Distributed Parallel Scheduling Algorithms for High-Speed Virtual Output Queuing Switches

Lotfi Mhamdi[1] Mounir Hamdi[2]
[1] Computer Engineering Dept., Delft University of Technology, The Netherlands
[2] Computer Science and Engineering Dept. HKUST, Hong Kong
lotfi@ce.et.tudelft.nl, hamdi@cse.ust.hk

***Abstract*—This paper presents a novel scalable switching architecture for input queued switches with its proper arbitration algorithms. In contrast to traditional switching architectures where the scheduler is implemented by one single centralized scheduling device, the proposed architecture connects several single scheduling devices in series and a distributed scheduling algorithm is run sequentially on them, whereby the inputs of each single scheduling device build connections to a group of outputs, considering both their local transmission requests as well as global outputs availability information. We show that a pipeline pattern can be used to increase the efficiency of the scheduling scheme with scheduling algorithms running in parallel on all the separate scheduling devices. 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, or very close to the performance of, other round robin scheduling algorithms. We also prove that under Bernoulli i.i.d. uniform traffic DPRR achieves 100% throughput. Secondly, 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

There are three major switching architectures in use today: input queued (IQ), combined input and output queued (CIOQ) and output queued (OQ) switches. As port densities and line rates increase, IQ switch architectures are acknowledged as a pragmatic approach for implementing scalable switches and routers. In IQ switches, virtual output queueing (VOQ) is used to eliminate the head of line (HoL) blocking [1]. With VOQ [2], a scheduler is needed to effectively resolve the input and output contention and to decide the departure order of cells from the input cards to the corresponding outputs.

Various scheduling algorithms have been 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 and can be implemented in hardware, such as iSLIP [5] and FIRM [6]. They have fairly good performance under uniform traffic; however, their performance is degraded at high loads when the traffic is bursty or non-uniform.

In contrast to the deterministic algorithms, a class of randomized algorithms has been proposed. Randomized algorithms achieve good performance in stability while keeping linear 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 [7]. The main reason for this is that the randomized algorithms have been designed with the objective to make them stable, rather than achieving a small average delay. Another problem with randomized algorithms is that they remain complex in hardware implementation. For example the "MERGE" procedure in LAURA and SERENA [8] may induce a large delay which degrades the performance of the algorithms. Consequently, these algorithms have been proposed primarily for switches with a small number of ports and are generally unsuitable for next-generation switches, where hundreds and even thousands of ports have to be considered.

There are two main challenges in cost-effectively designing very large capacity switches: port count and port speed. Unfortunately, these two challenges are interrelated and go hand in hand. With ever growing bandwidth demands, the question is how to build a very large capacity switch, with both large port count and high speed. Recent research work attempted to tackle the port speed scalability problem by trying to relax the arbitration time constraint [9][10][11]. To solve the very strict scheduling time constraint, some pipeline-based arbitration algorithms have been proposed. A scheme named Round-Robin Greedy Scheduling (RRGS) has been proposed in [9]. The RRGS algorithm is based on pipeline technique so that the arbiters perform their arbitrations for the future time slots. An improved version of RRGS, that uses weight, was introduced in [10] and was called Weighted-RRGS (WRRGS). It was pointed out in [11] that both RRGS and WRRGS fail to guarantee fairness among inputs and this is undesirable. An arbitration algorithm called Pipelined Parallel Maximal-Sized Matching Scheduling (PMM) was introduced in [11] and tries to overcome the failure of RRGS

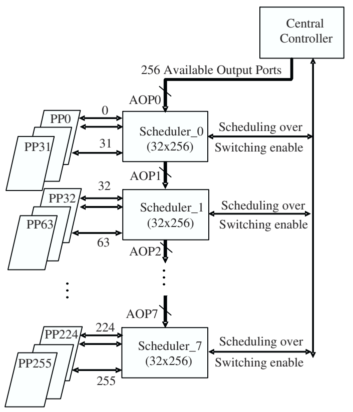

Fig. 1. A 256 × 256 scalable scheduling architecture using 8 32 × 256 scheduling devices.

and WRRGS. PMM operates in a pipelined manner and relaxes the timing constraint by allowing one scheduling phase to take more than one time slot. While PMM overcomes the failure of RRGS, both PMM and RRGS didn't, radically, solve the faced scalability issue nor did they consider the hardware implications and port scalability challenges.

In this paper, we propose a scalable switching architecture along with its purely advocated scheduling algorithms to solve both addressed scalability challenges. In particular, we present a scalable switching architecture by building a large scheduler made up of smaller scheduling devices. We also propose a proper scheduling algorithm called Distributed Parallel Round Robin scheduling (DPRR) and an improved version of DPRR, called Distributed Parallel Round Robin scheduling with Memory (DPRRM) for the new architecture. We show that the whole system of scalable IQ switches can be implemented in a practical fashion with current technology.

The rest of the paper is organized as follows: Section II introduces the proposed scheduling architecture and describes its sequential scheduling. Section III introduces two pipeline scheduling schemes; the simple pipeline scheduling and a distributed parallel scheduling. In Section IV, the scheduling algorithm DPRR and DPRRM are described. Section V presents the simulation results and the performance analysis. Finally, Section VI concludes the paper.

II. Scheduling Architecture And Sequential Scheduling

We consider a switch fabric with a large port count, for example a 256 × 256 switch. A single 256 × 256 crossbar switch has 256 × 256 crosspoints. To reduce the physical cost, we use eight separate 32 × 256 chips connected to each other to implement such a 256 × 256 switch. Figure 1 shows the block diagram of a 256 × 256 scheduler built using eight 32 × 256 schedulers. The eight 32 × 256 schedulers are connected one by one and scheduling algorithms are run sequentially on them to implement a 256 × 256 switch. Each 32 × 256 scheduler has 32 inputs connected to a port processor (PP) and 8 schedulers that can link to a total of 256 inputs. The Central Controller is used to control the whole system. A 256-bit control signal is used in the proposed architecture, referred to as available output port (AOP). All nodes 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 and '1' as 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 selected 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 logical '0's. A proper scheduling algorithm will be run on Scheduler_0 and consequently, some of the 256 outputs will have connections with the inputs of Scheduler_0.

The AOP is updated according to the reservation status of the 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. Then, Scheduler_1 selects some 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 the "Scheduling over" signal is sent to the Central Controller as well. All the separate schedulers will continue this scheduling process sequentially. When the Central Controller receives the "Scheduling over" signal from the last scheduler, Scheduler_7, it sends a "Switch enable" signal back to all the scheduling devices to indicate the end of this round of scheduling and enable the transmission of cells from the PPs to the crossbar.

To summarize, this 256 × 256 scalable scheduling architecture works sequentially through AOP signals: Scheduler_0 → Scheduler_1 → Scheduler_2 → ... Scheduler_7. Each of the single schedulers runs a proper scheduling algorithm and reserves some of the outputs in turn.

Figure 2 provides a more detailed example. Assume that we use four 8×32 scheduling devices to implement a 32×32 switching. Suppose the first scheduling device is the start node and it has the highest priority to select outputs from the 32 outputs. At the first cell time, all the 8 inputs of the first device can send their requests to all the 32 outputs and the proper scheduling algorithm is run to select outputs. 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}. AOP2={2, 7, 8, 9, 10, 11, 12, 13, ,21, 23, 24, 25, 26,

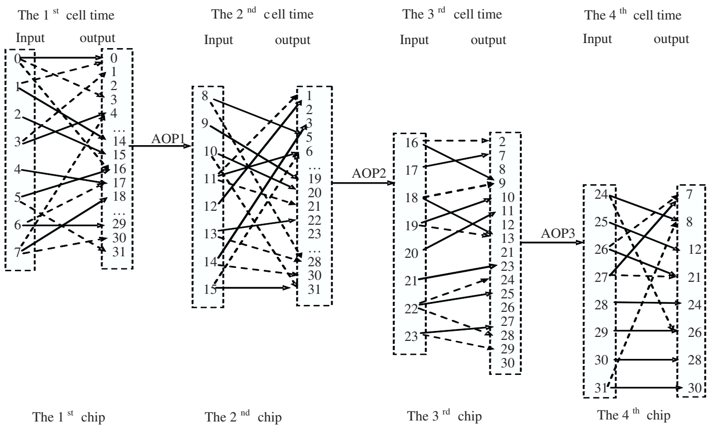

Fig. 2. Processing in scalable scheduling architecture.

27, 28, 30}. Similarly, AOP3 ={2, 9, 10, 11, 13, 23, 25, 27}. When all the 4 schedulers finish the whole round of scheduling, 32 inputs and outputs are completely matched.

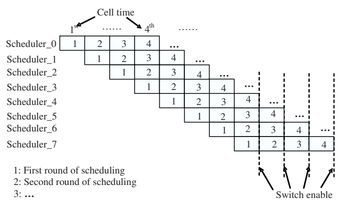

Fig. 3. A simple pipeline scheduling.

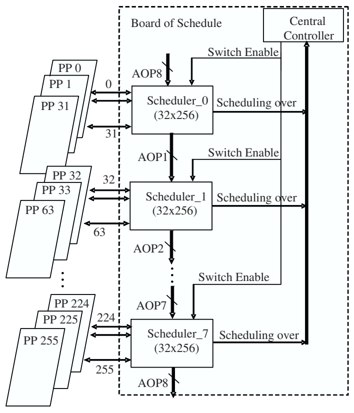

Fig. 4. The architecture of distributed parallel scheduling.

III. 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 reconfigure the crosspoints. A pipeline scheme can be employed to reduce the delay. Figure 3 describes the simple pipeline scheduling scheme. Each single scheduler can enter 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. When Scheduler_0 completes the 1st round of scheduling, it sends AOP1 to Scheduler_1 and then starts the 2nd round of scheduling. As soon as Scheduler_1 receives the signal of AOP1, it starts the 1st round of scheduling. When Scheduler_1 completes the 1st round of scheduling based on AOP1, it can start the 2nd round scheduling right after it receives the AOP1 signal from Scheduler_0.

Once the whole round of scheduling is completed, the "Switch enable" signal will be issued by the Central Controller to each PP, indicating the end of this scheduling round, and the crosspoint switches are reconfigured and each port transmits cells to its designated output in accordance with the selected configuration. This pipeline scheduling scheme makes the scalable scheduling more effective. One round of scheduling can be completed at each cell time, making it the same as a whole large switch.

B. Distributed parallel scheduling

Obviously, the above simple pipeline scheduling scheme cannot guarantee the scheduling fairness for each input. If the highest priority is given to some fixed device (e.g., Scheduler_0 is the start node with the highest priority in Figure 1), this simple pipeline pattern may result in an unbalanced 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. Figure 4 gives the block diagram of the scheduling architecture, in which all single schedulers are connected circularly, i.e. the 1st device is connected to the 2nd device, the 2nd the 3rd device, ..., and the

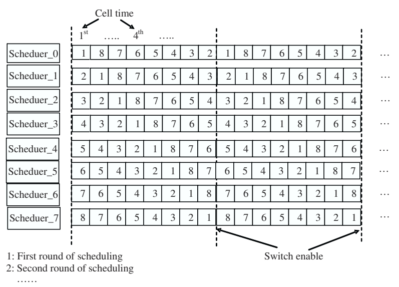

Fig. 5. Distributed parallel scheduling.

last device connected to the 1st device.

Based on this circular connection, a fair pipeline scheduling scheme is proposed. Figure 5 gives the distributed parallel scheduling scheme. All scheduling devices work in parallel, however, they work in different rounds of scheduling. For example, at the first cell time, Scheduler_0, Scheduler_1, ..., Scheduler_7 start their 1st, 2nd, 3rd, ..., 8th round of scheduling respectively. Then, at the second cell time, Scheduling_0 receives the signal AOP8 from Scheduler_7 and continues the 8th scheduling round. Scheduler_1 receives signal AOP1 from Scheduler_0 and continues the work of 1st round scheduling, etc.

In the following cell times, every single scheduler works in a similar way. After eight cell times, each single scheduling device has already completed eight scheduling rounds, 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 "Switch Enable" message from Central Controller, these scheduled cells will be transmitted to the crossbar. 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 a fair scheduling.

IV. Scheduling Algorithms

A. DPRR scheduling algorithm

1) Algorithm specification: Since weight-based algorithms are too complex to be practical, our proposed scheduling algorithm is based on a round robin scheme, which is a 3-step request-grant-accept process. For one scheduling device, it is an $M \times N$ switching, where $M < N$. For example, if we use four scheduling devices to implement a 32×32 switching, each of the scheduling devices is an 8×32 switching. 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. Since the number of inputs is less than the number of outputs, some pointers of the output arbiters cannot avoid synchronization. The ideal case is that on average every four pointers of the output arbiters point to one input so that each input will have equal probability to receive grants, thus increasing instant throughput.

We call our proposed scheduling algorithm Distributed Parallel Round-Robin (DPRR). The three steps in one iteration of DPRR are as follows:

- **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 alternatively, each for one time slot.
- **Step 3**-*Accept*: If an input receives grants, it accepts the one that appears next in a round-robin schedule starting at the highest priority element.

Now we explain the design of DPRR. We consider both uniform traffic and non-uniform traffic. The pointer setting and moving scheme favors Bernoulli i.i.d. uniform traffic, since the pointers are kept balanced 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 is that one round of scheduling requires 4 time slots to complete; thus 4 time slots are period times to update pointers. This feature reacts well under bursty traffic conditions, since cells arriving within the same burst might be served continuously and any burstiness delay will be reduced.

As for non-uniform traffic, two typical traffic patterns are hot-spot and diagonal. The hot-spot pattern assumes one output to be the "hots-pot", and the traffic load from all the inputs to this "hot-spot" is higher than to other outputs. In our experiment, Output 0 is the hot-spot with the highest rate (50%) of traffic destined to it, and all other traffic is distributed to other outputs uniformly. Another typical non-uniform traffic pattern is diagonal traffic, where input i sends 2/3 and 1/3 of its load to outputs i and i + 1, respectively.

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 more than 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 there is no request from input 3 which 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.*

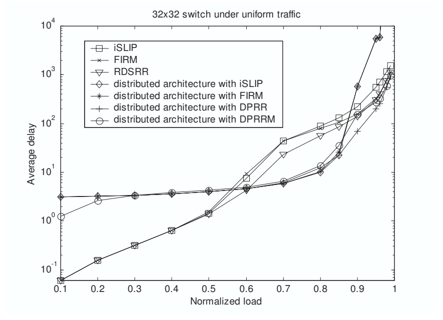

Fig. 6. Average delay under Bernoulli uniform traffic.

Proof: We assume that the offered load is 100% uniform traffic, so that every VOQ is always occupied. Let us consider a 16×16 switch, composed of four 4×16 scheduling devices. Since the pointers of the output arbiters are desynchronized in such a way that 4 outputs point to one input of each 4×16 scheduling device and the scheduling is 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 output pointers pointing 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. Therefore, each input of Scheduler_1 chooses one output from 3 candidates and sends accept.

The same applies to Scheduler_2, each input chooses one output from two candidates and sends 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, all output pointers will increase by one (module 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., which affects the queues occupancies; exploiting the arrival property is a good way to improve performance and ensure stability. To this end, we modified DPRR to make it stable which results in the improved algorithm, distributed parallel round robin scheduling with memory (DPRRM).

The following is the specification of DPRRM. Let $\mathcal{S}(t$ - 1) be the schedule used at previous time slot and $\mathcal{A}(t)$ is the matching obtained from arrival. To obtain $\mathcal{A}(t)$, first we construct arrival graph $\mathcal{G}(t)$, if there is an arrival from input i to output j, then we add an edge from input i to output j. If $\mathcal{G}(t)$ is a matching, then $\mathcal{A}(t)$ = $\mathcal{G}(t)$. If $\mathcal{G}(t)$ is not a matching, which means there is more than one arrival to one output, in this case, we choose the heaviest edge and remove the others for this output to obtain a matching $\mathcal{A}(t)$ from $\mathcal{G}(t)$.

Let $\mathcal{S}(t) = \arg\max_{\mathcal{S} \in \{\mathcal{S}(t-1), \mathcal{A}(t), \mathcal{D}(t)\}} < \mathcal{S}, \mathcal{Q}(t) >$, where $\mathcal{D}(t)$ is the DPRR matching and $\mathcal{Q}(t) >$ is the queue-lengths matrix. In the first several time slots, there is no matching from DPRR, we set all the elements of $\mathcal{D}(t)$ 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 $\mathcal{A}(t)$, which is derived from the arrival graph, as one of the probe matchings [7]. The arrival process is stationary and Bernoulli i.i.d. Hence, there is a finite probability > 0 such that $\mathcal{A}(t)$ is the MWM. According to [8], this is sufficient to prove the stability of DPRRM. ■

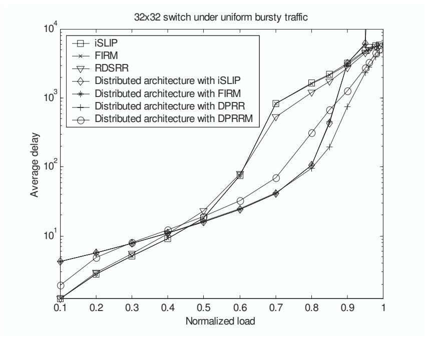

Fig. 7. Average delay under Bursty uniform traffic.

V. Simulation Results

In our simulation, we consider a 32×32 switch. In our proposed parallel scheduling architecture, we use four scheduling devices, each of which is a 8×32 switching to connect with each other. The traffic is admissible Bernoulli i.i.d. Uniform traffic, uniform bursty traffic (with burst length of 10 cells) and two non-uniform traffic patterns, namely the diagonal and hot-spot. The algorithms are executed using one iteration.

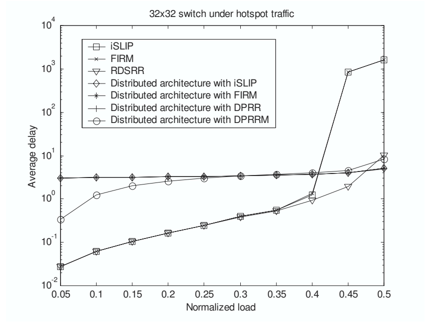

Fig. 8. Average delay under Hot-spot traffic.

Figure 6 illustrates the average delay performance of various algorithms under uniform traffic. From the figure, we can see that when the load is low, the delay of 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. This results in an initial latency, which depends on the number of scheduling devices used in the architecture. After that, cells are transmitted one cell time after another. Practically, it is equal 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, thus the delay is slightly lower than DPRR when the load is low. When the load is above 0.6, our proposed architecture with DPRR is much better than all the other algorithms, where iSLIP, FIRM, RDSRR run on a single 32×32 switch and iSLIP, FIRM run on the proposed distributed architecture. As we mentioned above, the pointer moving schemes of iSLIP and FIRM are not suitable for our proposed scheduling architecture as they tend to synchronize heavily. With DPRR, the pointers synchronize in a balanced way and the advantage of output reservation in turn by our proposed scheduling architecture is also shown. DPRRM has similar performance to DPRR.

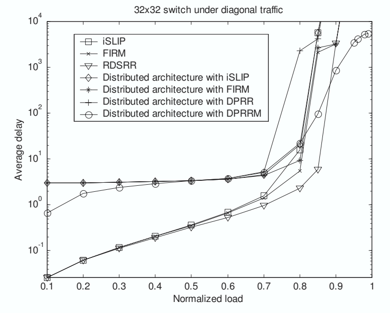

Fig. 9. Average delay under Diagonal traffic.

Figure 7 depicts the cell delay under bursty uniform traffic. It is similar to the case of uniform traffic. Under high traffic loads, the proposed scheduling architecture with DPRR has very good performance, since the pointer movement scheme makes it possible to send cells within a burst continuously. DPRRM maintains the same good performance as DPRR. Figure 8 shows the simulation results under hot-spot traffic. The delay of our proposed architecture is close to a constant in all ranges. Only in the highest load range does it increase slightly, however it is still much lower than iSLIP, FIRM and RDSRR running on a single switch. The delay of iSLIP and FIRM running on the proposed architecture is similar to DPRR and DPRRM running on the proposed architecture, since our proposed architecture ensures the service of the "hot-spot" all the time with high probability. In other words, the architecture favors hot-spot traffic.

Figure 9 describes the delay performance under diagonal traffic. When the traffic is high, the delay performance of our proposed architecture with DPRR is close to those algorithms run on a single switch, even slightly worse.

VI. Conclusion

A scalable switching architecture is crucial when the switching capacity becomes large (128 ports and beyond). In this paper, we proposed a fair scalable scheduling architecture, which employs distributed parallel pipeline scheduling for IQ switches. Using this scalable scheduling architecture, a large scheduler can be constructed by connecting several smaller schedulers. We also proposed a round-robin scheduling algorithm named DPRR and an improved version (DPRRM) for the proposed architecture. Our architecture employs the in turn reservation of outputs scheme, which increases the instant throughput and decreases bandwidth waste. The pointer setting and moving scheme of DPRR reduces pointer synchronization and thus 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. DPRRM exhibits good performance while being stable under any admissible input traffic.

References

[1] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. on Comm.*, vol. 35, no. 09, pp. 1337–1356, Dec. 1987.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. on Comp. Sys.*, pp. 319–352, Nov. 1993.

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

[4] ——, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches," *IEEE INFOCOM*, pp. 792 – 799, Apr. 1998.

[5] N. McKeown, "iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Trans. On Net.*, vol. 07, no. 02, pp. 188–201, Apr. 1999.

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

[7] P. Giaccone, "Queueing and Scheduling Algorithms for High Performance Routers," Ph.D. dissertation, Politecnico Di Torino, Feb. 2002.

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

[9] A. Smiljanic, R. Fan, and G. Ramamurthy, "RRGS-Round-Robin Greedy Scheduling for Electronic/Optical Terabit Switches," *IEEE Globecom*, pp. 1244–1250, 1999.

[10] A. Smiljanic, "Flexible Bandwidth Allocation in Terabit Packet Switches," *IEEE HPSR*, pp. 233–239, 1999.

[11] E. Oki, R. Rojas-Cessa, and H. J. Chao, "PMM: a pipelined maximal-sized matching scheduling approach for input-buffered switches," *IEEE Globecom*, pp. 35–39, 2001.