Cooperative network virtualization in the industrial server applications on Linux

Sergey Kovalev, Vasiliy Tolstoy

EMC Corporation RCOE
Saint-Petersburg, Russia
kovals@emc.com, tolstv@emc.com

Abstract

The industrial network server applications the authors encounter in their practice differ from the Linux standard ones. In most cases, the single-process highly optimized application is exposed as a number of virtual servers, and the segregation of traffic becomes a strong requirement. The network needs be configured independently for each virtual server. Usually, confining the application to a container is not possible but some level of cooperation could be ensured instead.

A few prototypes were built, using Linux policy-based routing and Linux kernel namespaces, combined with use of socket options. Tests show good performance, however, open questions still remain. This paper/talk explains the use case, presents the conceptual model, goes over the techniques applied and highlights the networking subsystem limitations encountered.

Keywords

Linux, virtualization, policy based routing, namespaces, industrial server applications, single process, socket options.

Classical vs. Industrial Server Applications

An industrial server box is a computer platform with a server application on it. The platform in a lot of cases is a standard Linux system, but the industrial server application differs from the classical Linux one.

Of course this dichotomy is a generalization; however this model is a good starting point for the discussion. Let us sum up the difference between two kind of server applications.

Classical Server Application

A classical Linux server application usually has the following characteristics:
•it is one process, possibly spawning
•serves one physical link (possibly aggregated)
•uses one target IP address
•different configuration requires to start a copy of base process (not always)
•uses standard system auxiliary services like name resolving, time, etc.

Industrial Server Application

The industrial server application differs from the standard one by the following:
•it is one multi-thread process, non-spawning (usually)
•serves multiple physical links
•serves multiple VLANs and IP subnets
•is exposed as multiple virtual servers (VS)
•uses multiple target IP addresses for each VS
•separate auxiliary services independently configured for each VS

Additional Features

The industrial server application can have some additional specific features. The authors would like to stress out the following ones:
•traffic reflection ability: to reply by the same device and route you got the request
•outgoing connections limited to the virtual server scope
•maximum reuse of the existing Linux stack (for the ease of support)
•fast startup and fast network configuration
•maximum use of hardware offload, TCP and iSCSI

Single-process Application

The industrial server applications tend to be single-process. The reasons for that are the following:
•low latency: no context switch, no cache invalidation, zero copy is possible
•high performance: the same as above
•deduplication: on the storage systems, the wider the deduplication scope the better

Cooperative Virtualization

Industrial server application requires some kind of Linux net stack slicing. It is important that the application is aware of it and could be designed with this requirement in mind. We can call it “cooperative virtualization”.

So what means do we have in Linux to support this kind of virtualization?
•virtual machines
•lightweight virtualization/containers (LXC/Dockers)
•Linux namespaces “by hand”
•policy based routing
•firewall configuration
•clever application design

Virtual machines and containers do not fit the requirements: they imply separate processes. Clever application design alone is limited by the socket API: not much control is available. The firewall configuration has its potential, but is limited by absence of the documented netfilter API.

In practice, the designer chooses from two other options: policy based routing (PBR) and Linux namespaces (NS).

Policy Based Routing

An application may create a number of sockets bound to a specific IP addresses and listen on them. We can write rules with the “src IP” criteria and have a per-IP routing tables then. It covers the “one IP per VS” use case, but is not enough for “set of IPs per VS”.

It would be good if the virtual server can tag its sockets by some ID and the PBR can use this tag as a rule criteria class. Fortunately, there is one: SO_MARK socket option is actually the PBR “fwmark”. The routing table system then gets trivial (see Table1).

Rules:
`...`
`30010: from 10.0.0.5 lookup` **`10`**
`30011: from 10.0.1.5 lookup` **`11`**
`30012: from all fwmark 0xc lookup` **`12`**
`...`
Table **10:**
`10.0.0.0/24 dev eth0 src 10.0.0.5`
`default via 10.0.0.1 dev eth0 src 10.0.0.5`
Table **11:**
`10.0.1.0/24 dev eth0 src 10.0.1.5`
`default via 10.0.1.1 dev eth0 src 10.0.1.5`
Table **12:**
`10.0.0.0/24 dev eth0 src 10.0.0.5`
`10.0.1.0/24 dev eth0 src 10.0.1.5`
`default via 10.0.0.1 dev eth0 src 10.0.0.5`

Table 1. PBR route tables system for a virtual server with two IPs.

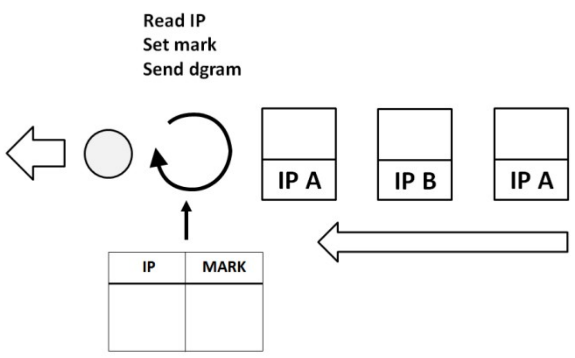
The application then needs to carefully tweak the number of routing tables when the IP address set of a virtual server changes.

UDP Problem

A few practical issues pop up when using the PBR approach. Most notable is the UDP problem. While for each TCP connections the kernel spawns off a new socket, the UDP “pseudo-connections” for a multiple virtual servers are usually done via the single socket bound to some well-known port. It is impossible to clone such a socket, so one cannot mark a socket once and use it for a certain connection.

To overcome this, a few approaches could be used. The most radical one is the dynamic change of socket mark on the per-UDP-packet basis (see Fig. 1). The application gets complicated and gets the performance issues.

Figure 1. Dynamic marking of a UDP socket.



Some Linux mechanism to split the incoming traffic between UDP sockets with different marks may help here.

Our Results with PBR

We have tested this approach on a system with an application exposed as a number of virtual NAS servers sharing the common backend. CIFS and NFS file access by TCP and UDP transports were supported. The system is based on the vanilla Linux kernel, and the application uses the custom name resolving library with different configuration for each VS.

The tests show that the performance is not affected except for the UDP transport, but the overall effect on performance is low because of the very limited use UDP has.

Network Namespaces

With the PBR, all the VSes share the same IP addres plan and firewall configuration. The namespaces allow us to have use the independent IP address plans and firewall configurations for group of VSes.

A thread can independently switch to a different namespace, open a socket and work with it. More than that, we have practically tested that a thread can open a socket in the namespace A, switch to the namespace B and open a socket there, then switch to the namespace C and still work with both A and B sockets. Therefore, an application may serve different network namespaces at the same time.

Any network device can be included in a namespace, including the virtual devices of (almost) any kind. It means that if you can separate the traffic between two devices, you can direct it to different namespaces.

The simple practical example we used is presented on Figure 2.

Two mod8021q devices, one for VLAN 100 and another for VLAN 200 were created on the eth0 NIC device. They were included in the namespaces A and B correspondingly. Each VLAN corresponds to the IP subnet with the same 10.1.1.0/24 IP addresses. The test application still could serve them both simultaneously.

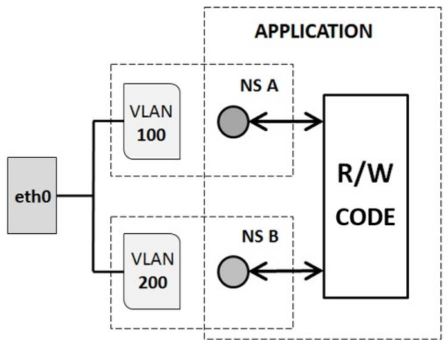


Figure 2. Basic multi-namespace application.

Problems with the Namespaces

The additional technologies and auxiliary services need to be sliced too:

- RPC port mapping (rpcbind)
- system time (NTP)
- TCP and iSCSI offload

Patching the rpcbind seems simple. However, the standard Linux rpcbind does not support multiple network namespaces out of the box, so the one who patches it is bound to support it then.

Running the rpcbind in a full-fledged container may solve the problem but make the system configuration even more complicated.

For the moment, independent time for different virtual servers is an open question.

We expect our further investigations to find us more problems of this kind.

Our Results with the Namespaces

We have created a prototype system with a multi-namespace test application exposed as a number of virtual servers in different namespaces.

To test the performance effect we have created a 100-namespaces configuration using 100 VLAN IDs both on the server and client VMs (see Fig. 3). On the server side, a single application accepts the TCP connections. On the client side, 100 different client processes establish the connections to the server, and send/receive some data in a cycle.

The performance impact of the namespaces was found to be negligible. The memory consumption increase is about 120 kB/NS at the net object creation time and does not change with the I/O load (see Fig. 4).

The system uses the vanilla Linux kernel. One of our goals was not to modify the kernel, and it seems that it is possible.

Our experiments with the other namespaces problem solutions are on the very early stage at the moment, and are not ready to be presented.

Figure 3. Scalability/performance test setup.

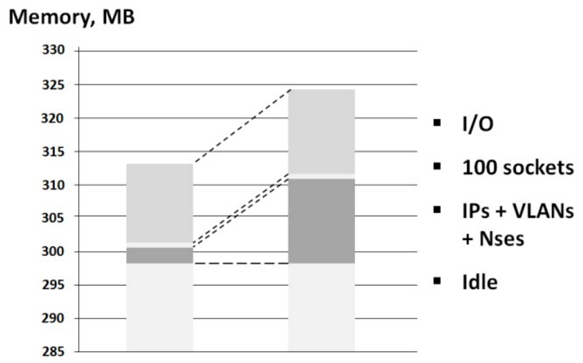


Figure 4. Scalability/performance test results.

Conclusion

Together with the reasonable application modifications, current Linux networking stack features allow the cooperative virtualization sufficient for the industrial server applications.

However, there still is room for improvement. Most notably, the PBR UDP problem, and the independent per-namespace system auxiliary services are the open questions.

Bibliography

Rami Rosen, *Linux Kernel Networking: Implementation and Theory* (Apress, 2013).
Christian Benvenuti, *Understanding Linux Network Internals* (O'Reilly Media, 2005).
Klaus Wehrle, Frank Pahlke, Hartmut Ritter, Daniel Muller and Marc Bechler, *Linux Network Architecture* (Prentice Hall, 2005)
Robert Love, *Linux Kernel Development, 3rd Edition* (Addison-Wesley Professional, 2010)
ip man page online, accessed February 10, 2015, http://man7.org/linux/man-pages/man8/ip.8.html

Authors Biographies

Sergey Kovalev is a Senior Software Engineer at EMC Corporation in Russia COE. He has worked for EMC since 2010. As a senior software engineer Sergey is responsible for the development of network management software for enterprise storage systems. His key interests are OS design, low-level programming and Linux networking. Sergey has a BS in Computer Sciences from the State Saint-Petersburg Polytechnic University.

Vasiliy Tolstoy is a Principal Software Engineer at EMC Corporation in Russia COE. He has worked for EMC since 2010. His current professional interests include Linux networking, virtualization, and SDN/NFV. He is also interested in UI design, OS design and hardware-level programming. He has 26 years of combined experience in software design, development, and system administration. He studied physics at the State Saint-Petersburg University.