Implementation of an NIC for Virtualization Servers to Support the High-Speed Feature of Virtual Machine Networking

Changsu Kim¹, Kiwoong Jung², and Hoekyoung Jung³*
Department of Computer Engineering, Pai Chai University, Doma2-Dong, Seo-gu, Daejeon, South Korea.

²*Correspondent Author

¹ORCID ID :0000-0002-2020-3142 & ²Scopus Author ID: 55804239300

Abstract
As the amount of Internet traffic continues to increase, the number of servers comprising a network is rapidly increasing and servers are fast changing to mass storage, high-speed and virtualization ones. This requires a high performance Network Interface Card (NIC) to support virtualization environments, and the parallel processing of high speed data including virtual flows needs to be highly efficient in order to accommodate the mass storage and virtualization of servers.

With such a background, this study suggests an NIC with the feature of a high efficiency parallel processing of high speed data by utilizing core affinity in multi-core environments, along with the feature of switching among virtual machines and the feature of an overlay tunneling engine. This resolves the problem of performance degradation in virtualization servers, which is a key issue for virtualization servers. It is possible to make a configuration to support the feature of virtualization servers and the feature of the QoS guarantee for scalable services when the feature of switching among virtual machines and the feature of a network protocol tunneling engine are implemented within an NIC, followed by interlocking these features with the feature of multi-core affinity utilization[2,5].

The introduction is followed by Chapter 2, which describes the content of system design. Chapter 3 provides an explanation on implementation and performance test, and Chapter 4 provides a conclusion.

SYSTEM DESIGN
This chapter describes the content of the entire system design. A multi-core based high-performance support system needs an embedded Linux OS to develop high performance OS features in multi-core based environments, and the entire system configuration is shown in Figure 1.

![Figure 1. The Entire System Configuration](image-url)
Network Traffic Monitor (NTM) :

Network Traffic Monitor (NTM) analyzes information on packets by monitoring network traffic, collects process information based on the information analyzed before, and thus serves a role to register and renew the network process information.

This consists of Packet Capture and Parser, which monitors and classifies packets, processes headers on protocol, and then generates access information (ip, port) to request renewal of process information on packets; Process Socket Info Manager, which generates information matched with the inode information of sockets by collecting information on sockets of each process; TCP Connection Info Manager, which collects the ip, port, and inode information of the TCP currently accessed; and Network Process Manager, which generates hash keys from the process information of packets captured by packet monitoring, finds inode information on the basis of the generated hash keys, and searches process or renews process information on the basis of the found inode information, as shown in Figure 2.

The major functions and features of each block are as follows.

1) Packet Capture and Parser

① init
- Calls determine_default_device and initializes the devices of the network devices list
- Extracts the IP information of generated network devices and renews the network devices information
- Generates the capture handler on the generated network devices list
- Registers a callback function in the handler for each protocol
- Sets nonblock not to be blocked at the time of packet capturing

② dp_dispatch
- Registers the pcap_callback function in the pcap_dispatch function
- Packet capture starts.
- Calls the pcap_callback function after packets are captured
- Checks the linktype of packets and in the case of ethernet, removes the ethernet header
- Checks if there is an IP protocol by identifying the ether_type, and when an IP protocol is found, collects the IP and PORT information
- Removes the IP information
Checks the ip_p information and when TCP is found, generates packet objects with the TCP information.

Calls the findConnection feature of Network Process Manager.

2) Process Socket Info Manager

① reread_mapping

- Collects the File Descriptor (FD) information in the process directory under /proc
- Separates only the socket information in the collected FD information, and maps the inode value and PID of each socket.

3) Connection Manager

① refreshconninode

- Reads the /proc/net/tcp files and generates hash information with IP:PORT-IP:PORT
- Maps the access information and inode information by matching the generated hash information and the inode information.

4) Network Process Manager

① reproc

- Corrects information when the mismatch between connection information and process information occurs as to the connection information transferred in a different process.
- Imports the process information in the process list.
- Checks the inode and pid values of process information.
- Renews information in the case of different information when compared with the existing information.

② findConnection

- Checks whether packet information has the same information with the existing information, and when the same information is found, returns the connection information.
- When the same information is not found, returns null.

③ connection_add

- Adds packet information to the existing connection information.

④ getProcess

- Generates new access information.
- Calls reread_mapping and renews process socket info (inodeproc).
- Calls refreshconninode and renews the existing TCP Connection Info (conninode).
- Searches inode information from hash information, and if not found, inverts hash information, and then searches again.
- If not found, registers and adds inode information in the unknown process; if found, renews the information in the relevant process information.

Process Optimized Core Scheduling (POCS): Process Optimized Core Scheduling (POCS) classifies concentration levels according to the network usage amount (frequency) in the network process; based on the classified concentration levels, it sets a priority and core affinity at the optimal core. And when the concentration level is low, POCS reverts the priority back to what it was, accordingly, and also reverts the core affinity back to what it was before optimization.

The purpose of a multi-core based high-performance support subsystem is to enhance the performance of a multi-core system through an optimized core scheduling of the network process. Search for optimized core, change/recovery of process priority and setting/recovery of affinity are conducted for higher performance of the network process.

POCS consists of Network Concentration Level Measurement, which analyzes a network usage amount in Network Process Info to compute a concentration level appropriate for system performance, and Core Affinity Management, which identifies the optimized core for each process and sets an affinity level according to the network concentration level, and then changes the priority. The block configuration is shown in Figure 3.
The major functions and features of each block are as follows.

1) Process Optimized Core Scheduling

①init
- Resisters alarm_cb with the callback function for SIGALRM, using the signal function (Registers alarm at the time after one second.)
- Conducts an initialization process so that when an alarm signal occurs after one second, alarm_cb is called to call do_refresh, then alarm is registered again at the time after one second, and alarm_cb is called to call do_refresh every one minute

②do_refresh
- Called every one minute by alarm signal
- Calls the reread_mapping function and updates the pid mapping information regarding the inode value
- Calls refreshconninode; and updates the tcp access information and the mapping information of the inode value
- Calls the network_concentration function to check whether the searched network process has been optimized for processing

③network_concentration
- Receives information of one network process and information on packets (network usage amount) as a parameter
- Computes the network concentration level by sending the packet information of the received process to the calculation_conc function
- Determines whether optimization has occurred according to the computed concentration level
- Before conducting optimization, the process at a high concentration level calls, according to whether optimization has occurred, process_boost by which optimization is conducted after saving the priority and affinity values before optimization in the process information
- In the case of a low concentration level, returns the value or restores the previous value to return according to whether optimization has occurred

2) Network Connection Level Measurement

①calculation_conc
- Returns the network concentration level on the basis of the packet information (network usage amount) of the network process
3) Core Affinity Manager

①process_restore
- Receives network process information as a parameter
- Reads the priority and process affinity values before optimization that are included in network process information
- Changes the priority (nice) and affinity of the relevant process to the read priority and affinity values
- Changes whether optimization has occurred to false in network information

②process_boost
- Receives the network process information and concentration values as a parameter
- Calculates the affinity value with the concentration value received as a parameter through the priority_surv function
- Calculates the process affinity value with the concentration value received as a parameter through the affinity_surv function
- Sets the calculated priority and affinity values

③priority_surv
- Computes the priority value by receiving the network affinity value as a parameter
- Conducts a masking process on the affinity that is most appropriate for the process by receiving the network affinity value as a parameter

Task Core Feeding (TCF) :
Multi-core based high-performance support subsystem provides a feature to allot tasks in the API form for each core when necessary. A TCF block consists of Process List Management (PLM) for managing and searching information on the currently running process; Thread List Management (TLM) for searching thread information of the process; and Core Affinity Management (CAM) for setting the CPU affinity of the process. And the component block diagram of TCF is shown in Figure 4.

![Figure 4. The Component Block Diagram of TCF](image)

The major functions and features of each block are as follows.

1) Process List Management

①get_pid_list
- Initializes the process queue

②pid_of_name
- Imports the process queue that is running by the process name

2) Thread List Management

①thread_of_pid_get
- Imports the thread running in the process of PID

②get_next_thread
- Imports thread PID from thread queue

③get_next_process
- Imports process information from process queue
3) Core Affinity Management

① set
- Adds the affinity of a specific core to the affinity of process

② set_all
- Sets the affinity of process or thread in all cores

③ set_single
- Sets the affinity of process in only one core

NIC IMPLEMENTATION AND PERFORMANCE TEST

This chapter provides an explanation on Freescale’s development platform P4080PCIe to implement a network interface card by forwarding the high speed feature among virtual machines that were designed in Chapter 3 to the existing platform, along with content descriptions of performance test.

P4080PCIe System :

This chapter provides an explanation on the structure of platform H/W of P4080PCIe, which is necessary to use P4080 Microprocessor and BSP for the implementation of an NIC suggested.

1P4080PCIe Platform :

The specifications of the platform board used are as follows, and the development board configuration is shown in Figure 5.

![Figure 5. P4080PCIe Development Board Configuration](image)

- CPU : Freescale’s QorIQ P4080 multi-core processor. The e500mc cores control Ethernet, console and USB ports.
- CPLD : Altera EPM1270 controls power sequence and MDIO/I2C/SPI interfaces.
- Power : Multi power modules are provided, and they perform various power conversions necessary on the board circuit.
- Netlogic AEL2020 dual 10G PHY ports are provided to support 10G Ethernet ports.

Test Environment Configuration for the Evaluation of Performance:

The improvement measurement environment for multi-core utilization is shown in Figure 6. A specific AP access and a response are measured in Apachebench equipped in the PC that is connected through LAN to a development target. The results of performance measurement are shown in Figure 7. The test has confirmed performance improvement, and a better improvement has been confirmed in the case of mass data.
CONCLUSION

In accommodating the need for the mass storage and virtualization of servers, the demand in the market for network access devices that specialize in virtualization servers, is rapidly increasing. Consequently, high performance network access devices need to be developed that specialize in virtualization servers to support the high speed feature of networking among virtual machines and the feature of the QoS guarantee of services based on the network flow among virtual machines.

With such a background, this study suggested an NIC with an increased throughput due to the feature of a high efficiency.
parallel processing of high speed data by utilizing core affinity in multi-core environments, along with the feature of switching among virtual machines within a host and the feature of a network protocol tunneling engine.

For the implementation of the suggested NIC, performance improvement was achieved through the core affinity utilization in multi-core environments by using Freescale’s P4080PCIe development platform. And features supporting virtualization servers, such as virtual switch and overlay network tunneling engine, were implemented in the NIC development platform, thus supporting virtualization environments, decreasing the load on virtualization servers and supporting the networking among virtual machines from an NIC.

The features above mentioned are expected to be utilized in developing a server support NIC. And further studies should be conducted regarding more enhanced virtualization such as a scheduling in the unit of virtual flow, or a scalable service support.

This work was supported by the research grant of Pai Chai University in 2017.

REFERENCES