Scalable Scheduling Architectures for High-Performance Crossbar-Based Switches

Jing Liu, Mounir Hamdi, Qingsheng Hu and C. Y. Tsui
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
hamdi@cs.ust.hk

Abstract—**This paper presents a novel scalable scheduling architecture for high-performance crossbar-based switches with virtual output queuing (VOQ) scheme. In contrast to traditional switching architectures where the scheduler is implemented by one single centralized scheduling device, the proposed scheduling architecture connects several small scheduling devices in series and the arbitration algorithm is executed in parallel. Thereby the inputs of each single scheduling device establish connections to a group of outputs, by considering both their local transmission requests as well as global outputs availability information. The advantage of this architecture lies in its ability to implement large schedulers (> 64) with several small scheduling devices as well as in its capability to achieve high-performance scheduling. We first introduce a distributed parallel round robin scheduling algorithm (DPRR) for the proposed architecture. Through the analysis of simulation results on various admissible traffics, it is shown that the performance of DPRR is much better than the performance of other round robin scheduling algorithms commonly used on centralized schedulers. We also prove that under Bernoulli i.i.d. uniform traffic, DPRR achieves 100% throughput. Moreover, we introduce a distributed parallel round robin scheduling algorithm with memory (DPRRM) as an improved version of DPRR to make it stable under any admissible traffic.**

I. INTRODUCTION

Numerous studies show that the Internet traffic is growing at rates of 100% to 150% every year. On the other hand, the switches/routers' capacity has doubled every 18 months. If these growth trends (Internet traffic vs. switches/routers capacity) are to continue unaltered for few years, this will result in a big disparity between the capacity of individual switches/routers and the Internet traffic volume. As a result, there is a growing need for closing this gap through the deployment of scalable packet switches/routers.

There are three major switching architectures used: input queued (IQ), combined input and output queued (CIOQ) and output queued (OQ) switches. As port densities and line rates increase, the IQ switch architectures are considered as a pragmatic approach for implementing scalable switches and routers. In IQ switches, virtual output queuing (VOQ) is typically used to eliminate the Head of Line (HoL) blocking [1]. With VOQ [2], each input link maintains a separate queue for each output link, which stores incoming cells destined to the corresponding output link. Then a *scheduler* determines how and when packets should be buffered, scheduled, and switched.

There have been various scheduling algorithms proposed for IQ switches in recent years. Maximum weight matching (MWM) algorithms have very good performance under any admissible traffic, for example, LQF, OCF and LPF [3][4]. However, they are too complex to be practical for high-performance switches. Maximal size matching (MSM) algorithms are practical to be implemented in hardware, such as iSLIP [5] and FIRM [6]. They achive a good performance under uniform traffic; however, their performance is degraded at high loads when the traffic is bursty or non-uniform. In contrast to deterministic algorithms, the recently proposed randomized algorithms achieve a good performance in terms of stability and require only linear computational complexity. However their delay is high compared with MSM algorithms. This is true even for non-uniform traffic as long as the maximal size matching algorithms are operating in their "stable" region [10]. The main reason for this is that the randomized algorithms have been designed with the objective of making them stable, rather than minimizing the average delay.

The implicit assumption made by most researchers is that these scheduling algorithms can fit within a single VLSI chip. That might be feasible when the switch is of small size (< 64x64). On the other hand, for larger switch sizes, that cannot be done. Because of I/O-space-power limitations of silicon devices, it is impossible to fit schedulers as simple as iSLIP on a single chip when the switch size becomes large. Surprisingly, there is no mention of this fact in the literature even though it has a big effect on the architecture as well as the performance of the scheduler.

To solve this problem, we present a scalable scheduling architecture that can implement a large scheduler using smaller scheduling devices. We also propose an appropriate scheduling algorithm called distributed parallel round robin scheduling (DPRR) and an improved version of DPRR, which is called distributed parallel round robin scheduling with memory (DPRRM) for the new architecture.

This paper is organized as following: Section 2 introduces background knowledge on various scheduling algorithms. In Section 3, a sequential scheduling scheme is presented. Section 4 discusses two pipeline scheduling schemes, one is the simple pipeline scheduling, and the other is the distributed parallel scheduling. In Section 5, the scheduling algorithm DPRR and DPRRM are introduced. Section 6 presents the simulation results and the performance analysis for our

0-7803-8375-3/04/$20.00 © 2004 IEEE

scalable scheduling architecture and scheduling algorithms. Finally, Section 7 concludes the paper.

II. BACKGROUND

A. *Round robin scheduling algorithms*

All existing round-robin scheduling algorithms are run with several iterations, and each iteration consists of three steps: request, grant and accept:

1. Each input sends requests to the outputs;
2. Each output selects one request independently and sends a Grant to the corresponding input;
3. Each input selects one Grant to accept.

Round robin matching (RRM) [5] is the basic round robin scheduling algorithm. RRM sends grant in the second step and accept in the third step by round-robin policy. Each output sends a grant to the request that appears next from the pointer in a fixed round-robin order. Each input works in the same way in selecting an output to issue accept. After selection, the output pointer is incremented (modulo N) to one location beyond the granted input. If no request is received, the pointer stays unchanged. Input pointers execute the same operation.

RRM does not perform very well, since it suffers from 'pointer synchronization' problem, which means the grant pointers of some outputs tend to point to the same input.

iSLIP [5] achieves improvement in solving the pointer synchronization problem by altering in the management of pointers: at each output, the pointer remains unchanged unless the corresponding grant is accepted. The minor change leads to a significant increased throughput.

Fcfs in round robin matching (FIRM) [6] achieves more fairness than iSLIP by moving the grant pointers to the granted input instead of staying unchanged when the grant is not accepted.

Based on the observation that the desynchronization of pointers lead to good performance, recently, a group of new round robin scheduling algorithms—static round robin (SRR) [7] were proposed. The basic idea is to keep pointers fully desynchronized. There are different variations of SRR Among them, rotating double SRR (RDSRR) performs the best. In the following we provide the specification of RDSRR:

Initialization. The output pointers are set to an initial pattern such that there is no duplication among the pointers. The same goes for the input pointers.

The three steps of one iteration are:

Step 1: Request. Each input sends a request to every output for which it has a queued cell.

Step 2: Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. To achieve fairness, clockwise and counter-clockwise rotations are used alternately, each for one time slot. The output notifies each input whether or not its request was granted. The pointer pointing to the highest priority element of the round robin schedule is always incremented by one (modulo N) whether there is a grant or not.

Step 3: Accept. If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer pointing to the highest priority element of the round-robin schedule is always incremented by one (modulo N) whether there is a grant or not.

The achievements with RDSRR are: (a) Lower delay (b) With clockwise and counter-clockwise rotation scheme, each input has a chance to be served. (c) Easy to implement in hardware.

B. *Randomized algorithms*

While deterministic scheduling algorithms are complex to implement, randomized algorithms are quite simple and their performance is surprisingly good.

A basic randomized algorithm is proposed by Tassiulas [8].

Algo1:

(a) Let $S(t)$ be the schedule used at time t.

(b) At time $t + 1$ choose a matching $R(t + 1)$ uniformly at random from the set of all $N!$ possible matchings.

(c) Let $S(t + 1) = \arg \max_{S \in \{S(t), R(t+1)\}} <S, Q(t+1)> (Q(t+1)$

is the queue-lengths matrix at time $t + 1$.)

Lemma 1 (Tassiulas[8]) Algo1 *is stable under any Bernoulli i.i.d. admissible input.*

$R(t + 1)$ is called a probe-matching. It is proved that if the probe-matching can hit the MWM with a finite probability, then the scheduling algorithm is stable under any admissible traffic [10].

If arrival information is exploited in obtaining $R(t + 1)$, the performance of the algorithm can be improved significantly. In [10], an arriving matching is generated to be the probe-matching. First, an arrival graph $A(t + 1)$ is constructed in the way that in case there is an arrival from an input to an output, an edge is added between the two. The arrival graph is not necessarily a matching, since multiple edges can be incident on the same output node due to multiple arrivals to that output. When $A(t + 1)$ is not a matching, for all the output nodes which have more than one edge incident on, choose one edge at random or with highest weight and discard the remaining edges. At the end of this process, each output is matched with exactly one input and the probe-matching with arrival information $R(t + 1)$ is obtained. The arrival process is stationary and Bernoulli i.i.d.. Hence, there is a finite probability that $R(t + 1)$ is the MWM. Thus, the scheduling algorithm which exploits arrival information in the probe-matching is stable. We call this algorithm **Algo 2.**

Lemma 2 (Paolo[10]) Algo2 *is stable under any Bernoulli i.i.d. admissible input.*

III. SCHEDULING ARCHITECTURE and SEQUENTIAL SCHEDULING

In this section, a new scheduling architecture is proposed. We consider a switch fabric with a large number of ports, for example a 256x256 switch. Figure 1 shows the block diagram of designing a 256x256 scheduler using eight 32x256 schedulers. The eight 32x256 schedulers are connected one by one and the scheduling algorithms are run sequentially on them to comfigure a 256x256 switch. Each 32x256 scheduler

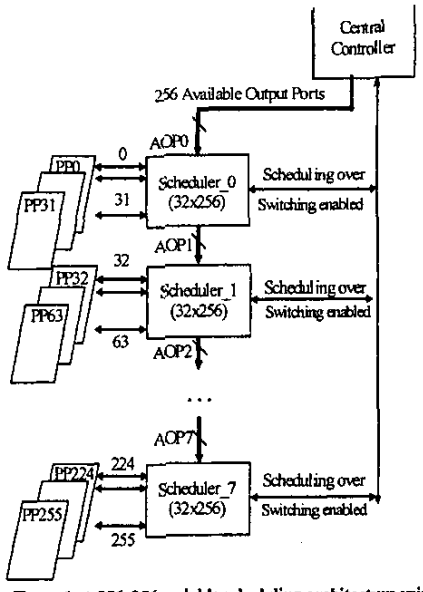

Figure 1. A 256x256 scalable scheduling architecture using 8 32x256 scheduling devices.

has 32 inputs connected to *port processors* (PPs) and a total of 256 inputs of the 8 schedulers are linked to PPs. The *Central Controller* is used to control the whole system.

A 256-bit control signal, which is called *available output port* (AOP) is used in the proposed architecture. All output ports have read and write access to the AOP, indicating the reservation status of each of the 256 outputs. We denote a logical '0' as an available output port while '1' is a previously reserved one. In the beginning, the Central Controller selects a default single scheduler as a start node of the sequential scheduling. Suppose Scheduler_0 is the selected one to be the start node. It will receive the signal AOP0 from the Central Controller. Since no output has been reserved yet, the 256 bits in AOP0 are all logic '0's. An appropriate scheduling algorithm will be run on Scheduler_0 and consequently, part of the 256 outputs will have connections with the inputs of Scheduler_0.

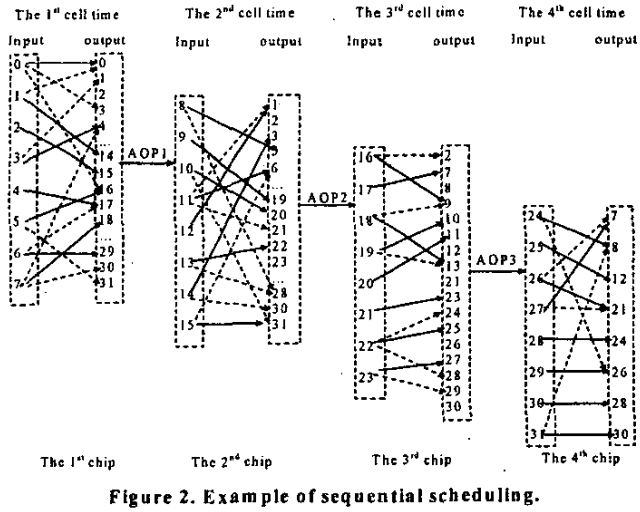

Figure 2. Example of sequential scheduling.

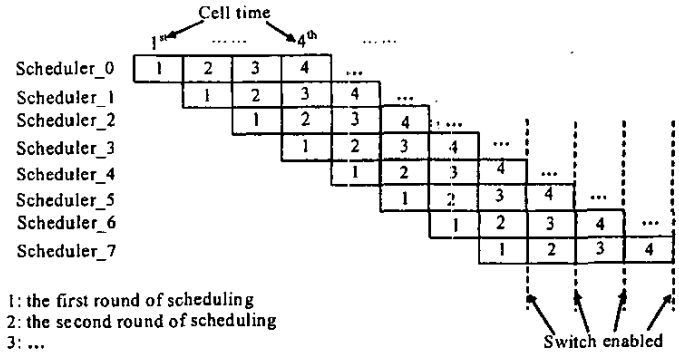

Figure 3. A simple pipeline scheduling.

The AOP will be updated according to the reservation status of outputs and Scheduler_1 will receive the updated AOP1. Meanwhile Scheduler_0 sends the signal of "Scheduling over" to Central Controller to notify the completion of its scheduling.

Thereafter, Scheduler_1 selects outputs from AOP1 by running the scheduling algorithm and updates AOP1 to AOP2 according to the reservation status. AOP2 is passed to Scheduler_2 to continue the scheduling and a "Scheduling over" signal is sent to the Central Controller again. All the separate schedulers will continue this scheduling process sequentially. When the Central Controller receives the signal of "Scheduling over" from the last scheduler, Scheduler_7, it will send a signal of "Switch enabled" back to all the single scheduling devices to indicate the end of this round's scheduling and enable the transmission of cells from the PPs to the crossbar.

Figure 2 gives a more detailed example. In this example we use four 8x32 scheduling devices to arbitrate a 32x32 switch.

At the first cell time, all the 8 inputs of the first device can send their requests to all the 32 outputs and the appropriate scheduling algorithm is run to select outputs. Further suppose, output ports {0, 4, 14, 15, 16, 17, 18 and 29} are matched with input 0 to input 7 respectively. Thus AOP1 is:

$AOP1$ = {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31}

At the second cell time, input 8 to input 15 send their requests to those outputs in AOP1. After the scheduling of the second device, another group of output ports will be reserved and must be removed from AOP1, so the remaining AOP is:

$AOP2$ = {2, 7, 8, 9, 10, 11, 12, 13, 21, 23, 24, 25, 26, 27, 28, 30}

Similarly, $AOP3$ = {2, 9, 10, 11, 13, 23, 25, 27}

When all the 4 scheduling devices finish the whole scheduling, 32 inputs and outputs are matched completely.

IV. PARALLEL PIPELINE SCHEDULING SCHEME

A. A simple pipeline scheduling

The sequential scheduling scheme reserves outputs for each separate scheduler in turn, which requires several time slots to carry out the scheduling and to reconfigure the crosspoints. A

pipeline scheme can be employed to reduce the delay dramatically.

Figure 3 shows the simple pipeline scheduling scheme. Each single scheduler can start the next round of scheduling immediately as long as it completes its current scheduling and receives the AOP signal from the last scheduler. One *round of scheduling* consists of several steps of scheduling performed by each of the single scheduling devices in turn. In Figure 3, the number of 1, 2, 3, 4, ..., stands for the first round of scheduling , the second round of scheduling, *etc.* When Scheduler_0 completes the 1st round of scheduling, it sends AOP to Scheduler_1, then starts the 2nd round of scheduling. As soon as Scheduler_1 receives the signal of AOP from Scheduler_0, it starts the 1st round of scheduling. When Scheduler_1 completes the 1st round of scheduling based on AOP, it can start the 2nd round of scheduling right after it receives the AOP signal of 2nd round scheduling from Scheduler_0.

After the whole round of scheduling is completed, the signal "Switch enable" will be issued from the Central Controller to each PP indicating that this round of scheduling is over and the switch is reconfigured. Each port transmits cells to its designated output in accordance with the selected configuration.

This pipeline scheduling scheme makes the scalable scheduling more efficient. One round of scheduling can be completed at each cell time. The delay is significantly reduced compared with sequential scheduling.

B. Distributed parallel scheduling

Obviously, this simple pipeline scheduling scheme cannot guarantee the fairness of scheduling for each input. If the highest priority is given to some fixed device (for example, Scheduler_0 is the start node with the highest priority in Figure 3), this simple pipeline pattern may result in an unbalancing selection when the load is non-uniform.

To improve the performance of the basic scalable scheduling architecture, we develop a fair scalable scheduling architecture— distributed parallel scheduling architecture. In this architecture, all single schedulers are connected in a round

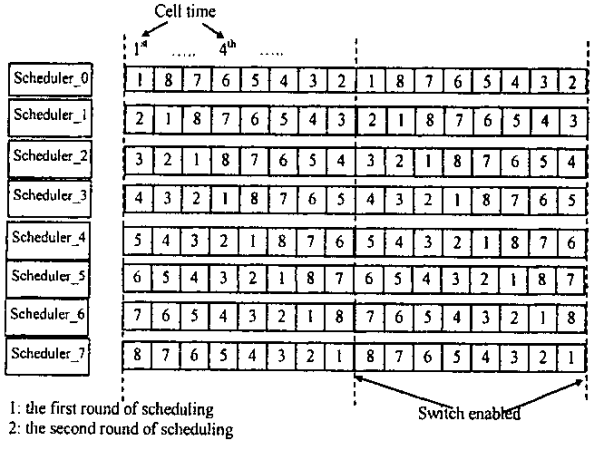

Figure 4. Distributed parallel scheduling.

way, *i.e.* the 1st device is connected to the 2nd device, the 2nd device connected to the 3rd device, etc. The last device connected to the 1st device. Based on this circular connection, a fair pipeline scheduling scheme is proposed. Figure 4 shows the distributed parallel scheduling scheme. All single scheduling devices work in parallel. However, they work in different rounds of scheduling. For example, at the first cell time, Scheduler_0, Scheduler_1, ... and Scheduler_7 starts their 1st, 2nd, 3rd, ..., 8th round of scheduling respectively (see Figure 4). Then, at the second cell time, Scheduler_0 receives the signal AOP8 from Scheduler_7 and continues the scheduling of 8th round. Scheduler_1 receives signal AOP1 from Scheduer_0 and continues the work of the 1st round scheduling, ..., in the same way, Scheduler_7 receives its AOP information from Scheduler_6 and starts the 7th round of scheduling.

At the following cell times, every single scheduler works in a similar way. After eight cell times, each single scheduling device has already completed eight rounds of scheduling, so in each port processor there are up to 8 cells that have been selected for transmission. All the configurations of different scheduling rounds are stored in memory and as soon as the port processors receive the message of "Switch Enable" from the Central Controller, these scheduled cells will be transmitted to the crossbar one configuration after another. For example, all cells belonging to the 1st round of scheduling from all PPs are transmitted first. Then the cells belonging to the 2nd round are transmitted until the cells belonging to the 8th round of scheduling are transmitted at the last cell time.

We can see that each single scheduler of the distributed parallel scheduling scheme has an equal chance of being the start node with the highest priority in reserving outputs. Thus, it is fair way of scheduling.

V. SCHEDULING ALGORITHMS

A. DPRR scheduling algorithm

1. Algorithm specification

After analyzing the scheduling architecture, we now introduce scheduling algorithms run on each of the separate schedulers. An appropriate scheduling algorithm is a key to the performance. Since weight-based algorithms are too complex to be practical, our proposed scheduling algorithm is based on round robin scheme. One scheduling device is an *MxN* switching device, whereby *M<N*. Since the number of inputs is less than the number of outputs, pointers of the output arbiters are quite easy to synchronize if we use typical iterative scheduling algorithms, such as iSLIP or FIRM. For example, within FIRM, if the grant for the request is not accepted by the input, the pointer of the output arbiter will stay at the granted one. Since *M* is less than *N*, in case the traffic load is high, one input will receive more than one grant from different outputs with a large probability and the input can only accept one output; all the others will be synchronized. In the simulation, we have seen that with iSLIP and FIRM, the distributed architecture does not perform well.

Let us consider an 8x32 switching device. In our proposed scheduling algorithm, we force the pointers of output arbiters to keep balance in synchronization. Suppose a 4x16

Table 1. Pointer configuration of outputs for Scheduler_0 of 4x16 switching device.

	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
Pointers in time slot 1	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Pointers in time slot 5	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1

scheduling device, on average, every 4 pointers of the output arbiters point to the same input and the pointers of the input arbiters are initially set to some patterns without any duplication. Every 4 (this equals the number of cell times for one round scheduling) time slots, both the input pointers and output pointers increase by one (module 16). A possible configuration for one of the 4x16 switching device, say Scheduler_0 for inputs from 0 to 3 is shown in Table 1. We also require that all the pointers of outputs that point to the same inputs of Scheduler_0 also point to the same inputs of Scheduler_1, 2 and 3 as well. For example in Table 1, at time slot 1, output 0, output 4, output 8 and output 12 point to the same input: input 3 of Scheduler_0, so those outputs are required to point to the same input of Scheduler_1, 2 and 3. The reason of this requirement is to keep balance in pointer synchronization. We call our proposed scheduling algorithm distributed parallel round robin (DPRR). It is based on the idea of RDSRR, which is proved to perform well under most of the traffic patterns. The three steps in one iteration of DPRR are as following:

Step 1: Request. Each input sends a request to every output for which it has a queued cell.

Step 2: Grant. If an output receives any request, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The search is in clockwise and counter-clockwise rotation alternately, each for one time slot.

Step 3: Accept. If an input receives grants, it accepts the one that appears next in a fixed round-robin schedule staring from the highest priority element.

Now we explain the design of DPRR. Both uniform traffic and non-uniform traffic are considered.

The pointer setting and moving scheme favors Bernoulli i.i.d. uniform traffic, since the pointers are kept balance in synchronization and the traffic is also distributed uniformly among all outputs. It is required that the pointers stay unchanged for a continuous 4 time slots. The reason for that is that one round of scheduling requires 4 time slots to complete; thus 4 time slots is a period time to update pointers. This feature desires bursty traffic, since cells arrive within the same burst might be served continuously so that the suffering of delay from burstiness will be reduced.

As for the non-uniform traffics, two typical traffic patterns are hotspot and diagonal. The hotspot traffic pattern assumes one output to be the "hotspot". The traffic load from all the inputs to this "hotspot" is much higher than to other outputs. For example, for a 4x4 switch, the traffic matrix of hotspot traffic is as following:

$$\begin{bmatrix} 2x & x & x & x \\ 2x & x & x & x \\ 2x & x & x & x \\ 2x & x & x & x \end{bmatrix}$$

Output 0 is the hot-spot with higher rate of traffic destined to it, and all other traffic is distributed to other outputs uniformly.

In our proposed scheduling architecture, since each of the scheduling devices runs the scheduling algorithm and reserves outputs in turn, if the "hotspot" fails to be reserved with Scheduler_0, it will be continued to be scheduled with Scheduler_1, and so on. The "hotspot" will have large possibility to be served in one round of scheduling. Thus the scalable scheduling architecture with reservation of outputs decreases the waste of bandwidth.

Another typical non-uniform traffic pattern is diagonal traffic. The traffic matrix is shown as following for a 4x4 switch:

$$\begin{bmatrix} x & 1-x & 0 & 0 \\ 0 & x & 1-x & 0 \\ 0 & 0 & x & 1-x \\ 1-x & 0 & 0 & x \end{bmatrix}$$

The traffic is concentrated on two diagonals. One is heavier than the other. (x = 2/3)

Consider output 0: the traffic only comes from input 0 and input 3. The one-direction searching scheme of iSLIP and FIRM will favor one input over the other. For example, when the pointer is located at 1, 2, and 3, request from input 3 will be granted. The only chance for the request from input 0 to be granted is when the pointer moves to 0 or in case there is no request from input 3. That causes unfairness. The two-direction searching scheme that is conducted alternately for both directions increases the fairness of scheduling.

2. *Desynchronization effect of DPRR*

Theorem 1. DPRR achieves 100% throughput under admissible Bernoulli i.i.d. uniform traffic.

Proof. We assume that the offered load is 100% uniform traffic, so that every VOQ is always occupied with cells. Let us consider a 16x16 switch, which is composed of four 4x16 scheduling devices. Thus, the pointers of the output arbiters are desynchronized in such a way, that 4 outputs point to one input of each 4x16 scheduling device and the scheduling are processed in turn with each scheduling device. Assume Scheduler_0 is the start node. Then each of the 4 inputs will choose one output from four candidates and send accept. When the scheduling moves to Scheduler_1, the AOP has 12 outputs and each input of Scheduler_1 has 3 outputs pointing to it (recall that all the pointers of outputs that point to the same inputs of Scheduler_0 also point to the same inputs of Scheduler_1, 2 and 3. Thus the removed outputs from AOP point to different inputs of Scheduler_1). Thus each input of Scheduler_1 chooses one output from 3 candidates and send

"accept". So does Scheduler_2, each input of which chooses one output from 2 candidates and send "accept". The inputs of Scheduler_3 will send "accept" to the rest of the outputs. Consequently each input will send cells to each output. Since every 4 time slots, the pointers of all outputs will increase by one (modulo 16), each input will keep sending cells to each output indefinitely. The utilization of each output link is 100%. Thus, DPRR achieves 100% throughput under uniform traffic.

B. DPRRM scheduling algorithm

1. Algorithm specification

Since the arrival process is stationary and Bernoulli i.i.d., and also queue accumulation is due to arrivals, thus to exploit the arrival property is a good way to improve performance and ensure stability. We modify DPRR to make it stable and as a result the improved algorithm, distributed parallel round robin scheduling with memory (DPRRM) is proposed. The the specification of DPRRM is as follows.

Let S_{t-1} be the schedule used at the previous time slot and A_t is the matching obtained from arrival. To obtain A_t, first we construct the arrival graph G_t. If there is an arrival from input i to output j, we add an edge from input i to output j. If G_t is a matching, then $A_t = G_t$. If G_t is not a matching, which means there are more than one arrival to one output, we choose the heaviest edge and remove the others for this output to obtain a matching A_t from G_t.

Let $S_t = \arg \max_{S \in \{S_{t-1}, A_t, D_t\}} \langle S, Q_t \rangle$, where D_t is the matching obtained from DPRR and Q_t is the queue-lengths matrix. In the first several time slots, there is no matching obtained from DPRR, we set all the elements of D_t to be 0's.

2. Stability of the algorithm

Theorem 2. DPRRM is stable under any admissible Bernoulli i.i.d. traffic.

Proof. In DPRRM, we use the matching A_t, which is derived from the arrival graph, as one of the probing matchings. The arrival process is stationary and Bernoulli i.i.d.. Hence, there is a finite probability $\delta > 0$ such that A_t is the MWM. According to **Lemma 2**, this is sufficient to prove the stability of DPRRM.

VI. ANALYSIS of SIMULATION RESULTS

In our simulation, we consider a 32x32 switch. In our proposed parallel scheduling architecture, we use four scheduling devices, each of which is a 8x32 switching device connected with each other. The traffic is Bernoulli i.i.d. and admissible (no input or output is overloaded). Uniform traffic, uniform bursty traffic (with bursty length of 10 cells) and various non-uniform traffic patterns, namely the diagonal and hotspot cases are considered. The algorithms are executed using one iteration.

Figure 5 shows the average delay performance of various algorithms under uniform traffic. From this figure, we can see that when the load is low, the delay of the proposed scheduling architecture is nearly a constant value. That is because when the scheduling starts, all the cells cannot be transmitted until one round of scheduling is completed. That results in an initial delay, which depends on the number of scheduling devices used in the architecture. Practically, it is close to $r-1$, where r is the number of scheduling devices used. For DPRRM, since there are cells sent from the arrival matching in the beginning, the delay is slightly lower than DPRR when the load is low. When the load is above 0.6, our proposed architecture with DPRR performs much better than all the other algorithms. As we mentioned above, the pointer moving schemes of iSLIP and FIRM are not suitable for our proposed scheduling architecture. With DPRR, the pointers synchronize in a balancing way and the reservation of outputs in turn by inputs reduces the waste of bandwidth. DPRRM has comparable performance with DPRR.

Figure 6 shows the delay performance under uniform bursty traffic. The result is similar to the case under uniform traffic.

Figure 7 shows the simulation results under hotspot traffic. The delay of our proposed architecture is close to a constant in all ranges. Only in the highest load range, it increases slightly. However it is still much lower than iSLIP, FIRM and RDSRR, run on a single switch. The delay of iSLIP and FIRM run on the proposed architecture is similar to that of DPRR and DPRRM run on the proposed architecture, since our proposed architecture with a large probability makes the"hotspot" be served all the time, the architecture favors hotspot traffic.

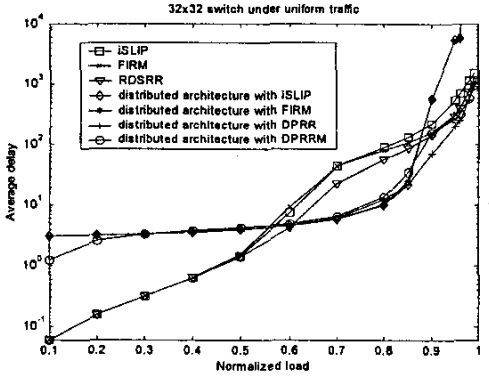

Figure 5. Average delay under uniform traffic.

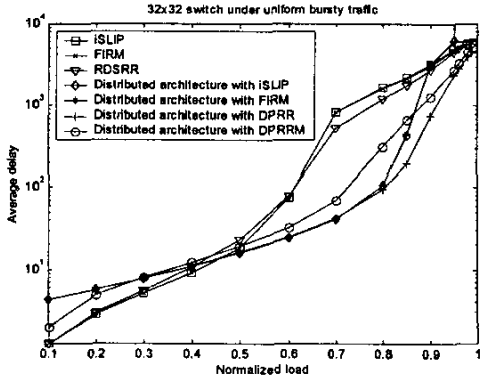

Figure 6. Average delay under uniform bursty traffic.

Figure 8 shows the delay performance under diagonal traffic. When the traffic is high, the delay performance of the *proposed* architecture with DPRR is close to those algorithms run on a single switch, even lower.

Let us consider one scheduling device for a 16x16 switch. Suppose it is Scheduler_0 for inputs 0 to 3. Figure 9 shows the requests sent by input 0 to 3 of Scheduler_0. From the figure, we can see that output 0 will grant input 0 all the time, since there is no request from input 15 will be sent to output 0 when the scheduling is processed on Scheduler_0. Hence, even the request from input 0 to output 1 is granted, it will always compete for acceptance with grant from output 0 to input 0, which makes the cells in VOQ(0,1) accumulate heavily. So do cells in VOQ(8,9), VOQ(16,17) and VOQ(24,25). That influences the delay performance. However, DPRRM shows much better performance. The delay of DPRRM increases steadily when the load becomes high. From the figure, we can also see that DPRRM shows a much better stability performance than all the other algorithms do.

VII. CONCLUSION

A scalable scheduling architecture is crucial for building high-capacity switches. In this paper, we present a fair scalable scheduling architecture, which employs a distributed parallel pipeline scheduling scheme for input queued switches. Using this scalable scheduling architecture, a large scheduler can be implemented using several smaller single scheduling

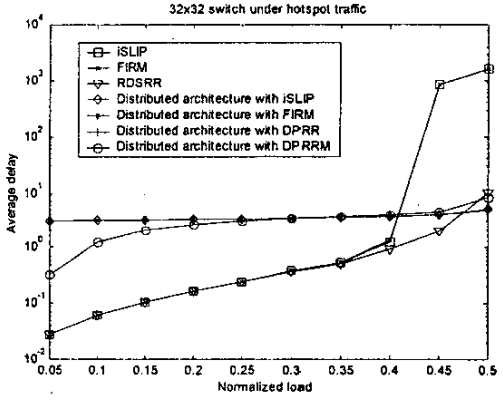

Figure 7. Average delay under hotspot traffic.

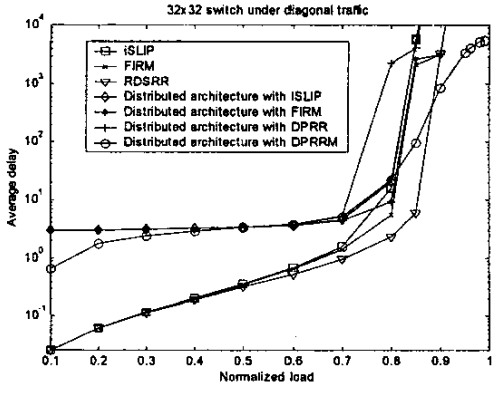

Figure 8. Average delay under diagonal traffic.

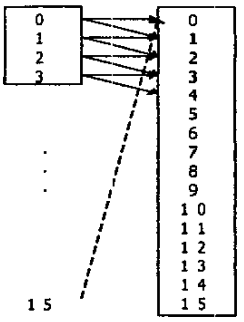

Figure 9. Requests of Scheduler_0 under diagonal traffic.

devices. We also propose a round robin scheduling algorithm named DPRR and an improved version DPRRM for our proposed scheduling architecture. Our architecture employ the reservation of outputs in turn scheme, which increases the instant throughput and decreases the waste of bandwidth. The pointer setting and moving scheme of DPRR reduces pointer synchronization and thus reduces cell delay. The simulation shows that our proposed architecture with DPRR has very good performance when the traffic load is high under most of the traffic patterns and the delay at low load is close to a constant value ($= r-1$) due to initial delay. DPRRM achieves a good performance as well as DPRR while it is stable under any admissible traffic. Thus under non-uniform traffic, DPRRM shows a high performance.

REFERENCES

[1] M. Karol, M. Hluchyj, and S. Morgan, "Input versus Output Queuing on a Space Division Switch," *IEEE Trans. Communications*, 35(12) (1987) pp.1347-1356.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Comput. Syst.*, pp. 319-52, Nov. 1993.

[3] A. Mekkittikul and N. McKeown, "A Starvation-free Algorithm for Achieving 100% Throughput in an Input-Queued Switch," *ICCCN '96*, Oct. 1996, pp.226-231.

[4] A. Mekkittikul and N. McKeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," *IEEE INFOCOM 98*, San Francisco, April, 1998, vol. 2, pp.792-799.

[5] N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches," *IEEE Transactions on Networking*, April 1999, Vol 7, No.2.

[6] D. N. Serpanos and P. I. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues," *IEEE INFOCOM*, 2000.

[7] Y. Jiang and M. Hamdi, "A Fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture," *High Performance Switching and Routing*, 2001 *IEEE Workshop on*, pp. 407-411.

[8] L. Tassiulas, "Linear Complexity Algorithms for Maximum Throughput in Radio Networks and Input Queued Switches," *IEEE INFOCOM'98*, New York, 1998, vol. 2, pp.533-539.

[9] P. Giaccone, B. Prabhakar, and D. Shah, "Towards Simple, High-performance Schedulers for High-aggregate Bandwidth Switches," *IEEE INFOCOM*, 2002.

[10] P. Giaccone, "Queueing and Scheduling Algorithms for High Performance Routers," *PhD Thesis (149 pages)*, Politecnico di Torino, Italy, February 2002.

[11] N. Mckeown, "Scheduling Algorithms for Input Queued Packet Switches," *PhD Thesis*, University of California at Berkeley, May 1995.