Intel® Dynamic Load Balancer (Intel® DLB) Software

User Guide

Revision v8.11.0

Sept 2024

Table of Contents

1 Introduction 5

2 Build Instructions 7

2.1 Package Contents 7
2.2 Intel® DLB Linux Driver Build 7
2.3 Intel® DPDK DLB PMD Build 8
2.4 Libdlb build 8

3 Sample Applications 9

3.1 Libdlb Sample Apps 9
3.2 DPDK Sample Apps 9
3.3 DLB Monitor 11
3.4 DLB Debug Utility 12

4 Live Migration 13

4.1 Framework 13
4.2 SIOV Live Migration 13
4.3 SRIOV Live Migration 15
4.4 QEMU for Live Migration 17

5 FAQs 18

5.1 VFIO-PCI Bind Issue 18

Tables

Table 1. Terminology 5

Revision History

Date	Revision	Description
Sept 2024	v8.11.0	Bug fixes, support for kernel 6.10 and enhancements.
June 2024	v8.10.0	Bug fixes, live migration SRIOV support and enhancements.
May 2024	v8.9.0	Bug fixes, container support without virtualization.
March 2024	v8.8.0	Bug fixes, enqueue time reordering and enhancements.
Feb 2024	v8.7.2	Driver version correction of previous release.
Jan 2024	v8.7.1	Bug fixes, improvements, add support for RHEL 9.4 and kernel 6.7.
Dec 2023	v8.7.0	Bug fixes, improvements for intel-next kernel 6.6 support.
Nov 2023	v8.6.0	Bug fixes, support for intel-next kernel 6.6 and improvements.
October 2023	v8.5.2	Bug fixes and improvements.
August 2023	v8.5.1	Mailbox bug fixes and improvements.
August 2023	v8.5.0	Delayed Pop Token Support for PF-PMD, Dynamic HL Support + Bug Fixes.
May 2023	v8.4.0	DLB 2.x OpenEuler Support, Dynamic and Fixed HL Sizes.
April 2023	v8.3.0	DLB 2.x BKC Kernel 6.2 support, HW Delay Token support + Bug fixes.
March 2023	v8.2.0	DLB 2.x VM Live Migration add Ordered traffic support + Bug fixes.
Feb 2023	v8.1.0	DLB 2.x VM Live Migration add VM/state save and restore
Dec 2022	v8.0.0	DLB 2.x VM Live Migration framework and SIOV 5.19 intel-next kernel support
Nov 2022	v7.8.1	DLB 2.0 Production Candidate (PC) Plus Release
Sept 2022	v7.8.0	DLB 2.0 Production Candidate (PC) Plus Release
June 2022	v7.7.0	DLB 2.0 Production Candidate (PC) Release
Mar 2022	v7.6.0	DLB 2.0 Beta3 Release
Jan 2022	v7.5.0	DLB 2.0 Beta2 Release
Oct 2021	v7.4.0	DLB 2.0 Beta Release
July 2021	v7.3.0	Vector support and interrupt support added
May 2021	v7.2.0	Updated release with SIOV & DLB Monitor features enabled
April 2021	v7.0.0	Initial release to 01.org

1 Introduction

The Intel® Dynamic Load Balancer (Intel® DLB) is a PCIe device that provides load-balanced, prioritized scheduling of events (that is, packets) across CPU cores enabling efficient core-to-core communication as discussed in this White Paper. It is a hardware accelerator located inside the latest Intel® Xeon® devices offered by Intel. It supports the event-driven programming model of DPDK's Event Device Library. This library is used in packet processing pipelines for multi-core scalability, dynamic load-balancing, and variety of packet distribution and synchronization schemes. The DLB can also be used without DPDK as discussed later in this document.

This document describes the steps involved in building the DLB Kernel Driver, DPDK DLB Poll Mode Driver and running the sample applications. It also introduces libdlb, a client library for building DLB applications without using DPDK framework. This release package supports Intel Dynamic Load Balancer 2.0 and 2.5.

Disclaimer:

This code is being provided to potential customers of DLB to enable the use of DLB well ahead of the kernel.org and DPDK.org upstreaming process. Based on the open source community feedback, the design of the code module can change in the future, including API interface definitions. If the open source implementation differs from what is presented in this release, Intel reserves the right to update the implementation to align with the open source version at a later time and stop supporting this early enablement version.

Table 1. Terminology

Term	Description
DLB	Intel Dynamic Load Balancer 2.x
DPDK	Data Plane Development Kit
mdev	Mediated Device
PMD	Poll Mode Driver
Bifurcated PMD	A DPDK PMD in which device configuration and management is handled in a kernel driver.
PF PMD	A DPDK PMD that takes ownership of the device PF function, its configuration and management.
PCIe	Peripheral Component Interconnect Express
PF	PCIe device Physical Function
QE	Queue Element – 16 Bytes data unit used by the DLB hardware to enqueue events in CQ

Term	Description
SRIOV	Single Root Input/Output Virtualization
SIOV	Scalable Input/Output Virtualization
VAS	Virtual Address Space. VAS is synonymous with scheduling domain in the case of DLB.
VFIO	Virtual Function Input/Output
VM	Virtual Machine

2 Build Instructions

2.1 Package Contents

This software release is provided as a gzip archive that can be unzipped to install the provided files and documentation. The archive is designed to provide three major components:

1. DLB Kernel Driver Source Code
2. Files needed to patch DPDK to enable both PF (Physical Function) and bifurcated modes.
3. The libdlb files that can be leveraged to use DLB without using DPDK.

For supported kernels, refer to README. When unzipped, the included files/directories will be structured as follows:

```
dlb
 |
 |-driver
     |
     |-dlb2
 |
 |-dpdk
    |
    |-dpdk_dlb_xxx.patch
 |
 |-libdlb
 |
 |-docs
```

2.2 Intel® DLB Linux Driver Build

DLB driver uses the kbuild build system. To build out-of-tree, simply run 'make'. You can optionally provide the KSRC environment variable to specify the kernel source tree to build against. (If unspecified, build uses the host OS's kernel headers.)

```
$ cd $DLB_SRC_TOP
```

(top-level directory of the extracted DLB source package tarball)

```
$ cd dlb/driver/dlb2
```

```
$ make
```

2.3 Intel® DPDK DLB PMD Build

The release package contains an addon patch to the DPDK base package for DLB PMD support. DPDK base packages are available for download at www.dpdk.org; be sure to download the version specified in the DPDK Version section of the Readme. More details on DLB PMD can be found at https://doc.dpdk.org/guides/eventdevs/dlb2.html

To apply the addon patch and build DPDK, follow these steps:

```
$ cd $DLB_SRC_TOP

$ wget <URL for DPDK base package>

$ tar xfJ dpdk-<DPDK version>.tar.xz
```

$ cd dpdk-<DPDK version> (This is the $DPDK_DIR path used in steps below)

```
$ patch -Np1 < $DLB_SRC_TOP/dlb/dpdk/<DPDK patch name>.patch
```

For meson-ninja build:

```
$ export DPDK_DIR=path to dpdk-<DPDK version>

$ export RTE_SDK=$DPDK_DIR

$ export RTE_TARGET=installdir

$ meson setup --prefix $RTE_SDK/$RTE_TARGET builddir

$ ninja -C builddir install
```

Additional meson configuration and build options can be found on dpdk.org.

2.4 Libdlb build

To build libdlb:

```
$ cd $DLB_SRC_TOP/dlb/libdlb

$ make
```

3 Sample Applications

3.1 Libdlb Sample Apps

The Libdlb library provides directed and load-balanced traffic tests to demonstrate features supported by the Intel DLB. The sample codes are located at libdlb/examples and are built together with libdlb. Details of supported APIs can be found in the software release under docs/libdlb_html. Documentation is accessible from index.html that can be opened using any browser. To load the dlb kernel driver and run the sample applications, follow these steps:

```
$ modprobe mdev

$ modprobe vfio_mdev

$ cd $DLB_SRC_TOP/dlb

$ insmod driver/dlb2/dlb2.ko

$ cd libdlb
```

For Directed Traffic test:

```
$ LD_LIBRARY_PATH=$PWD ./examples/dir_traffic -n 128
```

For Load Balanced Traffic test:

```
$ LD_LIBRARY_PATH=$PWD ./examples/ldb_traffic -n 128
```

(n: number of events to be sent)

The default wait mode supported is the interrupt mode. "-w" option can be used to change to poll or epoll mode. Use "-h" to display other command line options for the sample applications.

3.2 DPDK Sample Apps

DPDK contains dpdk-test-eventdev, a standalone application to demonstrate eventdev capabilities. Following are the steps to run its order_queue test with DLB. DLB eventdev supports two modes of operation, Bifurcated and the PF PMD modes. Either of the two can be used to run the sample app:

1) Load the drivers needed for DPDK applications:

- For Bifurcated mode, load the DLB Kernel Driver:

  ```
  $ insmod $DLB_SRC_TOP/dlb/driver/dlb2/dlb2.ko
  ```

- For PF PMD mode, bind DLB to either vfio-pci or uio_pci_generic driver.

  For vfio-pci driver:

```
$ modprobe vfio

$ modprobe vfio-pci

$ lspci -d :2710      #To check DLB2.0 device id (For ex: eb:00.0)

$ lspci -d :2714      # To check DLB2.5 device id (For ex: 14:00.0)

$ $DPDK_DIR/usertools/dpdk-devbind.py --bind vfio-pci \
   eb:00.0
```

For uio_pci_generic driver:

```
$ modprobe uio_pci_generic

$ $DPDK_DIR/usertools/dpdk-devbind.py \

  --bind uio_pci_generic eb:00.0
```

For bifurcated mode bind to dlb driver:

```
$ $DPDK_DIR/usertools/dpdk-devbind.py \

  --bind dlb2 eb:00.0
```

2) Setup Hugepages:

Check if the hugepage requirements are met using the following command:

```
  $ cat /proc/meminfo | grep Huge
```

If no hugepages are available, setup 2048 count of 2048kB sized hugepages as below (sudo required) :

```
$ mkdir -p /mnt/hugepages

$ mount -t hugetlbfs nodev /mnt/hugepages

$ echo 2048 > /sys/kernel/mm/hugepages/hugepages-
2048kB/nr_hugepages
```

3) Run the application

```
$ cd $DPDK_DIR   (DPDK base directory)

$ cd builddir/app
```

- To run the application in PF PMD Mode

```
$ ./dpdk-test-eventdev -c 0xf  -- --test=order_queue \
   --plcores=1 --wlcore=2,3 --nb_flows=64
```

- To run the application in Bifurcated mode, --vdev=dlb2_event needs to be passed and dlb device needs to be bound to DLB Driver.

```
$ ./dpdk-test-eventdev -c 0xf --vdev=dlb2_event \
   -- --test=order_queue --plcores=1 --wlcore=2,3 \
   --nb_flows=64
```

dpdk-test-eventdev's **perf-queue** test can be used to explore the different queue types supported by the Intel DLB. The –-stlist command-line option allows configuring the number of stages and scheduling types. 'p' for parallel, 'a' for atomic and 'o' for ordered queue types. `--prod_enq_burst_sz` option can be used for producer core to enqueue burst of events.

This test can also be run in both PF and Bifurcated PMD modes. Follow Steps 1 and 2 from above to load required drivers and setup hugepages, if not already done.

To run the application in PF PMD mode:

```
./dpdk-test-eventdev -c 0xf  -- --test=perf_queue --plcores=1 \
--wlcore=2,3 --nb_flows=64 --stlist=p --prod_enq_burst_sz=64
```

(DPDK uses DLB devices bound to vfio-pci or uio_pci_genric drivers)

For Bifurcated PMD mode:

```
 ./dpdk-test-eventdev -c 0xf --vdev=dlb2_event -- \
--test=perf_queue --plcores=1 --wlcore=2,3 --nb_flows=64 \
--stlist=p --prod_enq_burst_sz=64
```

```
(To make sure DPDK does not use DLB devices bound to vfio-pci or
uio_pci_generic modules, use –no-pci option or unbound them. Vdev
dlb2_event uses DLB devices bound to DLB driver dlb2.)
```

More details of different testcases supported by dpdk-test-eventdev app and command-line options can be found in

https://doc.dpdk.org/guides-21.11/tools/testeventdev.html

3.3 DLB Monitor

The DPDK patch provides dlb_monitor application, a telemetry tool that collects and displays DLB hardware and software configuration and statistics. The tool also identifies and reports common misconfiguration issues and potential application performance issues. When any dpdk application is run, dlb_monitor can be triggered as a secondary process to monitor eventdev port, queue, device status. This tool cannot be started independently without a primary process running. The stats can be printed once, or the application can run monitor in 'watch mode' (-w option) and the data is repeatedly collected and displayed at a user-specified frequency.

```
$ cd builddir/app
```

- To run dlb_monitor in PF PMD Mode :

```
$ ./dpdk-dlb_monitor  -- -w
```

- To run in Bifurcated PMD Mode :

```
$ ./dpdk-dlb_monitor --vdev dlb2_event  -- -w
```

3.4 DLB Debug Utility

The DLB tar file provides dlb_monitor_sec application, a tool that collects and displays DLB hardware register configuration and statistics for the libdlb applications. It also works with DPDK applications in Bifurcated mode. When any DLB application is running, *dlb_monitor_sec* can be triggered as another process to monitor ports, queues and device status.

The statistics can be printed once, or the application can run this utility in 'watch mode' (-w option) and the data is repeatedly collected and displayed at a user-specified frequency.

• To run dlb_monitor_sec:

```
$ cd dlb/libdlb/cli

$ ./dlb_monitor_sec -h
```

```
Usage: dlb_monitor_sec [options]
Options:
-i <dev_id>   DLB Device id (default: 0)
-r            Reset stats after displaying them
-t <duration> Measurement duration (seconds) (min: 1s, default: 1s)
-w            Repeatedly print stats
-z            Don't print ports or queues with 0 enqueue/dequeue/depth stats
-l            Print LDB queue statistics
-d            Print DIR queue statistics
-c            Print CQ  queue statistics
-a            Equivalent to setting 'ldcg' flags
-o            Generate CSV output file, (generates header.csv,output_raw.csv prefixed
with dlb<devid>)
```

4 Live Migration

4.1 Framework

DLB VFIO Live Migration interface has been added starting from release v8.0.0 for SIOV and 8.10.0 for SRIOV. All steps of the following live migration stages have been implemented:

1. Initiate Live Migration via VFIO interface
    a. VFIO Live Migration state machine
    b. Add VFIO_DEVICE_STATE_SAVING, _RUNNING, _STOP, _RESUMING
    c. Communication between source VM and destination VM
2. Save and restore VAS/Domain configuration
    a. VAS/Domain resource setup
        i. Port configuration
        ii. Queue configuration
        iii. Port-queue links
        iv. Sequence Number setup
    b. Virtual Port/Queue --> Physical P/Q mapping
3. Save and restore QEs in DLB (Needed only in case there is a traffic through DLB at the time of migration)
    a. History list entries
    b. Reorder buffers
    c. QE's in the Queues
4. Resume Application on the Destination device
    a. VFIO_DEVICE_STATE_RESUMING.

For Live Migration to work, two identical VM (guest) instances are needed with the same configuration. This allows saving the state of a guest VM and restoring it on another VM instance, thereby allowing an application instance to continue running while the state is transferred. Please note that 8 extra history list entries will be reserved for live migration per VM. Users should take this into account when allocating history list entries for a VM.

4.2 SIOV Live Migration

To test Live Migration for SIOV:

1. Load the DLB driver on the host.
2. Create an mdev (mdev-1) on dlb device 0:

```
export SYSFS_PATH=/sys/class/dlb2/dlb0/
export UUID1=`uuidgen`
export MDEV_PATH=/sys/bus/mdev/devices/$UUID1/dlb2_mdev/
echo $UUID1 >
$SYSFS_PATH/device/mdev_supported_types/dlb2-dlb/create
# Assign it all of the PF's resources
echo 2048 > $MDEV_PATH/num_atomic_inflights
echo 4096 > $MDEV_PATH/num_dir_credits
echo 64 > $MDEV_PATH/num_dir_ports
```

```
echo 2048 > $MDEV_PATH/num_hist_list_entries
echo 8192 > $MDEV_PATH/num_ldb_credits
echo 64 > $MDEV_PATH/num_ldb_ports
echo 32 > $MDEV_PATH/num_ldb_queues
echo 32 > $MDEV_PATH/num_sched_domains
echo 8 > $MDEV_PATH/num_sn0_slots
echo 8 > $MDEV_PATH/num_sn1_slots
```

3. Start the source VM with Qemu, and pass mdev-1 to the source VM. A sample Qemu command is provided below. The highlighted '**file**' option requires a VM image path:

```
/usr/libexec/qemu-kvm -enable-kvm -global kvm-
apic.vapic=false -m 4096 -cpu host -drive
format=raw,file=<file-name> -bios
/usr/share/qemu/OVMF.fd -device vfio-
pci,sysfsdev=/sys/bus/mdev/devices/$UUID1,x-enable-
migration=true -smp 8 -netdev
user,id=n1,hostfwd=tcp::2222-:22 -fsdev
local,security_model=none,id=fsdev0,path=/home/ -device
virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare -
nographic -serial stdio -monitor
telnet::2205,server,nowait -name debug-threads=on
```

4. From a different terminal, create another mdev (mdev-2) on dlb device 1:

```
export SYSFS_PATH=/sys/class/dlb2/dlb1/
export UUID2=`uuidgen`
export MDEV_PATH=/sys/bus/mdev/devices/$UUID2/dlb2_mdev/
### Create the mdev
echo $UUID2 >
$SYSFS_PATH/device/mdev_supported_types/dlb2-dlb/create
# Assign it all of the PF's resources
echo 2048 > $MDEV_PATH/num_atomic_inflights
echo 4096 > $MDEV_PATH/num_dir_credits
echo 64 > $MDEV_PATH/num_dir_ports
echo 2048 > $MDEV_PATH/num_hist_list_entries
echo 8192 > $MDEV_PATH/num_ldb_credits
echo 64 > $MDEV_PATH/num_ldb_ports
echo 32 > $MDEV_PATH/num_ldb_queues
echo 32 > $MDEV_PATH/num_sched_domains
echo 8 > $MDEV_PATH/num_sn0_slots
echo 8 > $MDEV_PATH/num_sn1_slots
```

5. Run the Qemu command for Destination VM and pass mdev-2 to it. This will not start the VM until the migration has been initiated.

```
/usr/libexec/qemu-kvm -enable-kvm -global kvm-
apic.vapic=false -m 4096 -cpu host -incoming tcp:0:6666
-drive format=raw,file=<file-name>  -bios
/usr/share/qemu/OVMF.fd -device vfio-
pci,sysfsdev=/sys/bus/mdev/devices/$UUID2,x-enable-
migration=true -smp 8 -netdev
user,id=n2,hostfwd=tcp::2203-:22 -fsdev
local,security_model=none,id=fsdev0,path=/home/ -device
virtio-9p-pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare -
```

```
nographic -serial stdio -monitor
telnet::2206,server,nowait -name debug-threads=on
```

6. On the source VM, load the dlb driver and run ldb_traffic sample as follows:
   ```
   ./ldb_traffic -n 1024
   ```

7. Trigger the migration using Qemu console of the source VM (to be triggered immediately after issuing the ldb_traffic command on the source VM) :
   ```
   telnet localhost 2205
   (qemu) migrate -d tcp:0:6666
   (qemu) info migrate
   ```

8. After the migration is completed, ldb_traffic is expected to continue and complete on the destination VM.

Currently Live Migration is supported only on intel-next kernel v5.15.

4.3 SRIOV Live Migration

Live Migration for SRIOV is supported only on intel-next kernel v6.2 and later. The Makefile will now generate TWO kernel drivers, an original DLB kernel driver **dlb2.ko** and dlb vfio pci driver **dlb2-vfio-pci.ko**, which is a vfio-pci driver for dlb2 with live migration support. Please use the following sequence to load the drivers.

a. insmod dlb2.ko
b. modprobe vfio-pci-core
c. insmod dlb2-vfio-pci.ko

There are different steps explained below for Live Migration to work for SRIOV.

1. Create a VF dlb device 0 and bind it to dlb2-vfio-pci,

   ```
   echo 1 > /sys/class/dlb2/dlb0/device/sriov_numvfs
   echo 0000:ea:00.1 > /sys/bus/pci/drivers/dlb2/unbind
   echo 8086 2711 > /sys/bus/pci/drivers/dlb2-vfio-pci/new_id

   # Assign it 1/2 of the PF's resources
   VF0_RSCS = /sys/bus/pci/devices/0000:ea:00.0/vf0_resources
   echo 1024 > $(VF0_RSCS)/num_atomic_inflights
   echo 1024 > $(VF0_RSCS)/num_dir_credits
   echo 16 > $(VF0_RSCS)/num_dir_ports
   echo 1024 > $(VF0_RSCS)/num_hist_list_entries
   echo 4096 > $(VF0_RSCS)/num_ldb_credits
   echo 16 > $(VF0_RSCS)/num_ldb_ports
   echo 16 > $(VF0_RSCS)/num_ldb_queues
   echo 16 > $(VF0_RSCS)/num_sched_domains
   echo 8 > $(VF0_RSCS)/num_sn0_slots
   echo 8 > $(VF0_RSCS)/num_sn1_slots
   ```

2. Start the source VM with Qemu, and pass vf0 from dlb0 to the source VM. A sample Qemu command is provided below. The highlighted ‘file’ option requires a VM image path:

```
qemu-system-x86_64 -enable-kvm -global kvm-apic.vapic=false \
-m 4096 -cpu host -drive format=raw,file=<file-name> \
-bios /usr/share/qemu/OVMF.fd  \
-object iommufd,id=iommufd0 \
-device vfio-pci,host=ea:00.01,iommufd=iommufd0,enable-
migration=on\
-smp 8 -netdev user,id=n1,hostfwd=tcp::2222-:22 -fsdev
local,security_model=none,id=fsdev0,path=/home/ -device virtio-9p-
pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare -nographic -serial
stdio -monitor telnet::2205,server,nowait -name debug-threads=on
```

3. From a different terminal, create another VF on dlb device 1:

```
echo 1 > /sys/class/dlb2/dlb0/device/sriov_numvfs
echo 0000:ef:00.1 > /sys/bus/pci/drivers/dlb2/unbind
echo 8086 2711 > /sys/bus/pci/drivers/dlb2-vfio-pci/new_id

# Assign it 1/2 of the PF's resources
VF0_RSCS = /sys/bus/pci/devices/0000:ef:00.0/vf0_resources
echo 1024 > $(VF0_RSCS)/num_atomic_inflights
echo 1024 > $(VF0_RSCS)/num_dir_credits
echo 16 > $(VF0_RSCS)/num_dir_ports
echo 1024 > $(VF0_RSCS)/num_hist_list_entries
echo 4096 > $(VF0_RSCS)/num_ldb_credits
echo 16 > $(VF0_RSCS)/num_ldb_ports
echo 16 > $(VF0_RSCS)/num_ldb_queues
echo 16 > $(VF0_RSCS)/num_sched_domains
echo 8 > $(VF0_RSCS)/num_sn0_slots
echo 8 > $(VF0_RSCS)/num_sn1_slots
```

4. Start the source VM with Qemu, and pass vf0 from dlb1 to the source VM. A sample Qemu command is provided below. The highlighted ‘file’ option requires a VM image path:

```
qemu-system-x86_64 -enable-kvm -global kvm-apic.vapic=false \
-m 4096 -cpu host -drive format=raw,file=<file-name> \
-bios /usr/share/qemu/OVMF.fd  \
-object iommufd,id=iommufd0 \
-device vfio-pci,host=ef:00.01,iommufd=iommufd0,enable-
migration=on\
-smp 8 -netdev user,id=n2,hostfwd=tcp::2203-:22 -fsdev
local,security_model=none,id=fsdev0,path=/home/ -device virtio-9p-
pci,id=fs0,fsdev=fsdev0,mount_tag=hostshare -nographic -serial
stdio -monitor telnet::2206,server,nowait -name debug-threads=on
```

5. On the source VM, load the dlb driver and run ldb_traffic sample as follows:
   `./ldb_traffic -n 1024`

6. Trigger the migration using Qemu console of the source VM (to be triggered immediately after issuing the ldb_traffic command on the source VM) :
   ```
   telnet localhost 2205
   (qemu) migrate -d tcp:0:6666
   (qemu) info migrate
   ```

7. After the migration is completed, ldb_traffic is expected to continue and complete on the destination VM.

4.4 QEMU for Live Migration

Qemu from Intel BKC 5.15 can be used to test SIOV live migration.

To test live migration with qemu from BKC 6.2 for either SRIOV or SIOV, please use the following steps to clone and change qemu (otherwise LM will not work due to DMA memory transfer issue).

a. Clone qemu repo: https://github.com/intel-innersource/virtualization.hypervisors.server.vmm.qemu-bkc
b. For EMR BKC 6.2, get branch emr-Q7.2-K6.2
c. Open the file migration/ram.c of the qemu repo, and add ram_list_init_bitmaps() in the beginning of ram_save_complete() as follows,

```
static int ram_save_complete(QEMUFile *f, void *opaque)

{

RAMState **temp = opaque;

RAMState *rs = *temp;

int ret = 0;

ram_list_init_bitmaps();     /* added this for DLB LM WA */

rs->last_stage = !migration_in_colo_state();

WITH_RCU_READ_LOCK_GUARD()

{ if (!migration_in_postcopy()) }
```

d. Save the file and recompile the qemu. mkdir build;cd build;../configure --target-list=x86_64-softmmu; make -j 32;

To test live migration with qemu from latest BKC 6.6 for either SRIOV or SIOV, please note that qemu option "x-enable-migration=true" is changed to "enable-migration=on". Also a new option x-force-all-dirty=on is added for "--device vifo-pci" as follows,

```
-device vfio-pci,host=ea:00.01,iommufd=iommufd0,enable-migration=on,x-force-all-dirty=on \
```

The standard qemu from stable-9.0 branch of github.com/qemu can also be used to test SRIOV live migration on kernel 6.6.

5 FAQs

5.1 VFIO-PCI Bind Issue

Error: “cannot bind to driver vfio-pci”

Fixes:

1. Enable VT-D in BIOS
2. Enable SRIOV in BIOS
3. Make sure the following is in /proc/cmdline (GRUB_CMDLINE_LINUX in /etc/default/grub): iommu=pt intel_iommu=on