A Hardware NIC Scheduler to Guarantee QoS on High Performance Servers[*]

J. M. Claver, M. Canseco, P. Agustí, G. León

Department of Computer Science and Engineering,
University Jaume I, 12071-Castellón, Spain,
{claver, canseco, agusti, leon}@icc.uji.es

Abstract. In this paper we present the architecture and implementation of a hardware NIC scheduler to guarantee QoS on servers for high speed LAN/SAN. Our proposal employs a programmable logic device based on an FPGA in order to store and update connection states, and to decide what data stream is to be sent next. The network architecture is connection-oriented and reliable, based on credit flow control. The architecture scales from 4 to 32 streams using a Xilinx Virtex 2000E. It supports links with speeds in the order of Gbps while, maintaining the delay and jitter constrains for the QoS streams.

1 Introduction

Nowadays, QoS requirements on high performance system/local area networks (SAN/LAN) make necessary the use of complex routers to offer QoS hardware support between all their components. The MultiMedia Router (MMR) architecture [1] is an example of this sort of routers, designed as an interconnection compact component to be used in cluster and LAN/SAN environments. However, this router requires a high number of virtual channels (VC) per port (128) and a complex scheduler associated with every link, which maintains and updates connection states, and selects packets to be sent.

A way to reduce router complexity and to optimize its design, while meeting QoS constraints, consists in improving traffic scheduling in servers [10]. The main idea is to reduce the number of buffers and VCs, taking advantage of QoS traffic sorting carried out by servers. Thus, both scheduling of routers and servers could work to wire speeds. To do that, it is necessary to use a hardware implementation to accelerate the scheduling process. However, the complexity of stream selection and priority updating poses an important implementation problem for scheduling a large number of streams using Gigabit links.

In order to achieve these goals we propose a NIC (Network Interface Controller) scheduler architecture for servers with QoS support. Previous efforts [11,13,9] have proposed priority queuing architectures for switches and NICs. Our architecture is inspired on link schedulers designed for the MMR router. A simplified form of this

[*] This work was partially supported by the Spanish CICYT grant TIC2003-08154-C06 and the European FEDER program.

router has been completely implemented on an FPGA, denoted as Simple MMR (SMMR) [4]. The SMMR has been conceived as a prototyping platform, adapting MMR design ideas to a more realistic chip area and clock rate constraints. Furthermore, SMMR allows design alternatives for the evaluation and optimization of the initial MMR architecture [6]. A first prototype based on this architecture is proposed for a hardware traffic generator to test the SMMR.

Our NIC scheduler is inspired on a previous work [9], which formulates a hardware solution for DWCS scheduling [14], using an *in chip* network processor and an FPGA on a board. On the other hand, our NIC architecture is a structured design which allows posterior improvements and optimizations. Also, the network architecture used in our case study allows implementing more components of the NIC scheduler in an FPGA. Only few tasks are coordinated by the server host processor in our implementation. These tasks will be performed by an FPGA embedded network processor in a future version.

The remainder of this paper is structured as follows. In the following section the network architecture is presented. In section 3 we review the architecture of the QoS hardware scheduled server. The hardware architecture and implementation of the NIC scheduler is described in section 4. In section 5 we report details on the experimental prototype and the performance evaluation. Finally, section 6 summarizes the results of this work and overview future lines of research.

2 Network Architecture

Our server NIC scheduler is designed for high performance SAN/LAN and server clusters. These environments produce heterogeneous traffic streams, best effort (BE) and QoS (constant bit rate, CBR, and variable bit rate, VBR), which have different bandwidth and latency requirements. We consider a network architecture in which data packets are partitioned into *flits* (flow control units) of 1024 bits. Every flit is composed by 64 phits, where a *phit* (16 bits) is the minimum information unit transferred through a network link. The transmission bitwidth used internally in the NIC scheduler will be 32 bits, which is the same bitwidth that the external memory words.

The establishment of QoS connections is carried out using an adaptive seeking algorithm of minimum path based on backtracking and denoted as Exhaustive Profitable Back tracking (EBP) [8]. For this purpose, two control packets, connection and disconnection, are used. Flow control used in this sort of traffic is PCS (Pipelined Circuit Switching). Best effort traffic uses a connectionless-oriented scheme, and a Virtual Cut Through (VCT) switching control. A Credit-based flow control, as in Infiniband [12], is used to avoid channel saturation and data loss. This technique consumes link bandwidth, which can be reduced by increasing flit size, and requires much smaller buffers.

Link communications are serial and data are synchronized in blocks of 1040 bits ((64+1) phits or 1flit + 1phit) that we denote as “flit frame”. Thus, a flit frame is divided in two parts, one corresponding to data, network control or synchronization flits, and another in which a credit may be sent.

3 Scheduler Architecture

A NIC scheduler for a server with QoS support must maximize link throughput, at the same time that guarantees real time requirements and avoids starvation of BE traffic. In the design of our NIC scheduler, we take into account the following strategies in order to achieve these goals: an admission control and bandwidth distribution to maximize the link throughput, an adaptive priority system to obtain QoS guarantees, and an inhibition mechanism on QoS streams, which avoids starvation of best effort traffic and controls link bandwidth usage.

In order to parameterize bandwidth required by all sorts of traffic (VBR, CBR and BE), time space is divided in multiples of a flit frame that we denote as "flit round" or just "round". The number K of flit frames per round determines the minimum bandwidth assigned to a traffic flow ($1/K$). In our experiments we set K=2048.

Admission and bandwidth assignment policies vary in function of traffic characteristics. For CBR traffic, only the bandwidth used is needed, which cab be expressed as the maximum number of flits that can be sent in a round. As VBR traffic bandwidth can vary from one round to the next, a statistic model is used to obtain medium and peak bandwidths, and to determine an adequate bandwidth reservation. Finally, remaining bandwidth is assigned to best effort traffic.

In this first scheduler model, VBR connexions are considered as CBR. Thus, to avoid packet loss and overtake delay requirements that will appear with statistics models, its peak bandwidth (PBR) is used to reserve bandwidth in scheduler. An adaptive priority system that guarantees QoS requirements is based either in delay or jitter. In this case, a priority algorithm based on bandwidth and delay has been selected, the SIABP (Simple-IABP) algorithm [2]. The algorithm has good behavior under high traffic loads and it is the core of the link scheduling algorithm in the MMR router [3]. The algorithm also increases the priority of packets proportionally to its waiting time in the buffer input queue. This design needs a regulating mechanism to control the correct use of bandwidth reserved for every stream, avoiding starvation of BE traffic. Therefore, for CBR traffic, the deadline to send a flit (T_Delay) is determined as a function of its transmitting pace (BW_{link}/BW_i , where BW_{link} is the link bandwidth and BW_i is the bandwidth reserved for the VC_i) in a round and its accumulated delay.

Taking into account the previous parameters, the order in which streams (S_i) are served follows Algorithm 1. Among all QoS streams ready to be sent – those with remaining bandwidth in the current round ($BW_R > 0$), buffer space in the receiver VC (credits), and satisfying its bandwidth constrains (T_Delay ≤ 0) – the highest priority stream is select to be sent. When there is more than one stream with the highest priority, any one of these streams is selected. The implementation of this algorithm is widely detailed in section 4.

A QoS server needs a high speed I/O interface to connect the storage system with the NIC scheduler board, providing the required bandwidth for every stream. New standards as the PCX commuted interface could provide the necessary bandwidth for current link bandwidths. Figure 1 shows the system architecture proposed for our QoS hardware scheduled server.

The system architecture is organized in two levels. The first level is carried out by server host processor, which provides first stage of admission control, initiate connection setup, stream identification and scheduling of stream buffers.

Algorithm 1. Scheduler stream selection

```
for all streams $S_i \in \{CBR\}$, $i=1,2,\ldots,n$ /*In parallel
/*Builds the priority vector V
    if {$S_i$.T_Delay$\leq 0$ & $S_i$.Credits$>0$ & $S_i$.BW$_R>0$}
        $V_i$= $S_i$.priority; /*$S_i$ participates in scheduling
    else
        $V_i$= 0; /*$S_i$ does not must send data
    Endif
end for
/*Selects the highest priority stream in $\log_2 n$ steps
{$V_H$}= {highest($V_k$)}, $\forall V_k \in V$; /*$V_k$ with the high priority
if {card({$V_H$}) == 1} Out= $H$;/*There is only one $V_H$
else Out= Any({$H$});/*Selects any one of the highest $V_H$
endif
```

The second level of this architecture is completely implemented on an FPGA, coordinating admission control with the host processor, managing stream scheduling, formatting and transmitting flits from stream buffers. So, the FPGA not only performs scheduling tasks, but also provides real time functions that were previously assigned to a network processor.

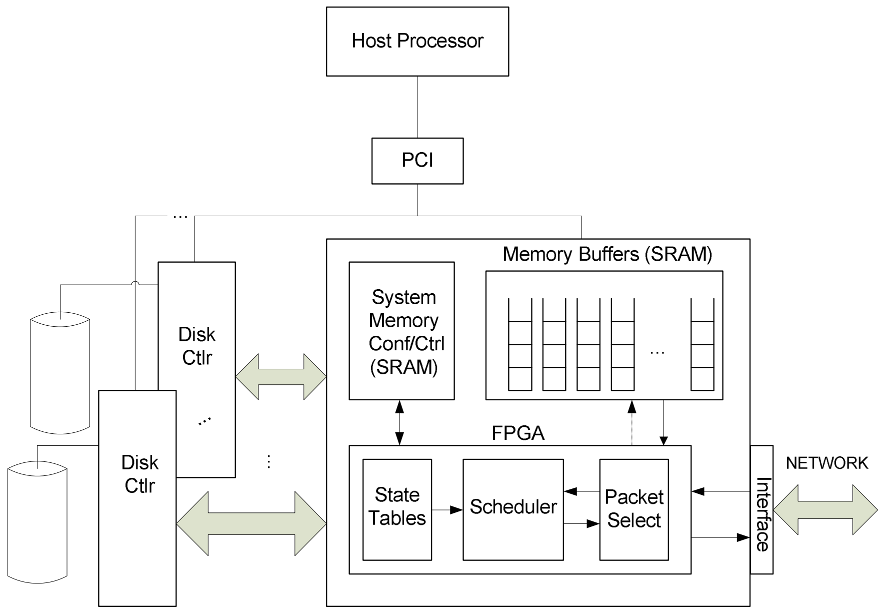

Fig. 1. System architecture of a QoS hardware scheduled server.

This distribution of tasks is due to timing strong requirements. Management of scheduling needs acceleration due to the temporal bounds and the fact that the communication with the network interface demands strict synchronization. Alternatively, this could be implemented using an ASIC. However, to keep pace with the constantly changing algorithms, standards and protocols, it is preferable to use reconfigurable logic.

4 Hardware Architecture and Implementation

The hardware architecture and implementation of our NIC scheduler combine the use of an FPGA board, a network interface, and a system of disks. The components implemented in the FPGA perform several tasks: communication with the server host processor, stream scheduling, and data input/output (see Figure 2).

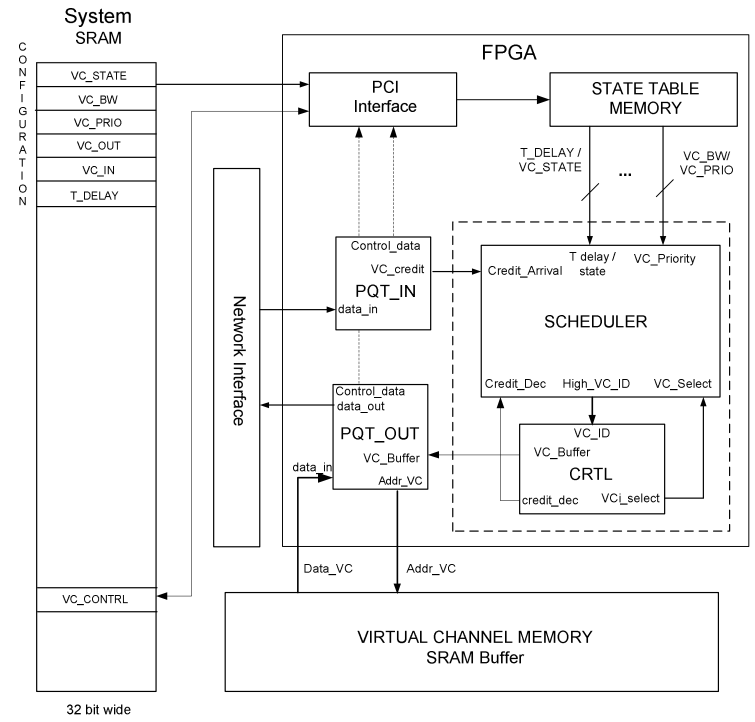

Fig. 2. Internal hardware and interfaces of an FPGA Server NIC scheduling prototype.

Connection setup is carried out by the host processor, loading parameters such as stream type (CBR, VBR), reserved bandwidth, transmitting pace (T_Delay), and VCs involved on a stream. As data packets are transmitted, a bidirectional communication host-FPGA controls VC buffers, interchanging data and control information. A flit arrival through the network interface is immediately communicated to the host. The host processor also indicates to the FPGA the end of a communication and updates its state. This is followed by the FPGA sending a disconnection flit. As it is described in section 2, communications are synchronized using a “flit frame”, composed by 64+1 phits.

The input module (PQT_IN) controls the input link, identifying received packets: control flits, synchronization flits, and flow control credits. When a control flit arrives, the PQT_IN resends this to the host. When a synchronization flit is received, PQT_IN does nothing. In the last cycle of a flit frame a credit can be received from the router connected to the server. Then, the PQT_IN module informs of the credit arrival to the host through the PCI interface and to the SCEDULER module as well, increasing the corresponding credit counter of this stream.

The output module (PQT_OUT) builds the head of a flit and sends it, followed by data from the corresponding VC buffer. PQT_OUT obtains data directly addressing VCs stored on the SRAM memory of the FPGA board. For each flit that is sent, PQT_OUT informs the host processor to manage its associated VC buffer.

The scheduling algorithm is divided in two steps. First, all the sub-schedulers work in parallel to generate a global priority vector to be sent, where every sub-scheduler contributes with its VC identification and current priority. Second, the high priority VC is selected from the priority vector by a MAX bitonic network module (see Figure 3).

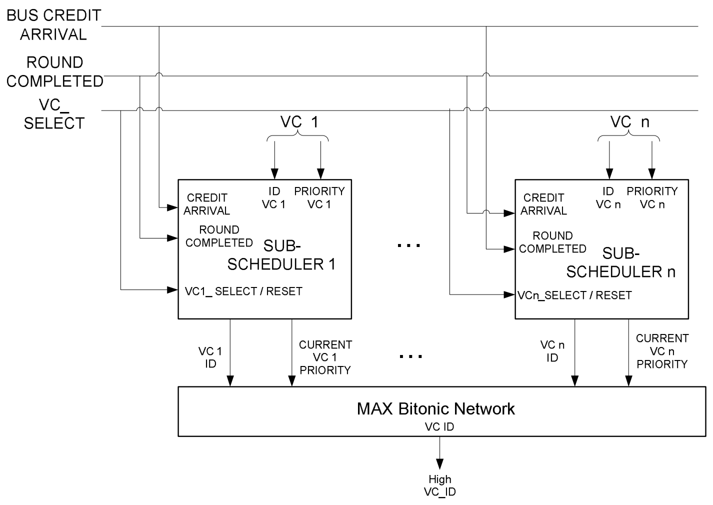

Fig. 3. Scheduler architecture composed of a sub-scheduler for every QoS data stream and of a MAX bitonic network.

This proposal employs an implementation of a local scheduler (sub-scheduler) for every VC, which is composed of two modules, working synchronously in parallel (Figure 4). The SIABP priority module [2, 6] stores the priority of a VC, which is directly related to its bandwidth reservation as the number of flits that can be sent per round. After a scheduling process, all the priorities are updated. If a VC is selected to send its flit into the next flit frame, its priority is reset. Otherwise, its priority is increased as a function of its waiting time (for further details, see [1]).

The candidate selection is performed by a bitonic network (MAX) that selects the high priority VC from the priority vector generated by the sub-schedulers. The MAX module takes $\log_2 n$ cycles and has a building cost in LUTs area of $O(n)$, where n is the maximum number of VCs per output link. It is possible to reduce the area of this module to either $O(n/2)$ or $O(\log_2 n)$, building a recirculating shuffle-exchange network as in [9]. Nevertheless, the resulting critical data path and complexity is then increased due to the presence of new multiplexers. This complexity is critical for the architecture of an FPGA, the building area is not significantly reduced, as the resulting delay clock is increased.

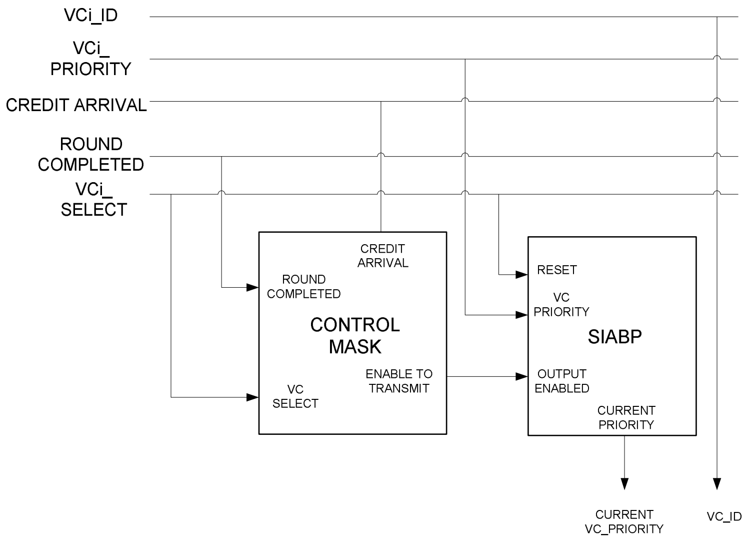

Fig. 4. Sub-scheduler architecture composed of a control module (CONTROL MASK) and of a SIABP priority module.

The CONTROL MASK module (see Figure 5) controls the remaining bandwidth of a stream, the bank of credits, and the transmitting pace using the parameter T_DELAY. Using this information, this module decides if a VC must appear in the priority vector. If a VC has its CBR RATE counter set to a value less than 1, its BW REMAINDER counter greater than 0, and its CREDIT BANK counter greater than 0, the signal ENABLE TO TRANSMIT is set. At the same time the module adds its current priority to the priority vector. The priority added is zero only when one of the previous conditions is not satisfied.

The control module (CTRL) is the last step in the scheduling process. It synchronizes the scheduler and PQT_OUT modules, and actives signals to update internal registers and scheduler counters. Thus, the total router cycles required for scheduling are $10 + \log_2 n$, where the 5 cycles are devoted to local stream scheduling, $\log_2 n$ cycles are spent on stream selection and 5 cycles more are consumed updating stream priorities.

5 Evaluation

This section presents the performance evaluation of our NIC scheduler architecture and its hardware implementation using a Celoxica RC1000 board [7]. This board has an FPGA Xilinx Virtex 2000E (38,400 LUTs and 640 Kb BRAM), 8 MB SRAM memory distributed in four independent banks, and two PCI Mezzanine (PCM) I/O cards.

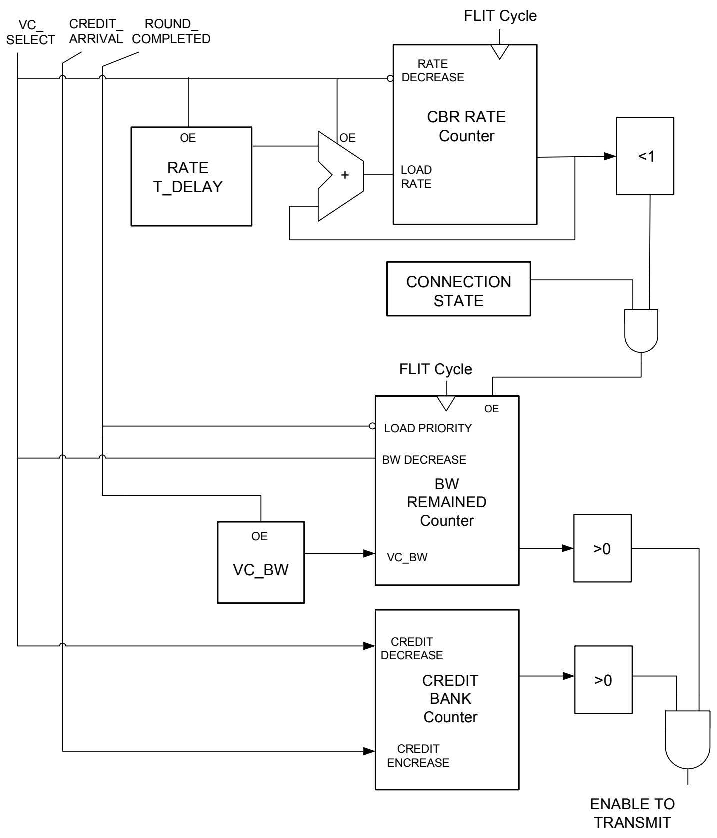

Fig. 5. Control module architecture (CONTROL MASK).

The components implemented into the FPGA carry out several tasks: communications with the host processor, scheduling of streams and data input/output. Communications between the FPGA and the host processor use DMA transferences between the host/PCI peer and the 32bit/33 MHz PLX controller of the RC1000 board.

In order to simplify the design process, the FPGA Virtex has been programmed using HandelC [5] and the Celoxica DK4 environment, and the ISE 6.1 Xilinx backend tools from Xilinx. This eases prototyping our design, and the updating process. HandelC is a behavioral C based hardware description system developed by

Celoxica that allows co-simulation. Parallelism of process and synchronization are taken from the CSP model, in particular, from the Occam language. HandelC uses standard data types with user defined bitwidths. Thus, HandelC provides an efficient use of hardware resources.

5.1 Area and Delay Results

For this evaluation we have scaled our design from 4 to 32 virtual channels. Thus, we can evaluate its behaviour and performances with a different number of streams. Figure 6 reports the area size and clock rate for 4, 8, 16, and 32 data streams. SRAM memory used for interface and VC buffers are not included in this table. As the number of streams handled by the NIC scheduler increase from 4 to 32, the number of LUTs utilized by the implementation grows linearly.

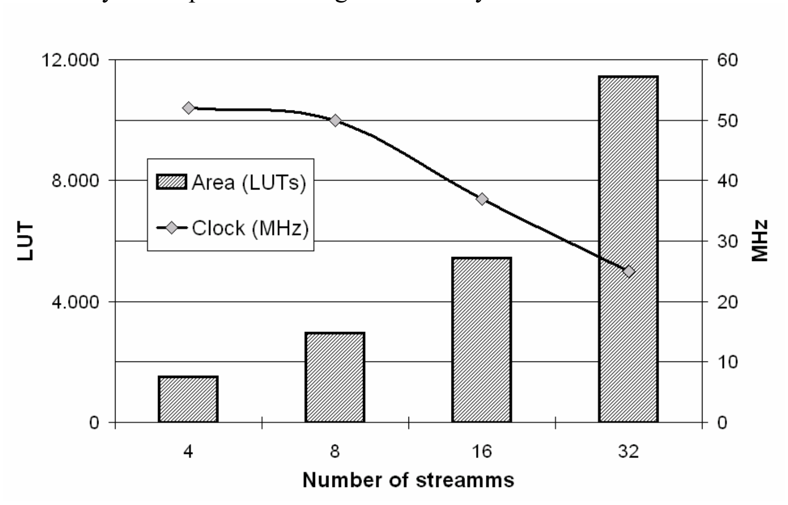

Fig. 6. Area and clock rate characteristics for different QoS server configurations.

The maximum clock rate for a 4 streams configuration is 52 MHz, which decrease down to 25 MHz for the configuration with 32 streams. A detailed time analysis shows that maximum clock rate is limited by the sub-scheduler module and not by the selection module MAX. However, the maximum clock rate decreases logarithmically, showing that our NIC scheduler implementation can meet packet-time requirements on Gigabit links.

5.2 Traffic Analysis

In order to highlight performance results of our QoS NIC scheduler, we have compared it with another scheduler without QoS characteristics, based on a Round-Robin scheduling algorithm, and implemented in an FPGA too. We perform two experiments to illustrate bandwidth distribution among data streams. In experiment 1, four streams with equal bandwidth division ratios (1:1:1:1) were scheduled. In the results the X-axis reports scheduled output time expressed in flit cycles (each flit cycle is equivalent to 65 router cycles), and the Y-axis reports the bandwidth as the number of flits scheduled per unit of time. We only report the first 100 flit cycles of simulation (65,000 router cycles). Figures 7 and 8 show the same bandwidth division for the two schedulers. The bandwidth of each stream settles after 70 time units and maintains the same value until the execution is terminated. So, bandwidth for every stream is guaranteed and output link throughput is 100%.

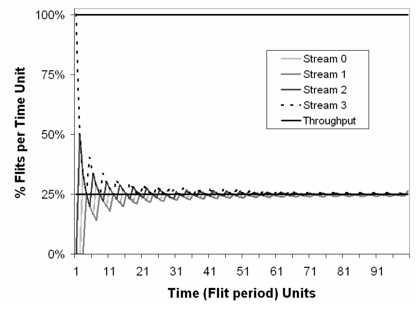

Fig. 7. Bandwidth distribution results for four streams with equal ratios (1:1:1:1) using the QoS scheduler.

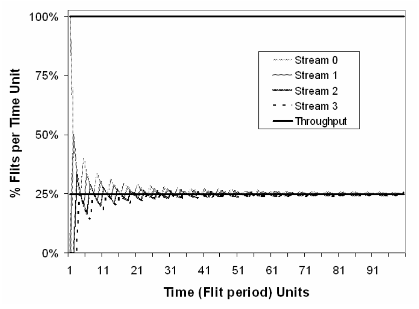

Fig. 8. Bandwidth distribution results for four streams with equal ratios (1:1:1:1) using a Round-Robin scheduler without QoS.

A second experiment with bandwidth division ratios (4:2:1:1) was also recorded. Figures 9 and 10 report a different bandwidth distribution for the two schedulers. When the QoS scheduler is used, the bandwidth of each stream settles after 70 time units and maintains the same value until the execution is finished. So, bandwidth for every stream is guaranteed and output link throughput is 100%.

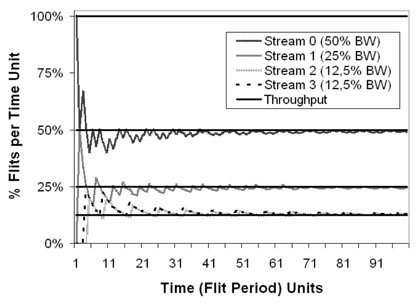

Fig. 9. Bandwidth distribution results for four streams with ratios (4:2:1:1) using the QoS scheduler.

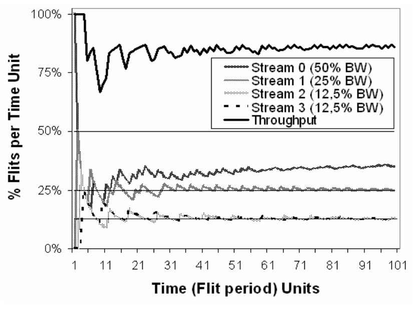

Fig. 10. Bandwidth distribution results for four streams with ratios (4:2:1:1) using a Round-Robin scheduler without QoS.

However, as is reported in Figure 10, bandwidth for every stream is not guaranteed when the Round-Robin scheduler is used, and output link maximum throughput is below 86%. Although bandwidth for streams with low requirements is guaranteed, bandwidth for the stream with high requirements is not.

6 Conclusions

In this paper we have presented the architecture and hardware implementation of a hardware NIC scheduler with QoS guarantees for servers on high speed LAN/SAN and high performance clusters. The proposed architecture is scalable because the occupation area grows linearly as the number of data streams is increased, and the NIC scheduler clock rate decreases logarithmically and tends to be stabilized around the 20 MHz. The QoS server behaviour guarantees bandwidth and delay requirements, attaining network link throughput close to 100%. NIC scheduler has been implemented in an FPGA and using a high level specification language like Handel-C.

Currently, we work in the design of a QoS server that handles combined CBR and VBR traffic with better efficacy and takes higher profit of maximum link throughput. Finally, we also work on increasing the clock rate of the NIC scheduler to adapt it to multi-gigabit data links.

References

1. Caminero, M.B.: Diseño de un Encaminador Orientado a Tráfico Multimedia en Entornos LAN. Phd. Thesis (in Spanish), Albacete, (Jun 2002)
2. Caminero, M.B., Carrión, C., Quiles, F.J., Duato, J., Yalamanchili, S.: A Cost-Effective Hardware Link Scheduling Algorithm for the Multimedia Router (MMR). Lecture Notes in Computer Science, Networking (ICN'01) (2001)
3. Caminero, M.B., Carrión, C., Quiles, F.J., Duato, J., Yalamanchili S.: Investigating switch scheduling algorithms to support QoS in the Multimedia Router (MMR). Proceedings, Workshop on Communication Architecture for Clusters (CAC'02), IEEE Computer Society (2002)
4. Canseco, M., Claver, J.M., León, G., Vilata, I.: Primeras Experiencias en la implementación de un encaminador QoS sobre una FPGA. IV Jornadas de Computación Reconfigurable y Aplicaciones, Bellaterra (Spain) (in Spanish) (Sep 2004)
5. Chapell, S., Sullivan, C.: Handel-C for co-processing an co-design of field programmable systems on chip. Proceeding of the JCRA'02 (2002)
6. Claver, J.M., Carrión, M.C., Canseco, M., Caminero, M.B., Quiles, F.J.: A New Hardware Efficient Link Scheduling Algorithm to Guarantee QoS on Clusters. Lecture Notes in Computer Science, Euro-Par 2005 Parallel Processing: 11th International, Ed. Springer-Verlag (2005)
7. Celoxica: RC1000 Software reference manual. (2001)
8. Gaughan, P.T., Yalamanchili, S.: Adaptive routing protocols for direct multiprocessor networks. IEEE Computer (May 1993)
9. Krishnamurthy, R.,Yalamanchili, S., Schwan, K., and West, R.: Architecture and Hardware for Scheduling Gigabit Packet Streams. Proceedings of the 10th Symposium on High Performance Interconnects Hot Interconnects (HotI'02) (2002)

10. Martínez, A., Alfaro, F.J., Sánchez, J.L., Duato, J.: Providing Full QoS Support in Clusters Using Only Two VCs at the Switches. XII IEEE International Conference on High Performance Computing (HiPC), Goa (India) (Dec 2005)
11. Rexford, J.L., Greenberg, A.G., Bonomi, F.G.: Hardware-efficient fair queuing architectures for High speed networks. Proceedings of the IEEE INFOCOM '96, San Francisco (Mar 1996).
12. Shanley, T.: Infiniband Network Architecture. Addison-Wesley (2003)
13. Moon, S., Rexford, J., Shin, K.: Scalable hardware priority queue architectures for high speed packet switches. IEEE Trans. on Computers, V. 49 No. 11, pp. 1215-1227 (2000)
14. West, R. and Poellabauer, C.: Analysis of a Window-Constrained Scheduler for Real-time and Best-Effort Packet Streams. Proceedings of the 21st Real Time Systems Symposium (Nov 2000)