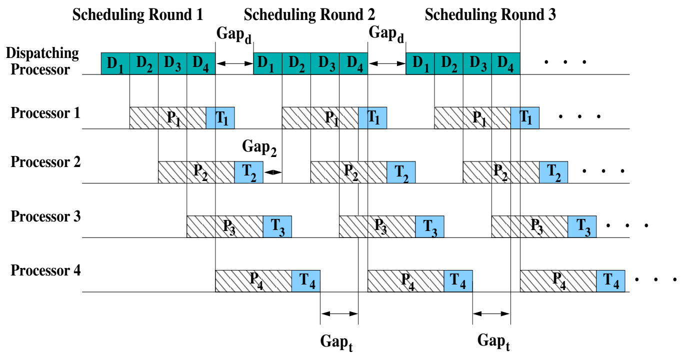


Fig. 5. Timing Diagram of Dynamic Batch CoScheduling

B. Batch Size Determination

As presented in section III-A, we can establish a partition of any given load S, calculated as $(S_1, S_2, ..., S_N)$ and $S_i = \alpha_i S$ $(i = 1, 2, ..., N)$, to ensure both load balancing and sequential data delivery in a single scheduling round. In this section, we design a dynamic packet scheduling algorithm for NPs based on this observation. For an NP to schedule packets, it repeats the sequential scheduling pattern over time while packets arrive dynamically, as illustrated in Figure 5. There are infinite number of scheduling rounds. Within each round, load balancing and sequential delivery are ensured for a batch of packets. To design such a dynamic scheduling algorithm, we first need to determine a feasible batch size for each scheduling round, i.e., how many packets should be scheduled in each round.

To find a feasible batch size, we note that at least one packet should be dispatched to each processor in a scheduling round. Therefore, we define an important system parameter *minimal schedulable batch size* I as the total bytes that should be scheduled in one scheduling round. Suppose the maximal possible length of a packet (in terms of bytes) that may arrive at the NP is L, the *minimal schedulable batch size* I is defined to be

$$I = CL \tag{4}$$

where C is the minimal positive integer such that

$$min_i(\alpha_i \times CL) \geq L \tag{5}$$

Equation 5 guarantees that at least one packet fits into the load fraction that can be dispatched to a worker processor. Hence, CL constitutes a minimal load size that should be guaranteed for one scheduling round. Given I as the minimal batch size, the *batch size* B is set to be a multiple of I as follows:

$$B = mI \tag{6}$$

where m is a positive integer referred as *batch granularity*. In the following section, we will see how the *batch granularity* m affects the system throughput given the minimal schedulable batch size I.

Now, we can do dynamic scheduling as follows: The load fraction B_i that should be dispatched to the ith worker processor P_i is calculated as $B_i = \alpha_i B$. As long as the packets arrive at a rate such that there are always B_i bytes to be scheduled to the current chosen processor i, the scheduling can run smoothly. So the dynamic scheduling algorithm can be applied to a non-backlogged system in an overload situation as well. However, in the following discussion, we assume a continuously backlogged system for the ease of discussion.

C. Scheduling Time Determination

In this section, we determine the best scheduling time of each round. There are two questions to be answered. First, is there any resource conflict between two adjacent scheduling rounds if we schedule multiple rounds continuously? Second, how to resolve the potential resource conflict if there is one?

Note that the basic requirement of a sequential delivery pattern that consists of multiple rounds is: the delivery of two adjacent batches cannot be overlapped, i.e., the delivery of a batch cannot start until the delivery of its previous batch completes. So when we schedule load in multiple rounds, gaps may be introduced between the initiation of two adjacent batches, because 1) the dispatching processor P_d cannot initiate a new batch until the first worker processor completes its T-step. 2) if the initiation of a new batch is too early, it may result in overlapped data delivery between two consecutive batches. Actually, during the time duration of this gap, the dispatching processor P_d has to wait even if it is idle. We denotes this idle time duration of P_d as Gap_d, as shown in Figure 5. Similarly, gaps may exist between the delivery of two adjacent batches, because the first worker processor cannot start transmitting packets until its T-step completes. During the time duration of this gap, the transmitting processor P_x has to be idle. We denote this gap as Gap_t, as shown in Figure 5. In addition, gaps may be introduced between the processing of two adjacent batches on a worker processor, because the processor cannot start receiving data until the dispatching processor initiates a new batch. During the time duration of this gap, the worker processor has to be idle. We denote this gap on the ith worker processor P_i as Gap_i. Assume that the batch size is uniform over all scheduling rounds, from the timing diagram, the following equations are obtained for a heterogeneous system:

$$\begin{aligned} Gap_d &\geq max_i(D_i + P_i + T_i) - \Sigma_{k=1}^{N} D_k && (7)\\ Gap_t &= \Sigma_{k=1}^{N} D_k + Gap_d - \Sigma_{k=1}^{N} T_k && (8)\\ Gap_i &= \Sigma_{k=1}^{N} D_k + Gap_d - (D_i + P_i + T_i) && (9) \end{aligned}$$

For a feasible scheduling pattern, none of Gap_d, Gap_t and Gap_i $(i = 1, 2, ..., N)$ can be negative. Based on this rationale, we derive the following constraints:

$$\begin{aligned} Gap_d^{min} &= \max((\max_i(D_i + P_i + T_i) - \Sigma_{k=1}^{N} D_k),\\ &\qquad (\Sigma_{k=1}^{N} T_k - \Sigma_{k=1}^{N} D_k)) && (10)\\ Gap_t^{min} &= Gap_d^{min} + \Sigma_{k=1}^{N} D_k - \Sigma_{k=1}^{N} T_k && (11)\\ Gap_i^{min} &= Gap_d^{min} + \Sigma_{k=1}^{i-1} D_k + \Sigma_{k=i+1}^{N} D_k - (P_i + T_i) && (12) \end{aligned}$$

Hence, in a heterogeneous system, as long as the dispatching processor maintains a gap Gap_d^{min} between two adjacent batches, a dynamic scheduling algorithm that ensures both sequential delivery and load balancing is obtained.

In a homogeneous system, we denote $z_{s,i} = z_{r,i} = z$ and $w_i = w$ for $i = 1, 2, ..., N$. According to the equations 2 and 3, $\alpha_i = 1/N$ for $i = 1, 2, ..., N$ in a homogeneous system. According to equations 4 and 5, the minimal schedulable batch size is calculated to be $I = NL$. The batch size B is then set to be $B = mI = mNL$ where m is a positive integer. Hence, the load dispatched to a worker processor in a single batch can be uniformly denoted as $B/N = mL$. Therefore, for a homogeneous system, the equation 10, 11 and 12 can be simplified as:

$$\begin{aligned} Gap_d^{min} &= max(((w + 2z)mL - zmNL), 0) && (13)\\ Gap_t^{min} &= Gap_d^{min} && (14)\\ Gap_i^{min} &= max(Gap_d^{min} + zmNL - (w + 2z)mL, 0) && (15) \end{aligned}$$

Equations 13, 14 and 15 suggest that

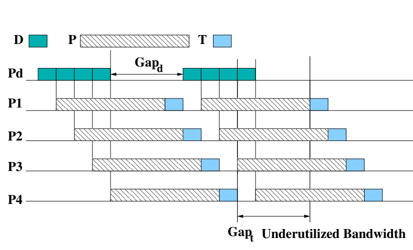
$$N_{saturate} = w/z + 2 \tag{16}$$

$$Gap_d = Gap_t = \begin{cases} (w + 2z)mL - zmNL & N < N_{saturate} \\ 0 & N \geq N_{saturate} \end{cases}$$

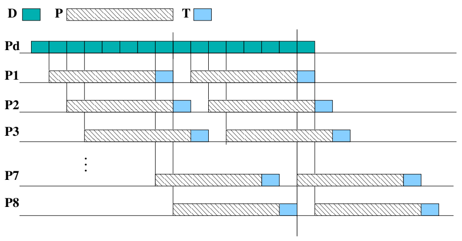
$$Gap_i = \begin{cases} zmNL - (w + 2z)mL & N > N_{saturate} \\ 0 & N \leq N_{saturate} \end{cases}$$

D. Scalability Analysis

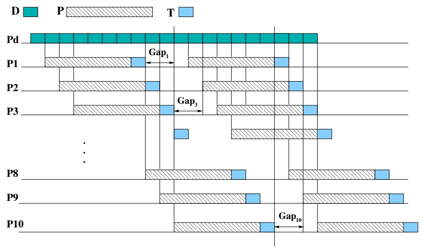
According to the analysis presented in the above subsection, the best scheduling time in a homogeneous system is determined by the number of processors N, the maximal packet length L and the batch granularity m. Let there be N homogeneous worker processors, and $N_{saturate}$ be calculated according to the network processor parameters (w, z) as $N_{saturate} = w/z + 2$. 1) If $N < N_{saturate}$, a gap should be introduced between the initiation of two adjacent batches. The length of the gap is defined as Gap_d in the equation 13. Figure 6(a) demonstrates a scheduling example. In this example, we assume $w = 6\mu s/byte$ and $z = s = 1\mu s/byte$, hence, $N_{saturate} = 8$. For the ease of discussion, this set of



(a) N=4 Gap_d=Gap_t >0, Gap_i=0

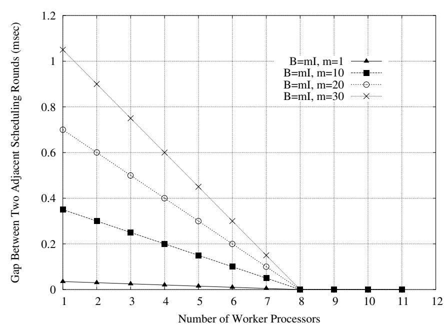


(b) N=8 Gap_d=Gap_t=Gap_i=0

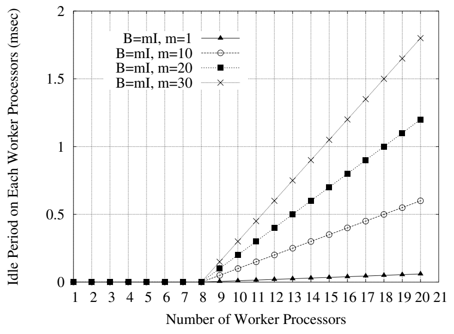


(c) N=10 Gap_i >0, Gap_d=Gap_t=0

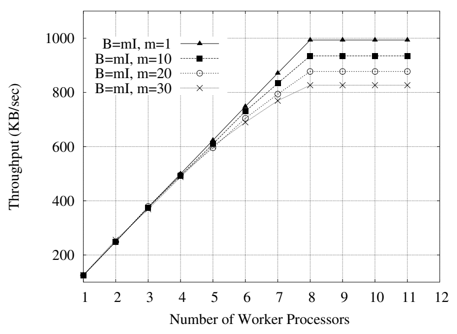
Fig. 6. Scheduling Time Determination in a Homogeneous System



(a) Gap_d and Gap_t between Two Adjacent Scheduling Rounds



(b) Gap_i on Each Worker Processor



(c) Scalability of the Throughput

Fig. 7. Scalability Analysis

values is assumed for w, z and s throughout the rest of the paper. 2) If N equals to $N_{saturate}$, adjacent scheduling rounds are initiated continuously without introducing gaps in between. A perfect continuous scheduling pattern can be achieved, as shown in Figure 6(b). 3) If $N > N_{saturate}$, no gap should be introduced between two adjacent scheduling rounds. However, Gap_i will naturally appear on each worker processor, as shown in Figure 6(c).

Hence, we reach very interesting conclusions about the system scalability. 1) If there are less than $N_{saturate}$ worker processors, as shown in Figure 6(a), there are idle periods on both dispatching processor and the transmitting processor to ensure sequential delivery. In this case, the processing rate cannot catch up with the transmission rate and causes the communication processors to wait for the worker processor to complete processing. 2) If there are exactly $N_{saturate}$ worker processors, all processors are fully utilized and there is no break between the delivery of adjacent batches, as shown in Figure 6(b). In this case, the communication rate and the processing rate perfectly match to give the best performance. 3) If there are more than $N_{saturate}$ worker processors, as shown in Figure 6(c), there are idle periods on each worker processor instead on the dispatching processor. In this case, the communication rate cannot catch up with the processing rate. Hence, some worker processors complete processing before the dispatching processing completes dispatching the whole round of data! Obviously, the computation resources are underutilized because the single dispatching processor becomes the bottleneck.

Therefore, $N_{saturate}$ represents the optimal number of worker processors that one dispatching processor can support. Note that $N_{saturate}$ is determined only by the computation/communication rate, w/z, independent of the batch size. Hence, for a given network processor with the configuration parameter defined as (w, z) and assume that processing always consumes much more time than the communication, i.e. $w > z$, there always exists an optimal number of worker processors that one dispatching processor can support. If there are more than $N_{saturate}$ worker processors, one more dispatching processor should be adopted to schedule load among those surplus worker processors to achieve good system performance.

Figure 7(a) and 7(b) plot the variation of the length of Gap_d and Gap_i as a function of the number of processors N, given the maximal packet length L as $5KB$. Gap_d decreases linearly as the number of processors increases. For a given number of processors, Gap_d increases linearly as the batch granularity increases. The Gap_i on the worker processors increases linearly as the number of processors or the batch granularity increases. Clearly, both gap lengths are minimized when the minimal batch size I is adopted.

Figure 7(c) shows the variation of the system throughput (in terms of KB/sec) as a function of the number of worker processors N, batch size B and batch granularity m. Obviously, $N_{saturate}$ represents the saturating point of the system throughput. The system throughput is flatten out as the number of worker processor exceeds $N_{saturate}$. It is caused by the increasing idle time appearing on the worker processors as the number of processors increases, which is illustrated by Figure 7(b). In addition, as the batch granularity m increases, the system throughput decreases. This degradation can be illustrated by increasing gap length as the batch size increases. Therefore, *the minimal schedulable batch size is the optimal batch size in terms of maximizing the system throughput.* In our scheduling algorithm, we define the minimal schedulable batch size I as a function of the maximal packet length L. In this way, the minimal schedulable batch size is always adapted to the packet size, thus achieves the optimal system throughput.

IV. Packetized Dynamic Batch CoScheduling

In the previous section, a dynamic scheduling algorithm DBCS has been designed for an NP to ensure both load balancing and perfect in-order delivery. However, we have assumed so far that the load is divisible at the granularity of one byte. In practice, we have to schedule workload at the granularity of one packet, and the packets may be of variable lengths. To schedule variable length packets in an NP, we design a Packetized version for the DBCS algorithm (P-DBCS) in this section.

As discussed in section III, according to the DBCS algorithm, given a batch size B, the ideal load distribution among multiple processors in any scheduling round is calculated as $(B_1, B_2, B_3, ..., B_N)$ where $B_i = \alpha_i B$ for $(i = 1, 2, ..., N)$. Now, we discuss how to schedule the real workload consisting of variable length packets using this load distribution.

Because the workload contains variable length packets, the load we actually schedule for the worker processors from P_1 through P_N may not be exactly $(B_1, B_2, ..., B_N)$. Let us denote the actual load distribution as $(\tilde{B_1}, \tilde{B_2}, ..., \tilde{B_N})$. To guarantee the desired sequential delivery, the actual load distribution $(\tilde{B_1}, \tilde{B_2}, ..., \tilde{B_N})$ should match the theoretical value $(B_1, B_2, ..., B_N)$ as close as possible. The design goal of the P-DBCS algorithm is to minimize the discrepancy between $(B_1, B_2, ..., B_N)$ and $(\tilde{B_1}, \tilde{B_2}, ..., \tilde{B_N})$. We define the discrepancy as $\sum_{k=1}^{N}(\tilde{B_i} - B_i)^2$.

We propose the P-DBCS algorithm as follows. The NP packet scheduler first calculates $(B_1, B_2, ..., B_N)$ to determine the ideal number of bytes that should be dispatched to each worker processor in a scheduling round. It also uses a pointer, denoted as $which$, to point to the processor that is currently receiving packets. So, B_{which} represents the ideal number of bytes that should be dispatched to the processor P_{which}. As the scheduling proceeds, the size of the packets that have been dispatched to P_{which} is deducted from B_{which}. The remaining load is recorded in a variable $Balance_{which}$. Ideally, the packet scheduler will dispatch at most $Balance_{which}$ bytes to the processor P_{which} before it changes the pointer $which$ to point to the next processor. However, because the packets are of variable lengths, it may not be possible for a packet to exactly fit into $Balance_{which}$. If the packet size exceeds $Balance_{which}$, the scheduler must make a decision as either to dispatch the packet or not . The dispatching decision is made as follows. Let the packet size be $PacketSize$, if $PacketSize < Balance_{which}$, the packet is scheduled to the processor P_{which}. Otherwise, it is scheduled only if

$$(PacketSize - Balance_i) \leq Balance_i \qquad (17)$$

Packetized Dynamic Batch CoScheduling

Parameter Initialization:

(1) Using equations (2) and (3), determine $\alpha_i, \ \forall i$.
(2) Using equations (4) and (5), determine the minimal batch size.
(3) Set the batch size B as a multiple of I. Calculate the number of bytes that should be scheduled to each worker processor in a scheduling round. $B_i = \alpha_i B, \ \forall i$.
(4) Using equation (10), calculate the gap Gap_d that should be adopted between the initiation of two adjacent batches.

Packet Scheduling:

```
$which \leftarrow 1$
$Balance_{which} \leftarrow B_{which}$
while ( true )
    $PSize \leftarrow$ size of the head-of-line packet
    if ($Balance_{which} \geq PacketSize/2$)
        Dispatch the head-of-line packet to $P_{which}$
        $Balance_{which} \leftarrow Balance_{which} - PacketSize$
    else
        $B_{which} \leftarrow B_{which} + Balance_{which}$
        if ( $which == N$ )
            $which \leftarrow 1$
            $Balance_{which} \leftarrow B_{which}$
            wait for $Gap_d$ seconds
        else
            $which \leftarrow which + 1$
            $Balance_{which} \leftarrow B_{which}$
```

Fig. 8. Packetized Dynamic Batch CoScheduling Algorithm

The rationale behind the algorithm is that: in equation 17, the left-hand represents the absolute value of $|Balance_{which}|$ when the packet is scheduled; and the right-hand represents the absolute value of $|Balance_{which}|$ when the packet is not scheduled. Therefore, we always make a decision to minimize the absolute value of $Balance_{which}$. Because $|Balance_{which}| = |\tilde{B_i}\text{-}B_i|$, the absolute difference between the actually scheduled load and the ideal load, measured as $\sum_{k=1}^{N}(\tilde{B_i} - B_i)^2$, is also minimized. Note that, equation 17 can be simplified as $Balance_i \geq PacketSize/2$.

Compared with two other well-known packet management policies, Surplus Round Robin (SRR) and Deficit Round Robin (DRR), our scheme improves over both by taking their combination. In each scheduling round, the absolute difference between the actual scheduled load and the ideal load, denoted as $|\tilde{B}_i$-$B_i|$, is bounded by the maximal packet length L in either DRR or SRR scheme; while it is bounded by half of the maximal packet length $L/2$ in our scheme.

As discussed above, we have minimized $\sum_{k=1}^{N}(\tilde{B}_i - B_i)^2$ for a single scheduling round. Over multiple scheduling rounds, the deviation of the actual load from the idea load may be accumulated to be a bigger and bigger deviation from the ideal load. To avoid this, the ideal load B_{which} for the next scheduling round is always adjusted as $B_{which} = B_{which} + Balance_{which}$ at the end of each scheduling round. The proposed P-DBCS algorithm is presented in Figure 8.

V. Fair Scheduling among Multiple Flows

In section IV, a P-DBCS algorithm is developed to schedule incoming packets among multiple processors. It not only achieves load balancing in the presence of variable length packets, but also ensures the minimal out-of-order departure of processed packets. In the above discussion, we have assumed so far that the incoming packets are all treated equally. However, in the practice, the incoming packets may belong to different network flows that have made different reservations. In this section, we extend the P-DBCS algorithm to provide fair scheduling among multiple flows. We assume that all flows are continuously backlogged.

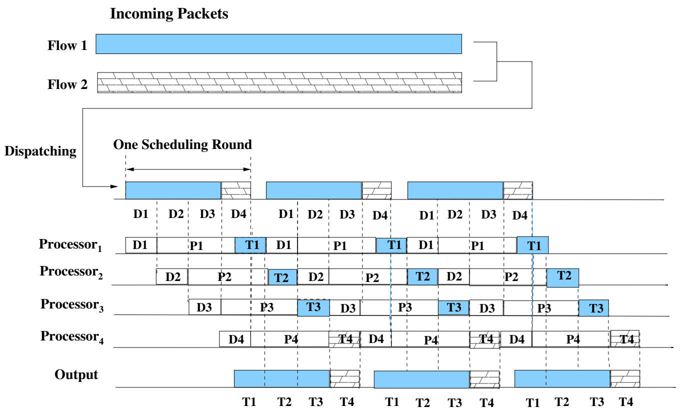


Fig. 9. Fair Scheduling Among Multiple Flows

Let there be M flows, each flow has made a reservation r_i and the inequation $\sum_{i=1}^{M} r_i \leq 1$ holds. We aim to service the packets of different flows at a rate that is proportional to their reservations. As shown in Figure 9, the incoming packets belong to two different flows, with their reservations defined as $(r_1, r_2) = (0.75, 0.15)$. To guarantee that flow 1 is serviced three times faster than flow 2, we need to schedule the packets as follows: in each scheduling round, the total number of bytes that are processed for flow 1 is three times that of the flow 2. Note that the P-DBCS algorithm has a nice property: *the scheduling order of the packets is maintained at the output link*. Therefore, as long as the packets of different flows are scheduled for processing proportionally to their reservations, the processed packets depart orderly and proportionally to their reservations, as illustrated by Figure 9.

To extend the P-DBCS algorithm to handle multiple flows, the basic idea is as follows: Given the batch size B and the flow reservations $(r_1, r_2, ..., r_M)$, the number of bytes that are scheduled in each round for flow i $(i = 1, 2, ..., M)$, denoted as F_i, is

$$F_i = r_i \times B \tag{18}$$

To perform a practical scheduling, we must guarantee that at least one packet is dispatched from any one flow. Hence, one more constraint must be set for the minimal schedulable batch size I as follows:

$$min_i(r_i \times I) \geq L \tag{19}$$

Therefore, I should be redefined as $I = CL$ such that C is the minimal integer that satisfies

$$min_i(r_i \times CL) \geq L \;\; and \;\; min_i(\alpha_i \times CL) \geq L \tag{20}$$

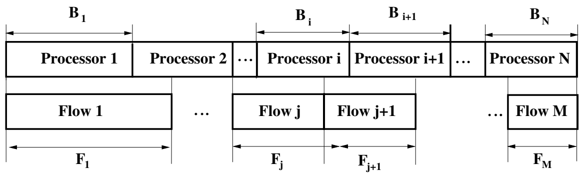


Fig. 10. Allocating Multiple Flows to Worker Processors

Once I is determined, B is set as a multiple of I. Note that F_i denotes the bytes that should be scheduled from a flow in a scheduling round. But the packets may not fit into F_i due to their variable lengths. Similarly, we can adopt the packet management policy described in the above section to deal with variable length packets. Let $FBalance_j$ be the remaining number of bytes that should be dispatched to flow j, a packet with the size $PacketSize$ is dispatched only if $FBalance_j \geq PacketSize/2$.

Let the load distribution among M flows be $(F_1, F_2, ..., F_M)$ and the load distribution among N servers be $(B_1, B_2, ..., B_N)$. We establish an allocation between flows and processors by sorting all processors in decreasing order of their B_is and all flows in decreasing order of their F_js. As shown in Figure 10, the mapping between the sorted B_is and the F_js determines an allocation of flows to the processors. *This allocation specifies the visiting order of both processors and flows.*

When the packet scheduler works, it keeps two pointers: one pointer, i, points to the processor that is currently receiving packets; the other pointer, j, points to the flow that is currently being serviced. The scheduler also keeps two balance counters: one counter, $Balance_i$, records the remaining number of bytes that should be dispatched to the processor i; the other counter, $FBalance_j$, records the remaining number of bytes that should be serviced for the flow j. To achieve both load balance among processors and fair scheduling among multiple flows, when the packet scheduler looks at the head-of-line packet of flow j, it compares $PacketSize$ to both $Balance_i$ and $FBalance_j$. The packet is scheduled only if

$$min(Balance_i, FBalance_j) \geq PacketSize/2 \qquad (21)$$

The algorithm to handle multiple flows using P-DBCS is illustrated in Figure 11. The algorithm achieves both load balancing among processors and fair scheduling among flows.

Fair Scheduling of Multiple Flows Using P-DBCS

Parameter Initialization:

(1) Using equations (2) and (3), determine α_i, $\forall i$.
(2) Using equation (20), determine the minimal batch size I.
(3) Set the batch size B as a multiple of I. Determine the number of bytes that should be scheduled to each worker processor in a scheduling round. $B_i = \alpha_i B$, $\forall i$.
(4) Using equation (10), determine the gap Gap_d that should be adopted between the initiation of two adjacent batches .
(5) Determine the number of bytes that should be scheduled from each flow. $F_j = r_j B$, $\forall j$.
(6) Sort all B_is and all F_js in their decreasing order such that $B_1 \geq B_2 \geq B_3 ... \geq B_N$ and $F_1 \geq F_2 \geq F_3 ... \geq F_M$.

Packet Scheduling:

```
$i \leftarrow 1$   /*Let $i$ point to processor 1*/
$j \leftarrow 1$   /*Let $j$ point to flow 1*/
$Balance_i \leftarrow B_i$
$FBalance_j \leftarrow F_j$
while ( true )
    $PSize \leftarrow$ size of the head-of-line packet of flow $j$
    if ($Balance_i < PSize/2$)
        $B_i \leftarrow B_i + Balance_i$
        if ( $i == N$ )
            $i \leftarrow 1$   /*let $i$ point to the first processor*/
            $Balance_i \leftarrow B_i$
            wait for $Gap_d$ seconds
        else
            $i \leftarrow i + 1$   /*let $i$ point to the next processor*/
            $Balance_i \leftarrow B_i$
    if ($FBalance_j \geq PSize/2$ and $Balance_i \geq PSize/2$)
        Dispatch the head-of-line packet of flow $j$ to $P_i$
        $Balance_i \leftarrow Balance_i - PSize$
        $FBalance_j \leftarrow FBalance_j - PSize$
    if ($FBalance_j < PSize/2$)
        $F_j \leftarrow F_j + FBalance_j$
        $j \leftarrow (j + 1)\%M$   /*let $j$ point to the next flow*/
        $FBalance_j \leftarrow F_j$
```

Fig. 11. Fair Scheduling of Multiple Flows Using P-DBCS

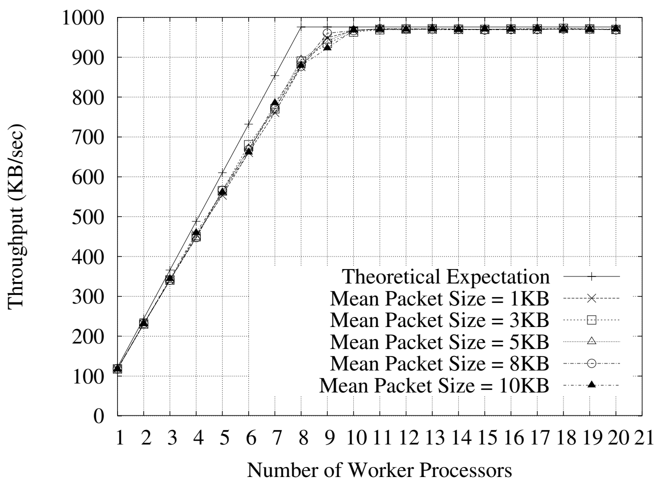
VI. Simulation Results and Discussion

In this section, we verify the analytical results, given in the earlier sections, through rigorous simulations. First, we study performance of P-DBCS and highlight certain intrinsic advantages of P-DBCS. Next, to explicitly show the performance gain expected by P-DBCS, we compare P-DBCS with several other strategies proposed in the literature. Lastly, we demonstrate the effectiveness of P-DBCS to handle multiple incoming flows with different reservations.

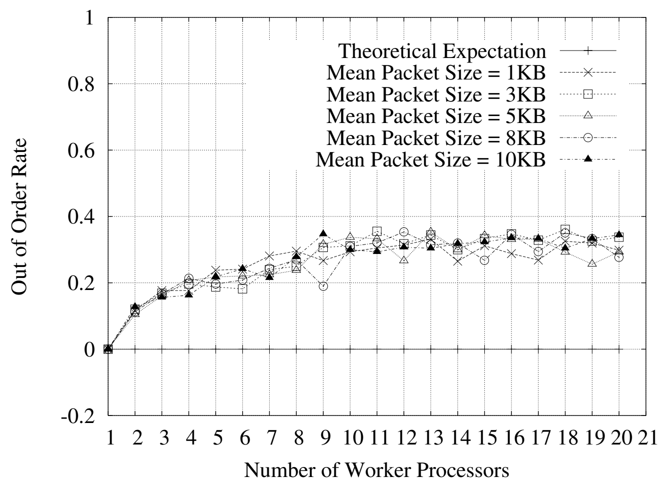
We developed a C simulator to model a heterogeneous NP system with one dispatching processor, multiple worker processors and one transmitting processor. This simulator is designed to process variable length packets in an NP. The packet size of the incoming flow is randomly determined by the exponential distribution. For simplicity, we assume a homogeneous backlogged system. Given that the communication bandwidth of a Intel IXP2400 Network Processor is 100Mbits/sec and considering some other foreseeable software overheads, we configure the system using the following parameters in our simulation. That is, $z_{r,i}T_{cm} = 1\mu s/byte$, $w_i T_{cp} = 6\mu s/byte$, $z_{s,i}T_{cm} = 1\mu s/byte$, $\forall i$. We use these values in the theoretical derivation of our scheduling strategy and strictly apply them in the simulation runs. The mean value of the packet size is varied from 1KB to 10KB and the number of worker processors N is varied from 1 to 20 to observe the performance.

A. Performance of the Packetized DBCS Algorithm

To observe the performance of P-DBCS for processing variable length packets in a realistic system, we conduct the simulation by varying the mean packet size of the incoming packets from 1KB to 10KB. As suggested by the scheduling algorithm in Figure 8, we use the maximum packet size of the flow to determine the optimal batch size. The simulation results are compared with the analytical results obtained in Section III-D. Figures 12(a) and 12(b) show the variation of the throughput and out-of-order rate as a function of the number of worker processors. In both figures, the analytical result is drawn as the theoretical expectation for the ease of comparison. Note that the theoretically expected throughput in Figure 12(a) is originally illustrated by Figure 7(c) in Section III-D. Clearly, the throughput of variable packet mean sizes closely matches the analytical result whereas the out-of-order rate is relatively higher than what is expected by the theory. However, this discrepancy is reasonable because the theoretical load distribution cannot be strictly guaranteed in the presence of variable length packets. As a result, the sequential delivery pattern is disturbed and out-of-order delivery occurs. In Figure 12(a) and 12(b), the curves of different mean packet sizes are twisted with each other, indicating the adaptation capability of P-DBCS to different packet size. It is also interesting to note that the saturating point of throughput occurs when there are 10 worker processors in the realistic system, while the theoretically proved saturating point is 8. This discrepancy is caused by the presence of variable packet lengths that lead to deviation of the actual load distribution from the ideal load distribution. After the saturating point, when more processors are used, the out-of-order rate tends to increase. Therefore, we need to select a proper number of processors that guarantees high throughput and incurs tolerable out-of-order rate.

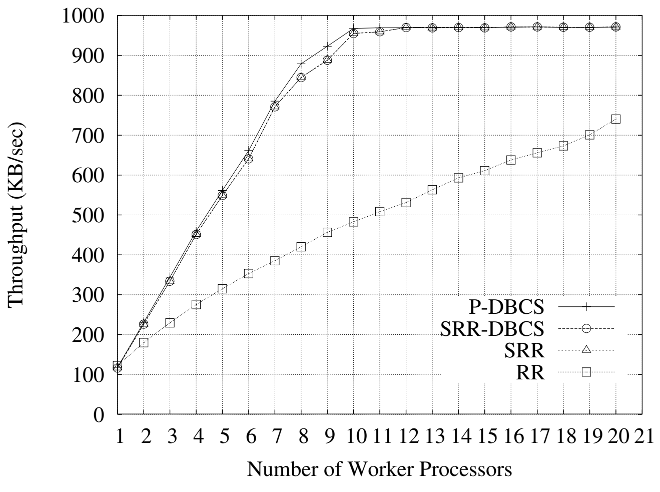


(a) Throughput Verification of P-DBCS

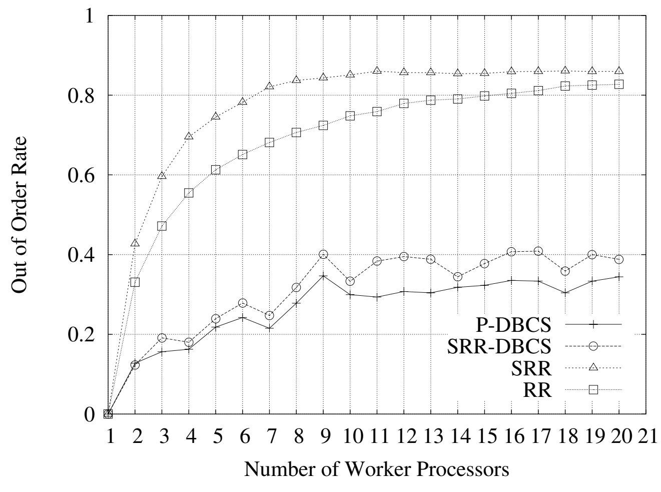


(b) Out-of-order Rate Verification of P-DBCS

Fig. 12. Experimental Verification of P-DBCS



(a) Throughput of Different Scheduling Schemes



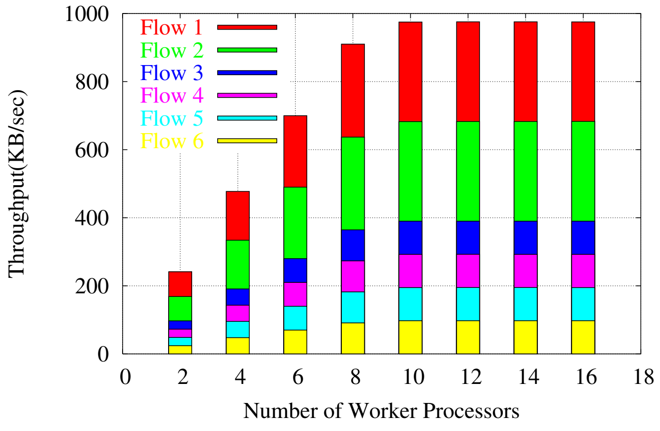
(b) Out-of-order Rate of Different Scheduling Schemes

Fig. 13. Experimental Comparison among Different Scheduling Schemes

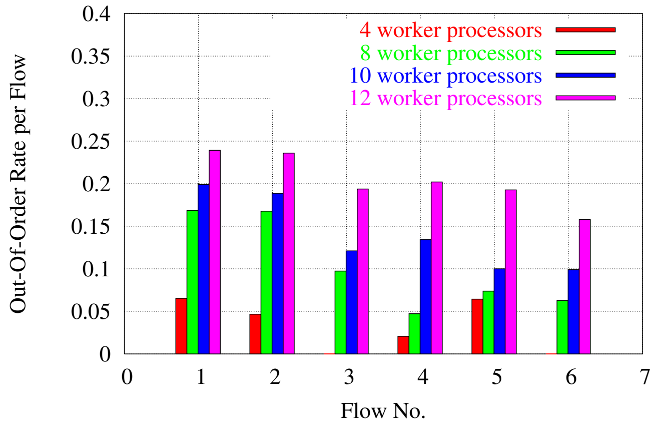
B. Performance Comparison with Other Schemes

Now, let us compare P-DBCS with several other load sharing schemes, namely, SRR-DBCS, SRR [8] and ordinary Round Robin (RR) [8], [1]. SRR-DBCS is a combined strategy of SRR and DBCS. In this scheme, we distribute the load among processors according to our DBCS algorithm but apply SRR to deal with variable length packets. In other words, P-DBCS and SRR-DBCS are two different packetized versions of DBCS. The ideas of SRR and RR are implemented as specified in the literature. With the RR scheme, one packet is dispatched to each worker processor in each round without considering the variety of packet size and the heterogeneity of processors. With the SRR scheme, the total number of bytes dispatched to each worker processor is proportional to their processing powers. SRR and RR do not consider any communication latency between the dispatching processor and worker processors in their design. To obtain a fair comparison, we introduce the same communication delay $z_{r,i}T_{cm} = 1\mu s/byte$ for each scheme as modeled in P-DBCS.

As shown in Figure 13(a) and 13(b), P-DBCS achieves the highest throughput and the smallest out-of-order rate among all the schemes. P-DBCS and SRR-DBCS greatly reduce the out-of-order rate over SRR and RR because DBCS maintains an in-order delivery pattern while balancing the load, whereas SRR and RR do not consider in-order delivery at all. RR



(a) Throughput of Multiple Flows



(b) Out-of-order Rate of Multiple Flows

Fig. 14. Scheduling Multiple Flows

offers the worst throughput due to the potential load imbalance incurred by blindly dispatching packets. By adapting the load to the heterogeneity of processors, P-DBCS, SRR-DBCS and SRR achieve comparable throughput because good load balancing is ensured. The advantage of P-DBCS over SRR-DBCS can be observed by the relatively lower out-of-order rate. In conclusion, P-DBCS outperforms all other schemes in both load balancing and sequential delivery. While taking extra care to minimize the out-of-order rate, P-DBCS still produces the highest throughput.

C. Fair Scheduling of Multiple Flows Using P-DBCS

Let there be six flows and each flow makes a different reservation, defined as (0.3, 0.3, 0.1, 0.1, 0.1, 0.1). To evaluate the effectiveness of P-DBCS algorithm to provide fair scheduling among multiple flows, we generate different flows at the same arrival rate and observe if the service rate of each flow is controlled by its reservation. Figure 14(a) shows the the total throughput and each flow's throughput as a function of the number of processors. Clearly, the total throughput is fairly shared among the flows. All the six flows are serviced at a rate proportional to their reservations, although they arrive at the same rate. Hence, a fair scheduling among flows is successfully achieved by our algorithm. Note that, while scheduling among multiple flows, the system still produces comparable total throughput as that presented in Figure 12(a), indicating good scalability under heavy workload. Figure 14(b) demonstrates each flow's out-of-order rate as a function of the number of processors. The out-of-order rate per flow slightly increases as the the number of processors increases. The flow with higher reservation tends to have higher out-of-order rate because larger portion of its packets are serviced and involved in the out-of-order delivery caused by variable packet lengths. Overall, the out-of-order rate falls into a small range which is similar to the out-of-order rate presented in Figure 12(b).

VII. Conclusion

In this paper, we proposed an efficient packet scheduling algorithm Packetized Dynamic Batch CoScheduling (P-DBCS) for a heterogeneous network processor system. P-DBCS is capable of scheduling variable length packets among a group of heterogeneous processors to ensure both load balancing and minimal out-of-order packet delivery. P-DBCS is based on the Dynamic Batch CoScheduling (DBCS) algorithm, which we developed from rigorous theoretical derivation by assuming the workload is perfectly divisible. Several important theoretical results were presented in the paper for DBCS algorithm. Extensive sensitivity results are provided through analysis and simulation to show that the proposed algorithms satisfy both the load balancing and in-order requirements in packet processing. To provide fair scheduling among multiple flows, we extend P-DBCS to service packets according to their reservations. Simulation results verified that fair scheduling among multiple flows is successfully achieved. We plan to implement the proposed packet scheduling algorithms in a real network processor, like Intel IXP 2400/2800, to verify the performance.

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] 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.

[6] E. Amir, S. McCanne, and R. Katz, "An active service framework and its application to real-time multimedia transcoding," *ACM SIGCOMM Symp.*, September 1998.

[7] M. Ott, G. Welling, S. Mathur, D. Reininger, and R. Izmailov, "The journey active network model," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 527–537, Mar. 2001.

[8] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable stripping protocol," *SIGCOMM*, pp. 131–141, 1996.

[9] 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.

[10] M. Satyanarayanan, "Scalable, secure, and highly available distributed file access," *IEEE Computer*, May 1990.

[11] L. Kencl and J. Y. L. Boudec, "Adaptive load sharing for network processors," *IEEE INFOCOM*, 2002.

[12] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, "Scheduling and load balancing in parallel and distributed systems," *IEEE CS Press*, 1995.

[13] J. Yao, J. Guo, L. Bhuyan, and Z. Xu, "Scheduling real-time multimedia tasks in network processors," *IEEE GLOBECOM, Dallas,*, November 2004.

[14] Y. C. Cheng and T. G. Robertazzi, "Distributed computation with communication delays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 6, November 1988.

[15] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, "Scheduling divisible loads in parallel and distributed systems," 1996.

[16] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, 1996.