The 28th International Conference on Distributed Computing Systems

Quantum-Adaptive Scheduling for Multi-Core Network Processors

Yue Zhang, Bin Liu, Lei Shi, Jingnan Yao, Laxmi Bhuyan
{zhang-yue, shijim}@mails.tsinghua.edu.cn
liub@tsinghua.edu.cn
{jyao,bhuyan}@cs.ucr.edu

Abstract

Efficiency and effectiveness are always the emphases of a scheduler, for both link and processor scheduling. Well-known scheduling algorithms such as Surplus Round Robin (SRR) and Elastic Round Robin (ERR) suffer from two fold shortcomings : 1) additional pre-processing queuing delay and post-processing resequencing delay are incurred due to the lack of short-term load-balancing; 2) bursty scheduling is caused due to blind preservation of scheduling history under non-backlogged traffic. In this paper, we propose a Quantum-Adaptive Scheduling (QAS) algorithm, which: 1) synchronizes all the quanta in a fine-grained manner and, 2) adjusts the quanta intelligently based on processor utilization. We theoretically prove that the Queuing Fairness Bound (QFB) for QAS is one third tighter than SRR and ERR. This result approaches the optimal value as obtained in Shortest Queue First (SQF) algorithm, while still maintaining O(1) complexity. Trace-driven simulations show that QAS reduces average packet delay by 18%~24% while cutting down the resequencing buffer size by more than 40% compared to SRR and ERR.

1.Introduction

The next generation networks are being designed as supporting service differentiated and application oriented, which transforms the role of traditional network nodes from simple packet forwarding to sophisticated application specific packet processing. Given the processing power limitation of a single processor and the inherent parallelism associated with network traffic, multi-processor architecture is widely adopted in Cavium [1], Intel IXP [2] Network Processor (NP) systems.

With the exponential growth in Internet bandwidth and demand by various real-time applications, such as voice over IP (VoIP) and video on demand (VoD), intelligent scheduling of packets over multi-link and multi-processor network systems becomes increasingly important. Achieving better load balance, higher throughput, minimizing the latency and packet disorder with low scheduling complexity are the key metrics for evaluating a scheduling algorithm. Packet scheduling in the Internet routers is usually based on simple round robin techniques. Deficit Round Robin (DRR) [3], Surplus Round Robin (SRR) [4] and the improved Elastic Round Robin (ERR) [5] scheduling algorithms have been proposed for both link and processor scheduling. Our research shows that although the above algorithms work well for long-term load balance and backlogged traffic, they perform poorly for short-term scheduling and non-backlogged scenario. We explain the problems below[1].

(1) Short-term load imbalance: In the classical DRR and SRR algorithms, traffic is scheduled circularly across all processor engines (PEs) according to the past scheduling load recorded in the deficit quantum. At the beginning of each scheduling round, the quantum of each PE is incremented with a fixed value no smaller than the longest packet processing time. This application dependent longest packet processing time, not being properly bounded, could go beyond tens of times of the shortest packet processing time. Take the following scenario for example: suppose the longest packet processing time is 1500 cycles, while all the arrival packets have a processing time of 64 cycles and belong to the same flow (defined by 5-tuple). Using SRR or DRR, not only are packets scheduled in bursts ([1500/64]=23 consecutive packets to each PE), but also no latter packets can be transmitted until all the backlogged packets at the previous PEs are sent out for preserving the packet order. As such, up to 23(N-1) packets could be blocked for an N-way system in the worst case.

(2) Unfairness under non-backlogged traffic load: SRR or DRR targets exclusively on balancing the scheduling load to each PE, not the queueing load at individual PE. The two targets are inconsistent under non-backlogged case, because if there is no traffic arrival for a while, the heavily scheduled PEs will soon

[1] For consistency all the demonstrations in this paper are based on a multi-core NP model as shown in Figure 1.

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.63

become light-loaded or even idle, meaning that the deficit quantum of all PEs should be re-adjusted or reset to the initial value. Unfortunately, in previous approaches, the residual quanta are always carried over to the next round regardless of the fact that the PEs already run empty, which will degrade the short-term load-balancing.

We present a novel scheduling algorithm, namely Quantum-Adaptive Scheduling (QAS), which could improve the fairness for both the link and the processor scheduling, based on the predictive method [6]. Our contributions are:

1) Based on the criteria of queuing fairness bound (QFB, see Section 2), we propose a Quantum-Adaptive Scheduling algorithm, which improves the load balancing performance in both short-term and long-term, as well as backlogged and non-backlogged traffic conditions;

2) We theoretically prove that, our QAS achieves a QFB bound close to the optimal value as obtained in the Shortest Queue First (SQF) algorithm. Real-trace simulations show that QAS can reduce the average packet delay by 18%~24% and cut down the resequencing buffer size by more than 40% compared to SRR and ERR algorithms, while still maintaining O(1) complexity;

The rest of this paper is organized as follows. In section II, we revisit the fairness definition and survey the related works. In section III we propose our QAS algorithm. Section IV analyzes its load-balancing and delay performance. Section V carries out real-trace driven simulations to reveal its advantages and finally section VI concludes the paper.

2.Fairness Definitions and Related Works

It is proved in [4] that the fair queuing and load sharing algorithms are essentially equivalent. However, the concept of fairness that is defined for the scenario of fair queuing needs to be properly translated to the load sharing scenario. In this section, we redefine the classical fairness criteria for this purpose. Following a detailed discussion of the related works, we highlight our design goals in the problem statement.

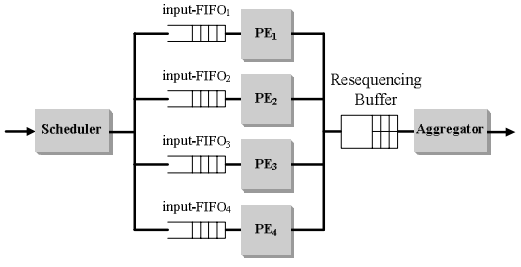

Figure 1. Parallel PE architecture in a multi-core network processor

2.1. Revisiting Different Fairness Criteria in Multi-core Scheduling

Classical scheduling algorithms such as DRR and SRR utilize quantum based mechanism to split the incoming traffic. Relative Fairness Bound (RFB) [3] is used to capture the fairness of such schemes.

Definition 1. Let $Sent_i(t, t+t_0)$ denote the total processing cycles scheduled to PE_i during time period $[t, t+t_0]$. RFB is defined as:

$$\mathrm{RFB} = \max\{|\, Sent_i(t,\ t+t_0) - Sent_j(t,\ t+t_0)\,|\}, \forall i, j, t, t_0$$

Specified in Definition 1, RFB is defined as the maximum difference in the processing cycles scheduled to any two PEs over all possible intervals of time. The objective of hitherto acknowledged quantum based algorithms is to minimize the RFB, which is often considered equivalent to the load-balancing of all the PEs. However, to achieve optimized system performance, the queuing load of each PE, consisting of not only the scheduled cycles but also the actual processed cycles, should be balanced. Hence, we redefine the fairness bound in Definition 2.

Definition 2. Let $Proc_i(t, t+t_0)$ be the total occupied cycles of PE_i during time period $(t, t+t_0)$. The Queuing Fairness Bound (QFB) is defined as:

$$\mathrm{QFB} = \max\{|\,[Sent_i(t,\ t+t_0) - Proc_i(t,\ t+t_0)] - [Sent_j(t,\ t+t_0) - Proc_j(t,\ t+t_0)]\,|\}, \forall i, j, t, t_0$$

In the non-backlogged case, as soon as certain PEs become idle while others remain occupied, QFB is distinguished from RFB due to the distinct *Proc* values. The time interval t_0 can be set to identify separate fairness requirement. When t_0 is large enough, i.e., during which the scheduling can proceed up to a few rounds, QFB reflects long-term fairness. On the other hand, when t_0 is so small that the scheduling lasts less than a single round, QFB reflects short-term fairness. Certainly, long-term load balance is the key to the system throughput. However, short-term imbalance can also incur considerable delay variation, which in turn causes packet congestion in the resequencing buffer.

Take an example NP model of two PEs. Assume that during a particular scheduling round, one packet x with the longest packet processing time M is scheduled to PE_1 and there is no packet arrival for a while. By the time PE_1 finishes processing, $Proc_1=Sent_1=M$, $Sent_2=0$. Then when a series of short packets arrive, since both PEs are idle, they should be served alternately. But if we try to minimize $RFB=Sent_1 - Sent_2=M$, as a result we have to schedule a burst of packets to PE_2. Instead, $QFB = |Sent_1 - Proc_1| - |Sent_2 - Proc_2| = 0$, generates the scheduling more fairly, by precisely capturing the difference of the queuing load.

2.2. Related Scheduling Algorithms

The scheduling issue for parallel systems has been studied intensively in several related areas from switch scheduling to high performance computing. Numerous algorithms have been proposed. Among all the packet based scheduling algorithms, SQF offers the best fairness bound by always scheduling packets to the shortest queue, however loses its attraction in practice due to its $O(\log N)$ complexity. For the same reason, RR based algorithms such as DRR, SRR and ERR become popular due to their simplicity.

DRR [3] is introduced to accommodate variable packet processing time. A practical instantiation of DRR is the well-known SRR [4]. By SRR, each PE maintains a surplus quantum. This quantum is carried over to the next round and incremented with a fixed quantum no smaller than the maximum packet processing time. Such an approach guarantees RFB as in Definition 1 and the $O(1)$ complexity. However, it cannot preserve short-term load-balancing because of the large increment of individual quantum. Specifically, when the total processing time of several consecutive packets is smaller than the longest packet processing time, they may be dispatched to the same PE regardless of other available idle PEs.

In [5], S. Kanhere et al. proposed the ERR, which improves over SRR by introducing a smaller quantum increment of $|P_{min}|+1$ to each PE at the beginning of each scheduling round. P_{min} is defined as the minimal remaining deficit quantum left in the last scheduling round. ERR improves over DRR and ERR for both RFB and QFB by reducing the increment of quantum from the longest packet processing time in all scheduling round M, to the longest packet processing time in the last round m. However, as the parallelism degree of an NP model increases, the performance of ERR will be asymptotic to that of SRR, under which circumstance m will approach M.

J. Guo et al. proposed Packetized Dynamic Batch-CoScheduling (P-DBCS) [7] which applies a combination of DRR and SRR schemes. P-DBCS provides better short-term load-balancing than DRR and SRR. However, since it still follows the fixed increment nature on the deficit quantum, the fairness metrics of QFB remain the same.

There are many other theoretical load-balancing algorithms in the literature. Hash-based algorithms perform well in long-term [8][9], however lead to unpredictable performance in short-term due to the heavy-tailed characteristics of real-life traffic pattern [10]. Generalized Processor Sharing (GPS) related fair queuing algorithms [11][12][13] can also be borrowed for load-balancing [4]. These algorithms ensure good fairness, however are unacceptable considering the complexity of sorting timestamps and computing system-potential function. Though the computing complexity could be reduced to $O(1)$, the sorting inherently incurs $O(\log N)$ complexity [13], which harms the scalability and scheduling speed-up.

2.3. Problem Statement

Given a multi-core NP model, we aim to:

1) Design a scheduling algorithm to minimize QFB, so as to improve the system load-balancing;

2) Preserve $O(1)$ time complexity in actual implementations.

3. Quantum-Adaptive Scheduling

3.1. Scheduling Scenario

Figure 2 details the scheduling model on which QAS based. Each PE_i ($i \in [1,N]$) maintains a deficit quantum, a service register and a utilization quantum, denoted as q_i, s_i, u_i, respectively. q_i records the past unfairness based on which a future scheduling is decided. s_i and u_i record the load dispatched in the last scheduling round for PE_i and its utilization, respectively. The scheduler also maintains a global service pointer *Ptr* indicating the current PE in service, and a global register G recording the common value which should be compensated to every q_i. q_i and u_i are used to compute the precise queuing load for PE_i. s_i and G are used to emulate the aggregative quantum update in a distributed manner.

Incoming packets are dispatched to a certain PE selected by the scheduler. In case that the selected PE is occupied, packets are buffered in an input FIFO queue. After being processed at individual PEs, packets are gathered at the resequencing buffer. To guarantee intra-flow packet order, the technique proposed in [14] is used. QAS is distinguished from previous algorithms in two main aspects: 1) the way it selects the PE to receive a packet; 2) how these quanta are updated after each scheduling, including q_i, s_i, u_i and G.

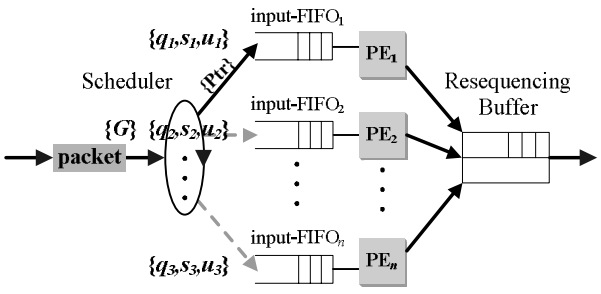

Figure 2. Input scheduler of an NP

3.2. Algorithm Description

QAS algorithm operates in four steps.

1) Initialize: At the beginning, the global register, service pointer, deficit quantum, service register, and the utilization quantum are cleared by G=Ptr=0, q_i=s_i=u_i=0. In each scheduling cycle, Steps 2 through 3 are executed sequentially.

2) Schedule: When packet P with processing time T arrives during round D, in the QAS algorithm, it is scheduled to the PE pointed by Ptr if only $q_{Ptr} \geq 0$. Otherwise if $q_{Ptr} < 0$, Ptr is incremented by one in a round robin manner until the next PE_i with $q_i \geq 0$ is found. Once Ptr moves to the next PE, the following quanta will be updated prior to the scheduling:

$$\begin{aligned} q_{Ptr} &= q_{Ptr} + (G - s_{Ptr}) \\ G &= G - s_{Ptr} \\ s_{Ptr} &= 0 \end{aligned} \tag{1}$$

3) Update: After packet P with processing time T is dispatched to the PE pointed by Ptr, the following counters need to be updated:

$$\begin{aligned} q_{Ptr} &= q_{Ptr} - (T + u_{Ptr}) \\ G &= G + (T + u_{Ptr})/(N-1) \\ s_{Ptr} &= s_{Ptr} + (T + u_{Ptr})/(N-1) \\ u_{Ptr} &= 0 \end{aligned} \tag{2}$$

4) Utilization Statistic: In each processing cycle (different from the scheduling cycle), utilization quantum of each idle PE is updated by $u_i = u_i + 1$. This step is performed by the PEs. The upper bound value of each u_i is set to q_i to avoid negative $q_i - u_i$.

In Step 2), for some cases, service pointer Ptr jumps more than once to skip those heavy-loaded PEs, as called jump-overs of Ptr. A look-ahead window is hereby introduced to restrict it. Setting the size of look-ahead window to 2 indicates that the scheduler checks at most two next deficit quanta q_{Ptr} and q_{Ptr+1} to finish a scheduling, otherwise it will end this turn and continue searching for a nonnegative q_i in the next scheduling cycle. We are able to keep the $O(1)$ time complexity with negligible performance degradation. This is because in each scheduling cycle, the number of ***comparison*** and ***update*** involved is limited and kept independent from the number of PEs. The pseudo code of QAS is shown in Figure 3.

Utilization quantum u_i is used to adjust deficit quantum q_i with the processor load. u_i is independently increased by the corresponding PE_i and reset by the scheduler. Thus, PE_i should either add its utilization quantum u_i in each idle cycle, or avoid in the busy cycle. Hence, this mechanism will not cause much additional overload to PEs. When scheduling for each PE, the scheduler subtracts the utilization quantum from the deficit quantum as suggested in (2). In this way, the discrepancy of PE processing time is merged into the deficit quantum.

s_i and G are N+1 auxiliary registers introduced to achieve $O(1)$ complexity. To describe and understand QAS easier, we consider an equivalent $O(N)$ algorithm S(Simple)-QAS without using s_i and G. In step 1), each q_i, u_i and Ptr are set to 0. In step 2), the PE with $q_{Ptr} \geq 0$ is selected to schedule in a round robin manner with look-ahead window used to bound the number of jump-overs. In step 3), all the q_i, $i \in [1,N]$ are updated as follow: for the PE to which packet P has been scheduled, in the S-QAS algorithm, updating its deficit quantum by $q_{Ptr} = q_{Ptr} - (T + u_{ptr})$ and for the deficit quanta of all the other PEs by $q_j = q_j + (T + u_{ptr})/(N-1)$, for any j, $j \neq Ptr$. Then reset u_{Ptr} to 0. It could be proved that the deficit quantum of the scheduled PE is updated with the same value as in the $O(1)$ scheme. Suppose there are K packets $T_1, T_2, \cdots, T_k$ scheduled to the last $N-1$ PEs before PE_i in the round robin manner. Because the global register G always records the last $N-1$ PEs' total scheduling quantum before, $G = (T_1 + u_1) / (N-1) + \cdots + (T_K + u_K) / (N-1)$. So in S-QAS, updating q_i with each $(T_j + u_j) / (N-1)$, $j \in [1,K]$ is the same as updating with G for one time in QAS.

QAS algorithm

Initialize:
$G = Ptr = 0$ //Global quantum, Service pointer
$q_i = s_i = u_i = 0$ //Deficit / Send / Utilization quantum
Scheduling Cycle:
while(true)
  if ($q_{Ptr} < 0$) //Dispatch the packet to PE Ptr
    //Check the next 2 PEs (look-ahead window = 2)
    for i from 1 to 2
      $Ptr = Ptr + 1$ //Move the pointer
      $q_{Ptr} = q_{Ptr} + G - s_{Ptr}$ //Assign the deficit quantum
      $G = G - s_{Ptr}$
      $s_{Ptr} = 0$
      if($q_{Ptr} >= 0$) //Dispatch the packet to PE Ptr
        break UPDATE
    Continue //Do not schedule in this cycle
  UPDATE:
  T = Packet_Size //Input packet
  $q_{Ptr} = q_{Ptr} > u_{Ptr}$? $q_{Ptr} - u_{Ptr}$: 0 //Avoid extra jump-overs
  $q_{Ptr} = q_{Ptr} - T$ //Update the deficit quantum
  $G = G + (T + u_{Ptr}) / (N-1)$
  $s_{Ptr} = s_{Ptr} + (T + u_{Ptr}) / (N-1)$
  $u_{Ptr} = 0$
Processing Cycle:
while(true)
  if (PE_i idle) //Increase u_i for each idle cycle
    $u_i = u_i + 1$
  else
    process packet //In the busy cycle

Figure 3. Pseudo code of QAS

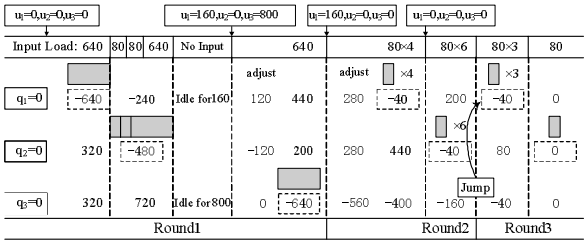

Figure 4. An illustrative example of S-QAS

We describe the S-QAS scheduling on a 3-PE example. An illustrative chart is given in Figure 4. Initially, all the quanta are reset to 0. We can see that the first incoming packet with processing time of 640 is dispatched to PE_1 since $q_1 \geq 0$. Then the quanta are updated to q_1=-640, q_2=q_3=320. After the following batch of packets scheduled to PE_2, the system encountered a period of vacancy for 800 cycles to make u_1=160, u_2=0, u_3=800. The adjustment occurs when the next packet of 640 cycles comes. The quanta are adjusted to q_1=120, q_2=-120, q_3=0 first, then the scheduler continues scheduling. In the second scheduling round, when the service pointer first moves to PE_3, it turns back to PE_1 once it finds q_3=-160<0.

To summarize, two novel mechanisms are having effects in QAS: the fine-grained quantum adaptation and the PE utilization-based adjustment. Compared to DRR, SRR and ERR, QAS' new mechanisms greatly improve short-term load-balancing performance in the following manner: 1) by updating the quantum of all the PEs after each packet scheduling, the scheduling unfairness is well bounded. 2) by introducing the PE utilization quantum, the scheduling emphasis is shifted from balancing the incoming traffic to equalizing the backlogged queue length.

4.Performance Analysis

In this section, we derive fairness, delay properties and work complexity of QAS, compared with SRR and ERR. Table 1 defines the notations used in this section.

Table 1. Notations and terminologies

N	Number of PEs in an NP model
q_i	Deficit quantum of PE_i
$q_{i,k}(h)$	Deficit quantum of PE_i when *Ptr* moves to PE_h during round k
$q'_{i,k}$	Negative quantum value of PE_i after being scheduled during round k
$Sent_i(t,t+t_0)$	The total amount of packets sent to PE_i during time period $[t, t+t_0]$, measured by the processing cycles
M	Maximal PE processing time on each input packet
m	Maximal PE processing time on each input packet scheduled in last round

4.1. Queuing Fairness Bound

From Definition 2, by always considering the queuing load, QFB are same between backlogged and non-backlogged cases. While under backlogged traffic, QFB is equivalent to RFB, which can be derived from the following formulas [5]. In any round k,

$$0 \leq Maxq(k) \leq m-1 \tag{3}$$

$$Sent_i(t,t+t_0) \leq n_i + \sum_{k=r_i-1}^{r_i+n_i-2} Maxq(k) + (m-1) \tag{4}$$

$$\begin{aligned} Sent_i(t,t+t_0) - Sent_j(t,t+t_0) &\leq 1 + 2(m-1) \\ &+ \sum_{k=r_i-1}^{r_i+n_i-2} Maxq(k) - \sum_{k=r_j-1}^{r_j+n_j-2} Maxq(k) \end{aligned} \tag{5}$$

From (3-5) we deduce the QFB of ERR as $3m$:

$$Sent_i(t,t+t_0) - Sent_j(t,t+t_0) \leq 3m \tag{6}$$

To describe concisely, we consider the QFB of S-QAS which has the same scheduling sequence with QAS but updates the deficit quantum in an $O(N)$ manner. Let $Minq_k$=$-m$ be the minimum quantum after the scheduling to each PE. Because the sum of all positive and negative quanta equals to zero at any time in S-QAS, so the sum of all positive quanta is smaller than $(N-1)m$. Assuming that the last quantum is the largest, defined as $Maxq_k$, which is accumulated with the total scheduling quantum of round k.

$$Maxq_k = q'_{n-1,k-1} + Sent_{1\sim n-1}(k)/(N-1) \tag{7}$$

At the beginning of round k, the sum of all positive quanta is no more than $(N-1)m$. In the worst case, each PE schedules one more packet of processing time m when its quantum drops nearly 0, and the scheduled quantum will be added to other quanta. Then the total scheduling quantum from PE_1 to PE_{n-1} in round k is:

$$\begin{aligned} Sent_{1\sim n-1}(k) &= (N-1)m + (N-1)m \\ &+ 2(N-1)m/(N-1) + 2(N-1)m/(N-1)^2 + \cdots \end{aligned} \tag{8}$$

Ignoring the higher order term in (8), we have:

$$Maxq_k = q'_{n-1,k-1} + 2m < 2m \tag{9}$$

Because $q'_{n-1,k-1}$ must be negative:

$$0 \leq Maxq_k \leq 2m \tag{10}$$

Hence in S-QAS, (4, 6) become (11, 12) respectively:

$$Sent_i(t,\ t+t_0) \leq n_i + \sum_{k=r_i-1}^{r_i+n_i-2} Maxq_k/(N-1) + m \tag{11}$$

$$Sent_i(t,t+t_0) - Sent_j(t,t+t_0) \leq 2mN/(N-1) \tag{12}$$

Note that even SQF with the best QFB cannot make it smaller than $2m$ in such a worst case where the packet with processing time m has been sent to each PE_i, $i \in [1,N-1]$, then two following packets with processing time m-1 and m have to be scheduled to PE_N at time t_2 and t_2+t_0, respectively, so:

$$\left| Sent_i(t_2,t_2+t_0) - Sent_N(t_2,t_2+t_0) \right| = 2m-1 \tag{13}$$

4.2. Start-up Latency

Start-up latency S is defined as the longest duration between the time instant when the first PE is served and the time instant when the last PE is served. It is an important parameter for a scheduling algorithm, because a tight bound on the start-up latency guarantees the service time for each PE to be within an expectable time range, which in turn increases the system throughput. Start-up latency can be expressed with the scheduling speed of r, as in (14, 15) [5]:

$$S_{SRR} \le (M + m - 1)(N - 1)/r + m/r \quad (14)$$

$$S_{ERR} \le (2m - 1)(N - 1)/r + m/r \quad (15)$$

In the first scheduling round of QAS, since all the quanta are equal to 0, $Sent_i(0) \le m$ holds for every PE. In the general situation, $Sent_i(r) \le m + 2m/(N\text{-}1)$. The last PE will be served after N-1 PEs, therefore:

$$S_{QAS} \le Sent_i(r)(N - 1)/r + m/r = (mN + 2m)/r \quad (16)$$

4.3. Look-ahead Window Size

Look-ahead window is used to compromise between the scheduling efficiency and time complexity. Large window size guarantees valid scheduling in each cycle, but incurs more quanta checks. Small window size, on the other hand, helps reduce the complexity.

Lemma 1: The quantum value q'_i is independent and identically distributed. Under the Tri-model distribution assumption for the packet length (processing time) of the Internet traffic, we have:

$$\mathrm{E}(q') = -0.36M \quad (18)$$

Proof: It is reasonable to assume a Tri-model distribution of the packet length (processing time). In terms of packet count, we have: 64-byte packets occupy 55.9%, 576-byte packets 20% and 1536-byte packets the remaining 24.1%. The 64-byte short packets seldom cause the service pointer move. Only the other two types of packets can make the quantum fall below 0. The negative value is between 0 and -L, with the expected value of -1/2L:

$$\mathrm{E}(q') = -(\frac{20}{44.1}\times 576 + \frac{24.1}{44.1}\times 1536)/2 = -550 = -0.36M \quad \square$$

At scheduling round k, PE_1 to PE_h ($1 \le h \le N$) have finished their scheduling and Ptr points to PE_{h+1}. In the worst case, all the h PEs have the same value as $-M$:

$$q_{1,k}(h+1) = \cdots = q_{h,k}(h+1) = -M \quad (19)$$

In QAS, after each scheduling, the current deficit quantum is decreased and all the other quanta are increased, so for the expected value:

$$E(q_{h+1,k}(h+1)) > E(q_{h+2,k}(h+1)) > \cdots > E(q_{h+n,k}(h+1)) \quad (20)$$

To simplify the derivation, we consider a worse case for the result that these quanta have the same value:

$$q_{h+1,k}(h+1) = \cdots = q_{h+n,k}(h+1) = hM/N - h \quad (21)$$

The deficit quantum of PE_{h+1} is $q_{h+1,k}(h+1) - q'_{h+1,k}$, which will increase the quanta from PE_{h+2} to PE_n, ignoring the high order minority, we get (22):

$$\sum_{1\le j\le n-h} q_{h+j,k} = \sum_{1\le j\le n-h} q_{h+j,k}(h+1) + \frac{(N-h)(N-h-1)}{2(N-1)} q_{h+1,k}(h+1) - (\frac{N-h-1}{N-1})q'_{h+1} - (\frac{N-h-2}{N-1})q'_{h+2} - \cdots - (\frac{1}{N-1})q'_{n-1} \quad (22)$$

From (18), use $\mathrm{E}(q')$ instead of each q' to simplify.

$$\sum_{1\le j\le n-h} q_{h+j,k} = hM + \frac{hM(N-h-1)}{2(N-1)} + \frac{(N-h)(N-h-1)}{2(N-1)} 0.36M \quad (23)$$

From QAS algorithm, the total sending quantity is:

$$\sum_{1\le j\le n-h} sent_{h+j}(k, k+1) = \sum_{1\le j\le n-h} [q_{h+j}(k) - \mathrm{E}(q'_k)] \quad (24)$$

For PE_1 in the next round k+1, from (23, 24) we arrive:

$$\begin{aligned} q_{1,k+1} &= q'_1 + [\sum_{2\le i\le h} sent_i(k, k+1) + \sum_{h+1\le i\le n} sent_i(k, k+1)]/(N-1) \\ &= -M + (h-1)M/(N-1) + \sum_{h+1\le i\le n} sent_i(k, k+1)/(N-1) \\ &= [-(N-h)M + (N-h)0.36M + hM]/(N-1) \\ &\quad + [hM(N-h-1) + (N-h)(N-h-1)0.36M]/2(N-1)^2 \end{aligned} \quad (25)$$

After merging we get (26):

$$[2(N-1)^2/M]q_{1,k+1} = -0.64h^2 + (3.56N - 3.92)h - 0.92(N^2 - N) \quad (26)$$

Here we only care about the sign of $q_{1,k+1}$. Taking N = 32, we get -0.64h^2 + 110h - 912.6 ≥ 0, the solution is: 8.74 ≤ h ≤ 163.1

Under some reasonable assumptions, the maximum number of continuous jump-overs is 8 when the degree of parallelism N is 32. To achieve this, it is required that all the eight conjoint PEs' quanta to be very close to the minimum after scheduling a 1536-byte packet. The probability is $P = (f \times 24.1/44.1)^8$, where f is the confidence interval. If f = 0.2, then P = 2.04e-8[2].

Table 2. Jump-overs in QAS scheduling

Jump Times	Number of Moves	Total Moves
0	733556 (92.1%)	796117
1	55803 (7.0%)	Jump times Per Move 1.09
2	5959 (0.7%)	
3	709 (0.1%)	
4	80	Max Jump Times 5
5	10	
6	0	

[2] It implies the impact from the missing probability can be ignored.

5.Simulations

We build a C++ simulator to simulate the behavior of our NP prototype as shown in Figure 1. By default, there are 32 identical PEs working in parallel inside an NP. The incoming traffic is driven by the real-life trace files, collected at the China Education and Research NETwork (CERNET) backbone (OC-192c) on 11/19/2005. Because the actual load is quite low, it is scaled to the rate we used. When the maximum load is scaled to reach up to 100% of the link bandwidth, the average load is only 45.3% as depicted in Figure 5. Inside the NP, every packet will be buffered at the output to recover its intra-flow packet order before departure. To compare the performance of different algorithms, we measure three major metrics: queuing fairness bound, average packet delay and resequencing buffer size. QAS, the previous algorithms RR, SRR, ERR, P-DBCS as well as the theoretically optimal algorithm SQF are all simulated.

5.1. Queuing Fairness Bound

As in Section II, QFB is defined as the maximal difference among the queuing load of all the PEs, which represents NP's load-balancing performance. In Figure 6, we depict the QFB under the load rate of 45.3%, which demonstrates that the QFB in QAS is much smaller than those in SRR and ERR. In detail, using QAS, the QFB will be constrained below M (M=1500 cycles), while by the other algorithms, the QFB will sometimes reach up to 2M. This validates our theoretical analysis that the QAS achieves much better load-balancing than its precedent algorithms.

5.2. Delay and the Resequencing Buffer Size

Figure 7 depicts the average packet delay under traffic load from 5% to 100%. It is important to note that the reasonable load is below 45.3%, but we still give the much more enlarged cases, in which the maximum load is far beyond the system capacity. It shows that QAS with a look-ahead window size 2 is sufficed to reduce 24% and 18% of the average packet delay from SRR and ERR, respectively. Moreover, its delay performance is very close to SQF. It is worth noting that the QAS achieves such gains without disturbing the $O(1)$ complexity. We also examine the resequencing buffer size for these algorithms. Figure 8 shows that the QAS only needs 60% size of the resequencing buffer compared with SRR and ERR. It equally indicates that QAS will lead to a smaller reordering delay and delay jitter.

5.3. Look-ahead Window Size

We examine the possibility of the pointer *Ptr* jump-over in executing the QAS. It shows that more than two jump-overs rarely happen. In Table 2, we measure the number of pointer jump-overs when look-ahead window size is set to 6. Among the 796117 service pointer moves for 2 million incoming packets, only 7.71% of the packets encounter pointer jump-over, less than 0.1% packets experience more than two jump-overs; and even fewer get more. By setting the look-ahead window size 2, the throughput degradation is limited to less than 0.1%. The performance under look-ahead window size N is shown in Figure 7 and 8. We can see that the marginal gain is quite small.

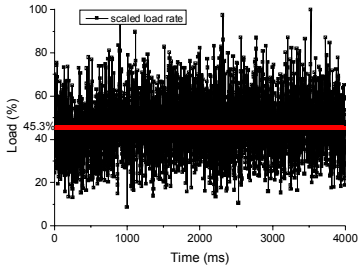

Figure 5. Bandwidth utilization of the enlarged real-trace

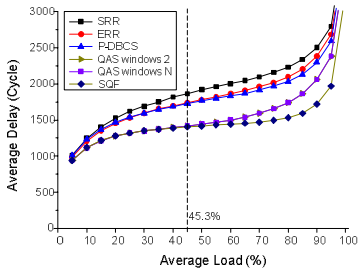

Figure 7. Average delay of different algorithms

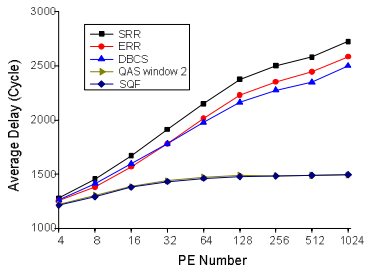

Figure 9. Average delay with varying number of PEs

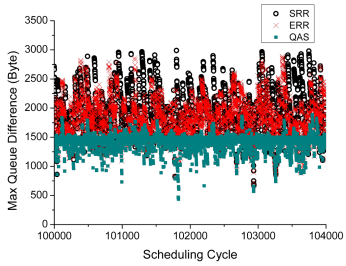

Figure 6. Maximal queuing load difference at run time

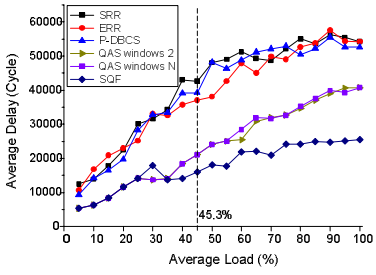

Figure 8. Resequencing buffer size of different algorithms

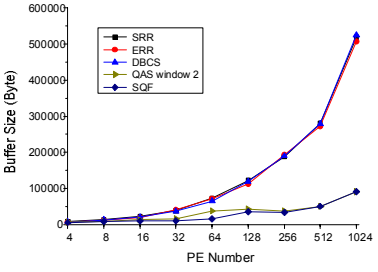

Figure 10. Resequencing buffer size with varying number of PEs

5.4. Impact on the Number of PEs

We also try to study the impact of the parallelism degree on the performance of an NP model. In this simulation slot, the average traffic load is fixed to 45.3%. The average packet delay and the resequencing buffer size of QAS are depicted in Figure 9 and 10, respectively. By QAS, the average delay remains stable when the number of PEs increases, while by SRR or ERR, the performance gets worse beginning from the 8-PE setting. In a 512-PE configuration, a tiny resequencing buffer of 50KB is required for QAS, which is 80% less than that of SRR and ERR.

6.Conclusion

We propose a quantum-adaptive scheduling (QAS) algorithm for both link and processor scheduling. QAS enables both short-term and long-term load-balancing uniformity, as well as the adaptability to non-backlogged traffic input. Compared to previous RR-based approaches, QAS introduces two major advantages. First, QAS updates the service quantum of each PE synchronously, hence achieving better short-term load-balancing uniformity. Second, QAS introduces a PE utilization-based adjustment to improve the performance under non-backlogged traffic. Table 3 summarizes the time complexity, fairness, start-up latency, average delay and buffer requirement of the commonly used scheduling algorithms. It can be seen that QAS performs obviously better.

Real-trace driven simulations show that in a highly parallel NP chip, QAS outperforms all the existing RR-based algorithms: reducing average packet delay by 24% and 18% from SRR and ERR respectively and decreasing resequencing buffer length by 40%.

Acknowledgement

This work is partly supported by NSFC (60573121, 60625201), China 973 program (2007CB310701), the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), Tsinghua Basic Research Foundation (JCpy2005054), 863 high-tech plan (2007AA01Z216).

References

[1] Cavium Networks, *octeon*, http://www.caviumnetworks.com/

[2] Intel Corporation, *IXP2800*, http://developer.intel.com/.

[3] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375-385, Jun. 1996.

[4] H. Adiseshu, G. Parulkar and G. Varghese, "Reliable FIFO Load Balancing over Multiple FIFO Channels," *Washington University Technical Report*, *WUCS-95-11*.

[5] S. Kanhere, H. Sethu and A. Parekh, "Fair and Efficient Packet Scheduling using Elastic Round Robin," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 4, pp. 324-336, Mar. 2002.

[6] T. Wolf and M. Franklin, "Design Tradeoffs for Embedded Network Processors," in *Proc.* of Architecture of Computing Systems, vol. 2299, pp. 149–164, Apr. 2002.

[7] J. Guo, J. Yao and L. Bhuyan, "An Efficient Packet Scheduling Algorithm in Network Processors," in *Proc. IEEE INFOCOM* 2005.

[8] L. Kencl and J.-Y. Le Boudec, "Adaptive Load Sharing for Network Processors," in *Proc. IEEE INFOCOM* 2002.

[9] W. Shi, M. H. MacGregor and P. Gburzynski, "Load Balancing for Parallel Forwarding," *IEEE/ACM Transactions on Networking*, vol.13 no.4, pp. 790-801, Aug. 2005.

[10] L. Shi, Y. Zhang, J. Yu, B. Xu, B. Liu and J. Li, "On the Extreme Parallelism Inside Next-Generation Network Processors," in *Proc. IEEE INFOCOM* 2007.

[11] S. J. Golestani, "A Self-clocked Fair Queueing Scheme for Broadband Application," in *Proc. IEEE INFOCOM* 1994.

[12] J. C. R. Bennett and H. Zhang, "WF2Q: worst-case fair weighted fair queuing," in *Proc. IEEE INFOCOM* 1996.

[13] Fabio M. Chiussi and Andrea Francini, "Minimum-delay self-clocked fair queuing algorithm for packetswitched networks," in *Proc. IEEE INFOCOM* 1998.

[14] B. Wu, Y. Xu, H. Lu and B. Liu, "An Efficient Scheduling Mechanism with Flow-Based Packet Reordering in a High-Speed Network Processor," in *Proc. IEEE HPSR* 2005.

Table 3. Comparison of different algorithms under 32-PE configuration

Scheduling Algorithms	Time Complexity	QFB	Start-up Latency	Average Delay 45.3% Load (cycle)	Resequencing Buffer 45.3% Load (byte)
DRR / SRR	$O(1)$	$M+2m$	$(M+m-1)N/r+m/r$	1862	42635
ERR	$O(1)$	$3m$	$(2m-1)N/r+m/r$	1739	37019
P-DBCS	$O(1)$	$M+2m$	$(M+m/2-1)N/r+m/r$	1727	39242
QAS window 2	$\boldsymbol{O(1)}$	$\boldsymbol{2mN/(N-1)}$	$\boldsymbol{(mN+2m)/r}$	**1418**	**21074**
SQF	$O(\log N)$	$2m-1$	N/A	1406	16061