Light & NOS

Dan Li
Tsinghua University
The Power of DPDK

- Performance gain
  - As claimed: 80 CPU cycles per packet
  - Significant gain compared with Kernel!

- What we care more…
  - How to leverage the performance gain to serve more applications
  - A great opportunity
    - Decoupling network operation from the kernel / operation system
    - Network can thus develop independently
Improving the performance of Linux network stack

- Too many works…

mTCP

- Based on DPDK
- Realizing network operations as a thread in application

6wind

- Do not know the technical details

What we want

- A DPDK-based network stack that can provide the functionality of network operating system
Light: a Polling-based, General-purpose, User-space, High-performance Network Stack
Light Architecture

- Light Parser
  - Transport layer
  - Network layer
  - Data link layer

- Light Daemon
  - Accept Ready Queue
  - Close Ready Queue
  - TX Ready Queue
  - Recv Ready Queue
  - Command Queue

- Light API

Shared Hugepage Memory

Socket
- RX Queue
- TX Queue

DPDK
Design Goals

- **Goal #1: minimize the modification of applications**
  - Ease the development of new applications
  - Benefit the porting of legacy applications

- **Goal #2: minimize the performance affect to applications**
  - The purpose of DPDK is to increase the I/O performance
  - We do not want that the performance of application is sacrificed due to DPDK
Goal #1: Minimizing the Modification of Apps.

- **Light provides network-related APIs as a lib to apps.**
  - `socket()`, `listen()`, `bind()`, `accept()`, `connect()`, `shutdown()`, `close()`, `socketpair()`
  - `send()`, `receive()`, `sendto()`, `recvfrom()`, `sendmsg()`, `recvmsg()`, `read()`, `write()`, `readv()`, `writev()`
  - `setsockopt()`, `getsockopt()`, `ioctl()`, `fcntl()`
  - `epoll()`, `select()`, `poll()`

- **Challenges**
  - How to mask the same network APIs of the kernel?
  - How to differentiate the two FD spaces (Light and kernel) in the application?

- **Now we need to add several lines of codes in app.**
  - DPDK initialization, DLL management
API Mask

- Applications uses dlsym() to redirect the function address to Light
  - Thus the same network APIs in the kernel are masked
- Do not need to modify the API calls of the application
  - Light’s APIs follow the same format of POSIX APIs
- Do not need to modify the kernel
  - Help the system stability
Two FD Spaces: Problems

- FD confusion
  - Both sockets and files are referred to by FDs
  - E.g., read(), write(), epoll()

- Problems of Epoll
  - Epoll in the application can wait for the events of network sockets, file events, as well as inter-process sockets
    - Epoll for network sockets is supported in Light
    - Epoll for file events is supported by kernel
    - Epoll for inter-process sockets can be either realized in Light or supported by kernel
  - The two Epolls cannot work in blocking mode simultaneously
    - Logic problem
Two FD Spaces: Our Solution

- For the FD confusion problem
  - Light assigns FDs from the upper bound of FD space, because the Kernel assigns FDs from the lower bound of the space

- For the two Epoll problem
  - If we want to detect the events of both FD spaces
    - Intercept Epoll and let it always work in non-blocking mode
    - Cons.: app. cannot be suspended, CPU resource waste
  - If we want to save CPU resource for blocking calls in app.
    - Realize network sockets and inter-process sockets in Light
    - Cons.: Cannot detect file events
Goal #2: Minimizing the Performance Affect to App.

- Challenge 1: DPDK I/O polling in Light occupies 100% CPU resource
  - App. might compete with Light for the CPU resource
- Challenge 2: How to minimize the overhead of inter-process communication
  - Both Light and application are user-space processes
  - The inter-process communication between Light and app. may incur high overhead to apps.
    - Compared with if app. uses kernel network stack
CPU Competition between Light & APPs

- Solution
  - Run Light Parser and Light Daemon in Light cores
  - Run App. and Light API in App. cores
  - Light cores and App. cores are physically separated

- I/O polling only occurs in Light cores, not in App. cores
Inter-process Communication between App. & Light

- **Basic mechanism**
  - Lockless queue based on shared memory
  - RTE-ring in DPDK

- **Blocking API calls in app.**
  - `Epoll()`, `recv()`, `send()`, `accept()`, `connect()`
  - Kernel can suspend the app. process and wake it up after data arrives, which saves the CPU resource
  - Light is a user-space process and cannot do what kernel does
Possible Methods

- **Method 1:**
  - If there is no event in the queue, the (Light API in) app. process goes to sleep
  - When an event comes, Light daemon uses signal to wakeup the process (batch process to further reduce the overhead)
  - Problem: if the last event fails to wake up the app. process
    - Signal can be lost
    - Time sequence error due to process management: an app. process receives the wakeup signal before it goes to sleep

- **Method 2:**
  - Similar way as in hybrid spinlock
  - While() and sleep for some time inside
  - Problem: still add some CPU overhead to app.
Our Solution

- Method 1 for `Epoll()`
- Method 2 for `send()`, `receive()`, `connect()`, `accept()`

**Reasons**

- The `Epoll` queue maintains all the events for all sockets of the process
  - Any kind of event from any socket can wakeup the app. Process
- The queue of the other 4 APIs only maintains a certain kind of events for a certain socket
  - The probability exists that the last event fails to wakeup the app. Process
Features of Light

- Minimal modification of applications
  - Run as a general-purpose service for applications
  - Currently app. only has to add several lines of codes

- Significant performance improvement
  - Inherit the advantage of DPDK
  - Minimize the performance affect to apps. due to DPDK

- Complete protocol stack
  - TCP (including congestion control), UDP, ICMP, IP, UDP, ARP, Ethernet…
Demonstration

- We run Nginx on Light and Linux kernel separately
  - Single process for Nginx
  - Apache benchmark
- Concurrent requests processed on Light more than doubles that in Linux
Thanks!

tolidan@tsinghua.edu.cn