Voltaire®
Messaging Accelerator (VMA)
Library for Linux

User Manual

July 2008
P/N: DOC-00393
Rev. A00

Business Headquarters
Voltaire Inc.
6 Fortune Drive, Suite 301
Billerica, MA
USA 01821
Tel: 978-439-5400
Fax: 978-439-5401

Israel Office
Voltaire Ltd.
9 Hamenofim St.
Bldg. A Herzeliya
46725, Israel
Tel: +972 (9) 971-7666
Fax: +972 (9) 971-7660

Document Part Number: P/N DOC-00393

Amendment Log
(Remove in PDF)

Date	Rev	Description of Change	Author	Requested by
20 Mar 08	A01	Name change from MCE to VMA. Added detail.	AvrahamM	Alex Rosenbaum
28 Apr 08	-	Added Confidential Watermark	Sylvia	Amir Presher

Contents

About this Manual ..... **v**
Audience vi
Document Conventions vi
Document Organization vii
Related Documentation vii

Chapter 1. Introduction **1-1**
1.1 VMA Overview 1-2
1.2 Basic Features 1-2
1.3 Target Applications 1-3
1.4 Advanced VMA Features 1-3
1.5 Extending InfiniBand Performance to GEth Networks 1-4

Chapter 2. VMA Library Architecture **2-1**
2.1 Top-level 2-2
2.2 Unicast Support 2-4

Chapter 3. Installing and Running VMA **3-1**
3.1 Prerequisites 3-2
3.2 Installation 3-2
3.3 Uninstalling 3-2
3.4 Upgrading 3-3
3.5 Query Install Package Information 3-3
3.6 Running 3-3

Chapter 4. Configuring VMA **4-1**
4.1 Configuration Parameters 4-2
4.2 Configuration Values 4-3

Chapter 5. Debugging, Troubleshooting and Monitoring **5-1**
5.1 Monitoring - vma_stats Utility 5-2
5.2 Debugging 5-3
5.3 Troubleshooting 5-3

Appendix A UDP Latency Tool: udp_lat **A-1**
A.1 Overview A-2
A.2 Running the Test A-2

A.3 Debugging for udp_lat A-5
A.4 Troubleshooting for udp_lat A-5
Appendix B Multicast Routing B-1
Appendix C Acronyms C-1

Figures

Figure 2-1. VMA Library System Architecture 2-3

Tables

Table 4-1. Configuration Values 4-3

About this Manual

This preface describes the objectives, audience and conventions of this manual. It also provides information on related documentation and obtaining technical assistance.

NOTE

Refer to the Voltaire release notes for last minute updates and restrictions.

NOTE

The Voltaire Technical Support Center (VSC) is at your service. You may access Warranty Service through our Web Request Form by using the following link:

http://www.voltaire.com/support.htm

Contact Us:

Please send your documentation-related comments and feedback or report mistakes to docs@Voltaire.com.

We are committed to constant and never-ending improvement. Your input will greatly help us in our endeavor.

Audience

This manual is primarily intended for market data professionals, messaging specialists, software engineers and architects, systems administrators tasked with installing/uninstalling/maintaining VMA, and ISV partners wishing to test /integrate their multicast consuming/producing applications with VMA.

Document Conventions

NOTE

Text set off in this manner presents clarifying information, specific instructions, commentary, sidelights, or interesting points of information.

IMPORTANT

Text set off in this manner indicates important information regarding a specific feature.

CAUTION

Text set off in this manner indicates that failure to follow directions could result in damage to equipment or loss of information.

Document Organization

This guide contains the following chapters:

- **Chapter 1 – Introduction:** an introductory overview to the VMA Library, including its features and practical applications.
- **Chapter 2 – VMA Library Architecture:** presents an overview of the architecture and internal mechanisms of the VMA Library.
- **Chapter 3 – Installing and Running VMA:** describes the procedure for installing and running VMA.
- **Chapter 4 – Configuring VMA:** shows how to configure VMA. Included is a complete list of all VMA configuration settings and their possible values.
- **Chapter 5 – Debugging, Troubleshooting and Monitoring:** provides basic information on how to analyze and debug a number of issues in the VMA Library as well as present the VMA monitoring and performance counters.
- **Appendix A – UDP Latency Tool: udp_lat:** presents Voltaire's sample application for testing latency with UDP traffic.
- **Appendix B – Multicast Routing:** describes the ways that applications can define the interfaces through which they receive or transmit the various multicast groups.
- **Appendix C – Acronyms:** expands the acronyms used in this document.

Related Documentation

- Voltaire Messaging Accelerator (VMA) Library for Linux Release Notes (DOC-00329)
- Voltaire Linux OFED v5.1.3 HCA 4X0/5X0 User Manual (DOC-00333)
- Voltaire IPR User Manual (399Z00005)

Chapter 1. Introduction

In This Chapter:

This chapter provides an introductory overview to the VMA Library, including its features and practical applications.

This chapter contains the following sections:

1.1 VMA Overview 1-2
1.2 Basic Features 1-2
1.3 Target Applications 1-3
1.4 Advanced VMA Features 1-3
1.5 Extending InfiniBand Performance to GEth Networks 1-4

1.1 VMA Overview

The Voltaire® Messaging Accelerator (VMA) Library is a multicast-offload, dynamically linked user space Linux library for transparently enhancing the performance of multicast networking-heavy applications over the InfiniBand network. Designed for multicast consumer applications and multicast producer applications equally, VMA enhances application performance by orders of magnitude without requiring any modification to the application code.

The VMA library accelerates UDP multicast socket applications by offloading their traffic to InfiniBand directly from user-space to the network interface card (HCA), without going through the kernel and IP stack. VMA increases multicast overall packet rate, reduces latency, and improves CPU utilization.

1.2 Basic Features

Some of the benefits different applications can gain by using the VMA Library are:

- Utilizes InfiniBand **direct hardware access** and advanced polling techniques
  - VMA **Kernel bypass** enabled by utilizing the InfiniBand direct HW access. The VMA Library bypasses the network stack for all multicast traffic transmit and receive socket API calls:
    - Reduces **Context Switches** and **Interrupts**:
      - > Lower latencies
      - > Higher throughput
      - > Improved CPU utilization
    - Minimal buffer copies between user data and hardware – a single copy only is all that is needed by the VMA to transfer a multicast offloaded packet from/to hardware to/from the application's data buffers

1.3 Target Applications

Good candidates for VMA include but are not limited to:

- Market data feed handler software consuming multicast data feeds (and often using multicast as a distribution mechanism downstream)
- Messaging applications responsible for producing/consuming relatively large amounts of multicast data including applications using messaging middleware such as Tibco Rendezvous (RV) or 29West LBM
- Caching/data distribution applications utilizing multicast for cache creation/state maintenance
- Any other applications making heavy use of multicast that require any combination of the following:
  - Higher packets per second (pps) rates than with Ethernet or IPoIB
  - Lower data distribution latency
  - Lower CPU utilization by the multicast consuming/producing application in order to support further application scalability

1.4 Advanced VMA Features

The VMA Library provides several significant advantages:

- Underlying wire protocol used for the multicast solution is standard UDP/IP/IPoIB and is interoperable with any UDP/IP networking stack. Thus, the opposite side of the multicast communication can be any machine with any OS and located on either an InfiniBand or an Ethernet network

IMPORTANT

Using standard protocol by the VMA enables an application to use the VMA for asymmetric acceleration purposes. A 'multicast consuming' only or a 'multicast publishing' only application can leverage this and still be compatible with Ethernet or IPoIB peers.

- Kernel bypass for multicast transmit and receive operations. This delivers much lower CPU overhead since TCP/IP stack overhead is not incurred
- Reduced number of context switches. All VMA software is implemented in user space in the user application's context. This allows the server to process a significantly higher rate of multicast messages than would otherwise be possible
- Minimal buffer copies. Data is transferred from the hardware (HCA) straight to the application buffer in user space with only a single intermediate user space buffer and zero kernel IO buffers
- Fewer hardware interrupts for received/transmitted packets

- Fewer queue congestion problems witnessed in standard TCP/IP applications
- Supports legacy socket applications over native InfiniBand API and semantics
- Maximizes Msgs/Second (MPS)
- Minimizes message latency
- Reduces latency spikes (outliers)
- Lowers CPU usage required to handle multicast traffic

1.5 Extending InfiniBand Performance to GbE Networks

The VMA advantages can be extended beyond InfiniBand fabrics to Ethernet networks by using the Voltaire Ethernet-to-InfiniBand router modules, also known as IPRs. A Voltaire IPR is a high performance InfiniBand-to-Ethernet hardware bridge designed for easy installation onto Voltaire's InfiniBand Switches. The IPRs enable enterprise-class network applications to span servers that reside on Gigabit Ethernet networks and InfiniBand fabrics while maximizing the performance characteristics of each network technology.

The IPR provides logically transparent connectivity between the two network environments by automatically converting InfiniBand IPoIB traffic to Ethernet using specialized silicon to perform the conversion.

Using the IPR to connect servers running VMA on InfiniBand with servers running standard multicast traffic on Ethernet NICs improves both the packet rate (PPS) and total end-to-end latency. Another significant benefit is reduced CPU overhead.

Chapter 2. VMA Library Architecture

In This Chapter:

This chapter presents an overview of the architecture and internal mechanisms of the VMA Library.

This chapter contains the following sections:

2.1 Top-level 2-2
2.2 Unicast Support 2-4

2.1 Top-level

The VMA Library is a dynamically linked user space library. Using Voltaire's VMA Library requires no code changes or recompiling of user applications. Instead, it is dynamically loaded via the Linux OS environment variable, **LD_PRELOAD**.

When a user application transmits multicast data or listens for multicast data, the VMA Library intercepts the socket receive and send calls made to the datagram socket address families. Then, instead of allowing the multicast packet to pass on to the usual OS network kernel libraries, the Voltaire library implements the underlying work in user space. It does this by implementing native InfiniBand API and semantics, and enabling the multicast packets to be passed directly between the user application and InfiniBand HCA (InfiniBand Host/Channel Adapter), bypassing the kernel and its UDP handling stack. This enables the user to exploit InfiniBand native capabilities, the code being implemented in native InfiniBand APIs, so freeing users from the need to make any changes to their applications. The VMA Library does all the heavy lifting under the hood while transparently presenting the same standard socket API to the application and so redirecting the data flow.

The VMA Library operates in a standard networking stack fashion to serve multiple network interfaces, namely Ethernet or InfiniBand.

The VMA Library behaves according to the way the application calls the **setsockopt** directives and the administrator sets the route lookup to determine the requested interface to be used for the socket traffic. The library knows whether data is passing to/from an Ethernet NIC or to/from an InfiniBand HCA. If the data is passing to/from an InfiniBand HCA the VMA Library intercepts the call and does the bypass work. If the data is passing to/from an Ethernet NIC the VMA Library passes the call to the usual kernel libraries responsible for handling multicast. Thus, the same application can listen in on multiple Ethernet NICs and InfiniBand HCAs without any configuration changes being required for the hybrid environment.

The following diagram presents the system architecture of the VMA library, showing the location of the library in relation to multicast applications the sockets library, the OS kernel space, and the HCA:

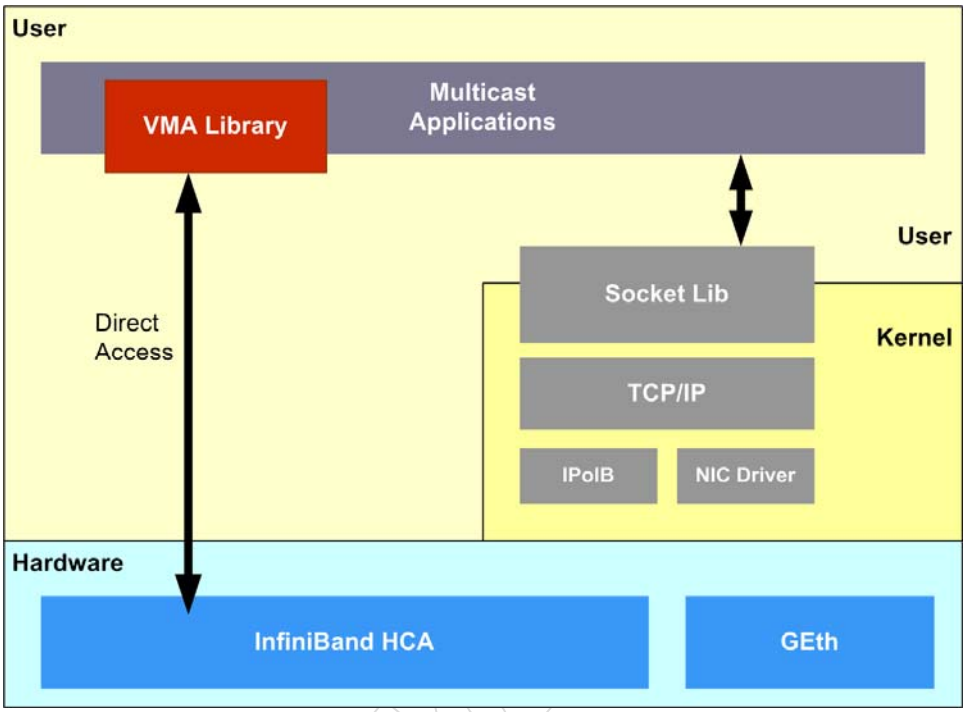

Figure 2-1. VMA Library System Architecture

2.2 Unicast Support

In addition to handling all multicast packet traffic, the VMA Library can also offload unicast packets that satisfy the following two conditions:

- UDP packets reordering is not permitted
- A single application socket is being used for both multicast and unicast packets transfer

For all other cases it is preferred to allow the OS to handle the unicast traffic. By default, the VMA routes all unicast traffic to/from the OS socket.

IMPORTANT

Note that while multicast packets handled by the VMA Library are interoperable with IPoIB and UDP/IP, VMA unicast handled packets are not. Thus:

IF both peers of a unicast connection are operating VMA

=> VMA unicast packets offload can be enabled (and if so, it must be enabled at both peers)

ELSE

=> VMA unicast packets offload must be disabled at both peers

Chapter 3. Installing and Running VMA

In This Chapter:

This chapter describes the procedure for installing and running VMA.

This chapter contains the following sections:

3.1 Prerequisites 3-2
3.2 Installation 3-2
3.3 Uninstalling 3-2
3.4 Upgrading 3-3
3.5 Query Install Package Information 3-3
3.6 Running 3-3

3.1 Prerequisites

- One of the supported operating systems must be running on one of the supported CPU architectures. See Release Notes for more details.
- Make sure that one of the supported Voltaire Linux OFED based stacks is installed on your system prior to VMA installation. See VMA Release Notes for more details.

3.2 Installation

The VMA Library is delivered by Voltaire as a user space library, and is called **libvma.X.Y.Z**.

Install the package as any other rpm package, as follows:

```
#rpm -i libvma.X.Y.Z-R.rpm
```

The installation copies the VMA library to **/usr/lib[64]/libvma.so.**

The VMA monitoring utility is installed at: **/usr/bin/vma_stat**.

A proprietary synthetic latency test for multicast is installed at: **/usr/bin/udp_lat**

The installation location of the **README.txt** and **VMA_VERSION** (version information) file is as follows:

- Redhat: **/use/share/doc/libvma-X.Y.Z-R/**
- SuSE: **/use/share/doc/packages/libvma-X.Y.Z-R/**

3.3 Uninstalling

To uninstall VMA, enter as follows:

```
#rpm -e libvma
```

NOTE

If you wish to uninstall `ofed/gridstack,` **remember to first uninstall VMA.**

3.4 Upgrading

To upgrade VMA, enter as follows:

```
#rpm -U libvma.X.Y.Z-R.rpm
```

When upgrading you can also first uninstall the previously installed VMA package and then start to install the new VMA version.

3.5 Query Install Package Information

To query the installed VMA package information, enter as follows:

```
#rpm -qil libvma
```

If the VMA package is not installed then an appropriate message will be shown.

If the VMA package is installed then the rpm will log the VMA package information and the installed file list.

3.6 Running

Before a user application can be run add the library **`libvma.so`** to the **`env`** variable **`LD_PRELOAD`**.

Example:

```
#LD_PRELOAD=libvma.so iperf -uc 224.22.22.22 -t 5
```

Chapter 4. Configuring VMA

In This Chapter:

This chapter shows how to configure the VMA. Included is a complete list of all VMA configuration settings and their possible values.

The VMA configuration parameters are controlled with system environment variables.

This chapter contains the following sections:

4.1 Configuration Parameters 4-2
4.2 Configuration Values 4-3

4.1 Configuration Parameters

The behavior of the VMA is controlled by VMA configuration parameters. Thus, the user can alter the VMA's behavior by modifying the value of the appropriate parameter.

The behavior of the VMA can be altered by modifying the value of a VMA configuration parameter. The VMA configuration parameters are all Linux OS environment variables. Set these parameters prior to loading the application with the VMA.

All parameters have defaults and all can be modified.

The parameters may be set in a system file, which can be run manually or automatically.

On default startup the VMA Library prints to **`stderr`** the VMA version, the configuration parameters being used and their values.

NOTE

The VMA version information, parameters and values are subject to change.

Example:

```
VMA:  INFO   : ---------------------------------------
VMA:  INFO   : Version: 2.0.25.0
VMA:  INFO   : Revision: 756
VMA:  INFO   : Build Date: 2008-03-10
VMA:  INFO   : Settings:
VMA:  INFO   : ---------------------------------------
VMA:  INFO   :   Log level      3            [VMA_TRACELEVEL]
VMA:  INFO   :   Log file                    [VMA_LOG_FILE]
VMA:  INFO   :   Tx Offload     Enabled      [VMA_TX_OFFLOAD]
VMA:  INFO   :   Tx QP Bufs     1024         [VMA_TX_BUFS]
VMA:  INFO   :   Tx Block Mode  0            [VMA_TX_BLOCK]
VMA:  INFO   :   Tx IP Checksum Enabled      [VMA_TX_IP_CHECKSUM]
VMA:  INFO   :   Tx MC Loopback Enabled      [VMA_TX_MC_LOOPBACK]
VMA:  INFO   :   Rx Offload     Enabled      [VMA_RX_OFFLOAD]
VMA:  INFO   :   Rx QP Bufs     1024         [VMA_RX_BUFS]
VMA:  INFO   :   Rx Sock Limit  512          [VMA_RX_SOCKET_MAX]
VMA:  INFO   :   Rx Polls       200          [VMA_RX_POLL]
VMA:  INFO   :   Rx Drain CQ    Disabled     [VMA_RX_DRAIN_CQ]
VMA:  INFO   :   Rx Skip OS     100          [VMA_RX_SKIP_OS]
VMA:  INFO   :   Select Polls   0            [VMA_SELECT_POLL]
VMA:  INFO   :   Select Skip OS 4            [VMA_SELECT_SKIP_OS]
VMA:  INFO   :   Select IB Irq  Enabled      [VMA_SELECT_IB_INTR]
VMA:  INFO   :   UC Offload     Disabled     [VMA_UC_OFFLOAD]
VMA:  INFO   :   IGMP flag      Enabled      [VMA_IGMP]
VMA:  INFO   :   QP MC Attach   Disabled     [VMA_FORCE_QP_MC_ATTACH]
VMA:  INFO   :   MTU            1500         [VMA_MTU]
VMA:  INFO   : ---------------------------------------
```

4.2 Configuration Values

Table 4-1. Configuration Values

Setting	Description and Examples
VMA_TRACELEVEL	**0 = PANIC –** Panic level logging. This would generally cause fatal behavior and halt the application. Typically, this is caused by memory allocation problems. This level is rarely used.
	1 = ERROR – Runtime errors in the VMA. Typically, these can aid the developer identify internal logic errors, such as: errors from underlying OS or InfiniBand verbs calls, and internal double mapping/unmapping of objects.
	2 = WARNING – Runtime warning that does not disrupt application workflow. A warning may indicate problems in the setup or the overall setup configuration. E.g., address resolution failures (due to an incorrect routing setup configuration), corrupted IP packets in the receive path, or unsupported functions requested by the user application.
	3 = INFO – General information passed to the user of the application. This includes configuration logging or general information to aid the developer better use the VMA Library.
	4 = DEBUG – High level insight to the operations performed in the VMA. All socket API calls are logged and internal high level control channels log their activity.
	5 = FUNC – Low level run-time logging of activity. This logging level includes basic Tx and Rx logging in the fast path. Note that using this setting will lower application performance. It is recommended to use this level with the **VMA_LOG_FILE** parameter.
	6 = FUNC_ALL – Very low level run-time logging of activity. This logging level will *drastically* lower application performance. It is recommended to use this level with the **VMA_LOG_FILE** parameter.
VMA_LOG_FILE	Redirects all VMA logging to a specific user defined file. This is very useful when raising the **VMA_TRACELEVEL.** Example: `#VMA_LOG_FILE=/tmp/vma_log.txt`

Setting	Description and Examples
VMA_TX_OFFLOAD **VMA_RX_OFFLOAD**	VMA offloads Rx and Tx path UDP multicast traffic. Enabling or disabling each of these will redirect that path to/from the OS. Range: 0 - Disabled, 1 - Enabled Default: 1 (for both Rx and Tx)
VMA_TX_BUFS **VMA_RX_BUFS**	Number of Tx and Rx data buffers to be allocated for the process. Range: 512 bytes until 64K Default: 1K each The size of the Rx/Tx buffers is determined by the **VMA_MTU** parameter value (see below). If this value is raised then packet rate peaking can be better sustained, however this will increase memory usage. A smaller number of data buffers gives a smaller memory footprint, but may not sustain peaks in data rate.
VMA_TX_BLOCK	Controls the application's blocking behavior with regards to Tx path. Modes are: **VMA_TX_BLOCK = 0** (default) uses the application's blocking mode selection **VMA_TX_BLOCK = 1** forces the application to block on all Tx paths **VMA_TX_BLOCK = 2** forces the application to not block on any Tx path The default mode, **VMA_TX_BLOCK (0)**, is the recommended mode. No other mode is currently recommended.
VMA_TX_IP_CHECKSUM	Disables IP checksum calculation on a send path to improve performance (when working with VMA). Range: 0 - Disabled, 1 - Enabled Default: 1
VMA_TX_MC_LOOPBACK	Sets the initial value used by VMA internally to control multicast loopback packet behavior during transmission. An application that calls **setsockopt()** with **IP_MULTICAST_LOOP** will overwrite the initial value set by this parameter. Range: 0 - Disabled, 1 - Enabled Default: 1
VMA_TX_DROP_MODE	Debug parameter used to check various sent packet dropping modules. Assigning this parameter any value other than 0 (zero) makes it visible on the VMA startup log. Range: 0 - Disabled, 1 - Enabled Default: 0 (= this parameter is hidden from the VMA startup log)

Setting	Description and Examples
VMA_RX_SOCKET_MAX	Socket limit of Rx ready packets. If the application does not drain a particular socket and the limit is reached, newly received datagrams will be dropped.
VMA_RX_POLL	Number of times to unsuccessfully poll an Rx for VMA packets before going to sleep. Range: 0 .. 100,000 Default: 200 This value can be reduced to lower the load upon the CPU, however the price paid for this is that the Rx latency is expected to increase. Recommended values: • 0 – when CPU usage is critical and Rx path latency is not critical • 10000 – when CPU usage is not critical and Rx path latency is critical Once VMA has gone to sleep, if it is in blocked mode it waits for an interrupt; if it is in non-blocked mode it returns -1. This Rx polling is performed when the application is working with direct blocked calls to **recv()**, **recvfrom()** and **recvmsg()**. When the Rx path has successful poll hits (see Chapter 5, Debugging, Troubleshooting and Monitoring) the latency is improved dramatically. However, this causes increased CPU utilization.
VMA_RX_DRAIN_CQ	When Disabled, each receive call returns the user buffer for only the first ready packet from the socket's ready queue. When Enabled, each receive call drains all packets from hardware (all WCE from CQ) before returning to the user. When this parameter is enabled, VMA may use more CPU but will show more accurate vma_stats. Range: 0 - Disabled, 1 - Enabled Default: 0

Setting	Description and Examples
VMA_RX_SKIP_OS	The number of successfully polled, offloaded VMA packets that should be received before even once going to poll the OS for non-offloaded packets (e.g., from unicast or non-InfiniBand interfaces). Range: 0 .. 10,000 Default: 100 Thus, this parameter controls the ratio of attempting to receive OS packets versus the ratio of attempting to receive VMA offloaded packets. In case of successful Rx polling ("poll hit") the VMA can skip checking the OS for non-offloaded packets (since it puts the system into Sleep mode and in that mode it anyway checks the OS for non-offloaded packets). The Rx fd can quickly return the ready Rx packet to the application for processing. Comparison with **VMA_RX_POLL**: **VMA_RX_POLL** controls the number of times to *failures* in receiving a VMA packet before timing out, whereas this parameter controls how many *successful*, consecutive VMA packets must be received before then doing the next thing, namely checking the OS queues.
VMA_SELECT_POLL	The number of times to poll an Rx for VMA packets before going to sleep (when waiting and also when calling select()). Range: 0 .. 100,000 Default: 0 When the selected path has successfully received poll hits (see Chapter 5, Debugging, Troubleshooting and Monitoring) the latency is improved dramatically. However, this comes on account of CPU utilization.
VMA_SELECT_SKIP_OS	Similar to **VMA_RX_SKIP_OS**, but in **select()** this will force the VMA to check the non-offloaded sockets even though an offloaded socket has a ready packet that was found while polling. Range: 0 .. 10,000 Default: 4
VMA_SELECT_IB_INTR	When disabled no InfiniBand interrupts are used during **select()** socket calls. This mode of work is not recommended. This parameter is used by applications that use **VMA_SELECT_POLL** for polling (with the default zero millisecond timeout). Range: 0 - Disabled, 1 - Enabled Default: 0

Setting	Description and Examples
VMA_UC_OFFLOAD	When enabled, VMA will offload from the OS all unicast UDP packets as well as multicast packets. Range: 0 - Disabled, 1 - Enabled Default: 0 When disabled, VMA will redirect all unicast UDP traffic to/from the OS. The main reason for enabling this is to prevent reordering unicast and multi-cast packets on a single socket.  **IMPORTANT** **This parameter is to be enabled ONLY for RMDS.** Not interoperable with UDP/IP unicast. Must have VMA on both sides.
VMA_IGMP	Whether the `igmp` flag is enabled / disabled (see the release notes in the section on IGMP support). Range: 0 - Disabled, 1 - Enabled Default: 1
VMA_MTU	Size in bytes of each Rx and Tx data buffer. This value sets the fragmentation size of the packets sent by the VMA Library. Default: 1500 bytes Recommendations: • 2044 is recommended for InfiniBand-only networks • 1500 is recommended for interoperability with Ethernet network networks

Chapter 5. Debugging, Troubleshooting and Monitoring

In This Chapter:

This chapter provides basic information on how to analyze and debug a number of issues in the VMA Library. When troubleshooting problems be sure to have first studied this information; it will help you to better monitor and identify the problems and solve them. This chapter also presents the VMA monitoring and performance counters**.**

This chapter contains the following sections:

5.1 Monitoring - vma_stats Utility 5-2
5.2 Debugging 5-3
5.3 Troubleshooting 5-3

5.1 Monitoring - vma_stats Utility

Networking applications open various types of sockets. The VMA library holds separate performance counters for each socket of the datagram (UDP) IP family type. The VMA internal performance counters accumulate information also for **select()** usage by the whole application.

Use the included **vma_stats** utility to view the per-socket information and performance counters during run time.

Usage:

```
#vma_stats <pid> [<info_level>]
```

where:

pid is the process ID using **libvma.so** for which the performance counters are requested

info_level = 1 will cause some configuration information to be logged concerning each socket

If the user application performed transmit or receive activity on a socket then these values will be logged when the sockets are closed. The VMA logs its internal performance counters if **VMA_TRACELEVEL=4** (see Table 4-1, "Configuration Values").

Below is an example of a log of socket performance counters with explanation of the results:

```
VMA:  sockinfo:fd[10]: Tx Offload: 643 KB / 448 / 0 [bytes/packets/errors]
VMA:  sockinfo:fd[10]: Tx OS info: 0 KB / 0 / 0 [bytes/packets/errors]
VMA:  sockinfo:fd[10]: Rx Offload: 643 KB / 448 / 0 [bytes/packets/errors]
VMA:  sockinfo:fd[10]: Rx OS info: 0 KB / 0 / 0 [bytes/packets/errors]
VMA:  sockinfo:fd[10]: Rx poll: 1 / 2040 (99.99%) [miss/hit]
VMA:  sockinfo:fd[10]: Rx ready: max 1 / dropped 0 (0.00%)
```

The log of socket performance counters indicates that communications are performing well:

- No transmission or reception errors on this socket (user **fd=10**)
- No packets transmitted or receive via the OS
- Almost no missed Rx polls (see **VMA_RX_POLL**). This implies that the receiving thread did not enter a blocked state. Thus, there was no context switch to hurt latency
- No dropped packets caused by socket limit (see **VMA_RX_SOCKET_MAX**)

5.2 Debugging

VMA Logs

Use the VMA logs in order to trace VMA operations. VMA logs can be controlled by the **VMA_TRACELEVEL** variable. This variable's default value is 3, which means that the only logs obtained are with severity of PANIC, ERROR and WARNING.

Increase its value up to 6 (as described in Chapter 4) to see more information about each thread's operation.

IPoIB Counters

Look at the IPoIB counters (by using the **ifconfig** command) in order to understand if the traffic is passing through the kernel or through VMA (RX and TX).

5.3 Troubleshooting

- On running an application with VMA the following error is reported:

```
ERROR: ld.so: object 'libvma.so' from LD_PRELOAD cannot be
preloaded: ignored.
```

  Check again if **libvma** is properly installed, and if **libvma.so** is located in **/usr/lib** (or in **/usr/lib64,** for 64-bit machines)

- On attempting to install **vma rpm** the following error is reported:

```
#rpm –ivh libvma-w.x.y-z.rpm
error: can't create transaction lock
```

  Install the rpm with privileged user (root).

- One of the following warnings is reported:

```
VMA:  WARNING: fd[6]:connect() address 224.1.2.3 failed resolving as
Tx on IB interfaces
```

  or

```
VMA:  WARNING: fd[6]:setsockopt() address 224.1.2.3 failed resolving
as Rx MC on IB interfaces
```

  Use the **route** command to check that multicast addresses in the route table are mapped to the IPoIB interface. If they are not, map them as follows:

```
#route add -net 224.0.0.0 netmask 240.0.0.0 dev <ipoib interface>
```

- One of the following warnings is reported:

```
VMA:  WARNING: fd[6]:connect() address 224.1.2.3 failed resolving as
TX on IB for interfaces 192.168.1.100
```

or

```
VMA:  WARNING: fd[6]:setsockopt() address 224.1.2.3 failed resolving
as Rx MC on IB for interfaces 192.168.1.100
```

Check that your application that is loaded with VMS is configured to use the correct InfiniBand interface IP address and that the InfiniBand interface is in its running state.

- The following warning is reported:

```
VMA:  WARNING:*******************************************************
VMA:  WARNING: Your current max locked memory is: 33554432. Please
change it to unlimited.
VMA:  WARNING: Set this user's default to `ulimit -l unlimited`.
VMA:  WARNING: Read more about this issue in the VMA's User Manual.
VMA:  WARNING:*******************************************************
```

When working with root, increase the maximum locked memory to 'unlimited' by the following command:

```
#ulimit -l unlimited
```

When working as a non-privileged user, ask your administrator to increase the maximum locked memory to unlimited.

- The following warning is reported:

```
VMA:  WARNING:
*************************************************
VMA:  WARNING: Error in reading UMCAST flag for interface
192.168.0.10 while VMA_IGMP is enabled!
VMA:  WARNING: Working in this mode most probably causes VMA
performance degradation
VMA:  WARNING: Please "export VMA_IGMP=0" before loading your
application with VMA library
VMA:  WARNING: Read the IGMP section in the VMA's User Manual for
more information
VMA:  WARNING:
*************************************************
```

This warning message means that you are using VMA with an older version of OFED that does not support user space IGMP.

You can disable VMS_IGMP=0 if you do not need to receive multicast packets from the Ethernet to the InfiniBand fabric.

If you do expect to receive multicast packets from the Ethernet to the InfiniBand fabric with VMA then you need to upgrade your OFED based network stack.

- When the following warning is reported:

```
VMA:  WARNING:
*****************************************************
VMA:  WARNING: UMCAST flag is Disabled for interface ib0 while
VMA_IGMP is Enabled!
VMA:  WARNING: Working in this mode most probably causes VMA
performance degradation
VMA:  WARNING: Please "echo 1 > /sys/class/net/ib0/umcast" or
"export VMA_IGMP=0" before loading your application with VMA library
VMA:  WARNING: Read the IGMP section in the VMA's User Manual for
more information
VMA:  WARNING:
*****************************************************
```

  This warning message means that you are using VMA while the user space IGMP support in OFED is disabled.

  You can disabled VMS_IGMP=0 if you do not need to receive multicast packets from the Ethernet to the InfiniBand fabric.

  If you do expect to receive multicast packets from the Ethernet to the InfiniBand fabric with VMA then you need to enable the user space IGMP support for your InfiniBand interface:

```
#echo 1 > /sys/class/net/ib0/umcast
```

Appendix A UDP Latency Tool: udp_lat

In This Appendix:

This chapter presents **udp_lat,** Voltaire's sample application for testing latency with UDP traffic. **udp_lat** uses VMA monitoring and performance counters to measure network latency characteristics between two machines on a fabric.

This appendix contains the following sections:

A.1 Overview A-2
A.2 Running the Test A-2
A.3 Debugging for udp_lat A-5
A.4 Troubleshooting for udp_lat A-5

A.1 Overview

udp_lat operates by sending packets from the client to the server, which then sends the packets back to the client. This measured round-trip time is the route trip time (RTT) between the two machines on a specific network path.

udp_lat tests the improvement of UDP traffic latency when running applications with and without the VMA.

Dividing the total number of packets that performs this round trip by some fixed period of time gives the average RTT.

Dividing the average RTT by 2 gives the average latency for a given one-way path between the two machines.

A.2 Running the Test

udp_lat is installed on the machine at **/usr/bin/udp_lat**.

Following are a number of examples of running **udp_lat**:

A.2.1 Running over IPoIB

1. First configure the route table to map multicast addresses to the IPoIB interface on both client and server machines, as follows:

   ```
   #route add -net 224.0.0.0 netmask 240.0.0.0 dev ib0
   ```

   In this case ib0 is the IPoIB interface.

   Now we are ready to run the test.

2. Run the server as follows:

   ```
   #udp_lat –s –i 224.1.1.1
   ```

   The following output is obtained:

   ```
   udp_lat: [SERVER] listen on:  224.1.1.1 port 11111
   ```

   Now the server is waiting for the client's message.

3. Run the client as follows:

   ```
   #udp_lat –c –i 224.1.1.1
   ```

   The following output is obtained:

   ```
   udp_lat: [CLIENT] send on:  224.1.1.1 port 11111
   udp_lat: Client Start sending ...
   client_sig_handler:client_counter=21220 in 1 sec, latency=23.562
   [usec]
   ```

4. Interpretation of the results:

   The above example shows an average latency of 23.562 microseconds, in which 21396 messages where sent in 1 second.

A.2.2 Running over VMA

1. As in the previous test, configure the route table to map multicast addresses to the IPoIB interface, on both the client and server:

```
#route add -net 224.0.0.0 netmask 240.0.0.0 dev ib0
```

   In this case ib0 is the IPoIB interface.

2. Run the server, as follows:

```
#VMA_RX_POLL=10000 LD_PRELOAD=libvma.so udp_lat –s –i 224.1.1.1
```

   Note the setting of **VMA_RX_POLL** to 10000 before loading the VMS. This is to ensure successful polling.

   The following output is obtained:

```
VMA:  INFO   : Version: x.x.x.x
VMA:  INFO   : Settings:
VMA:  INFO   : ---------------------------------------
VMA:  INFO   :   Log level      3       [VMA_TRACELEVEL]
VMA:  INFO   :   Log file               [VMA_LOG_FILE]
VMA:  INFO   :   Tx QP Bufs     1024    [VMA_TX_BUFS]
VMA:  INFO   :   Rx QP Bufs     1024    [VMA_RX_BUFS]
VMA:  INFO   :   Rx Sock Limit  128     [VMA_RX_SOCKET_MAX]
VMA:  INFO   :   Rx Polls       10000   [VMA_RX_POLL]
VMA:  INFO   :   Tx Block Mode  0       [VMA_TX_BLOCK]
VMA:  INFO   :   Skip OS calls  4       [VMA_SKIP_OS_CALL]
VMA:  INFO   :   Checksum       1       [VMA_CHECKSUM]
VMA:  INFO   :   IGMP flag      0       [VMA_IGMP]
VMA:  INFO   :   UC Offload     Off     [VMA_UC_OFFLOAD]
VMA:  INFO   :   MTU            1500    [VMA_MTU]
VMA:  INFO   : ---------------------------------------
udp_lat: [SERVER] listen on:  224.1.1.1 port 11111
```

   Now the server is waiting for the client's message.

3. Run the client, as follows:

```
# VMA_RX_POLL=10000 LD_PRELOAD=libvma.so udp_lat –c –i 224.1.1.1
```

   Note the setting of **VMA_RX_POLL** to 10000 before loading the VMS. This is to ensure successful polling.

The following output is obtained:

```
VMA:  INFO   : Version: x.x.x.x
VMA:  INFO   : Settings:
VMA:  INFO   : --------------------------------------
VMA:  INFO   :   Log level      3       [VMA_TRACELEVEL]
VMA:  INFO   :   Log file               [VMA_LOG_FILE]
VMA:  INFO   :   Tx QP Bufs     1024    [VMA_TX_BUFS]
VMA:  INFO   :   Rx QP Bufs     1024    [VMA_RX_BUFS]
VMA:  INFO   :   Rx Sock Limit  128     [VMA_RX_SOCKET_MAX]
VMA:  INFO   :   Rx Polls       10000   [VMA_RX_POLL]
VMA:  INFO   :   Tx Block Mode  0       [VMA_TX_BLOCK]
VMA:  INFO   :   Skip OS calls  4       [VMA_SKIP_OS_CALL]
VMA:  INFO   :   Checksum       1       [VMA_CHECKSUM]
VMA:  INFO   :   IGMP flag      0       [VMA_IGMP]
VMA:  INFO   :   UC Offload     Off     [VMA_UC_OFFLOAD]
VMA:  INFO   :   MTU            1500    [VMA_MTU]
VMA:  INFO   : --------------------------------------
udp_lat: [CLIENT] send on:  224.1.1.1 port 11111
udp_lat: Client Start sending ...
client_sig_handler:client_counter=67060 in 1 sec, latency=7.456
[usec]
```

4. Interpretation of the results:

   The above example shows an average latency of 7.456 microseconds, in which 67060 messages where sent in 1 second.

A.2.3 Running over Ethernet

1. First configure the route table to map multicast addresses to the Ethernet interface, on both client and server:

   ```
   #route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
   ```

   In this case eth0 is the Ethernet interface.

   Now we are ready to run the test.

2. Run the server, as follows:

   ```
   #udp_lat –s –i 224.1.1.1
   ```

   The following output is obtained:

   ```
   udp_lat: [SERVER] listen on:  224.1.1.1 port 11111
   ```

   Now the server is waiting for the client's message.

3. Run the client, as follows:

   ```
   #udp_lat –c –i 224.1.1.1
   ```

The following output is obtained:

```
udp_lat: [CLIENT] send on:  224.1.1.1 port 11111
udp_lat: Client Start sending ...
client_sig_handler:client_counter=7012 in 1 sec, latency=71.368
[usec]
```

4. Comparing the results, the latency obtained in the above example:
   - IPoIB: 23.562 [usec]
   - VMA: 7.456 [usec]
   - Ethernet: 71.368 [usec]

   Using VMA we see a latency improvement of 16.106 usec compared to standard IPoIB (68%). And a latency improvement of 63.912 usec over Ethernet (89%).

A.3 Debugging for udp_lat

Run **`udp_lat --help`** to get usage information.

Notes:

- The message size or duration for which the test runs can be modified.
- A multicast IP address or a unicast IP address can be used.
- Different UDP ports can be used.

A.4 Troubleshooting for udp_lat

udp_lat error:

```
udp_lat: No messages were received from the server. Is the server down?
```

If the above error is received check the connection between the client and server, check the route table entries for the multicast group, and make sure the server is running.

Appendix B Multicast Routing

This appendix describes the ways that applications can define the interfaces through which they receive or transmit the various multicast groups.

All applications that receive and/or transmit multicast traffic on a multiple interface host should define the network interfaces through which they would prefer to receive or transmit the various multicast groups.

One method of defining the network interface is if a networking application can use existing socket API semantics for multicast packet receive and transmit to map the multicast traffic. In this case the route table does not have to be updated for multicast group mapping.

`setsockopt` is the socket API that than handles these definitions.

When the application uses **`setsockopt`** with **`IP_ADD_MEMBERSHP`** for the receive path multicast join request it defines the interface through which it wants the VMA to join the multicast group and listen for incoming multicast packets for the specified multicast group on specified socket.

When the application uses **`setsockopt`** with **`IP_MULTICAST_IF`** on the transmit path it defines the interface through which the VMA will transmit outgoing multicast packet on that specific socket.

If the user application does not use some or all the above socket lib **`setsockopt`** API calls then the VMA uses the network routing table mapping to find the appropriate interface to be used for receiving or transmitting multicast packets. Use the **`route`** command to check that multicast addresses in the route table are mapped to the InfiniBand interface. If they not mapped you can map them as follows:

```
#route add -net 224.0.0.0 netmask 240.0.0.0 dev ib0
```

It is best to perform the above mapping before running the user application with VMA so that multicast packets are routed via the InfiniBand interface and not via the default Ethernet interface eth0.

The general rule is that the VMA routing is the same as the IPoIB routing.

Appendix C Acronyms

API	Application Programmer's Interface
CQ	Completion Queue
FD	File Descriptor
GEth	Gigabit Ethernet Hardware Interface
HCA	Host Channel Adaptor
HIS	Host Identification Service
IB	InfiniBand
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPoIB	IP over IB
IPR	IP Router
MTU	Maximum Transmission Unit
NIC	Network Interface Card
OFED	OpenFabrics Enterprise Distribution
OS	Operating System
pps	Packets Per Second
QP	Queue Pair
RMDS	Reuters Market Data System
RTT	Route Trip Time
UDP	Unreliable Datagram Packet
usec	microseconds

UMCAST	User Mode Multicast
VMA	Voltaire Messaging Accelerator
VMS	Voltaire Messaging Service
WCE	Work Completion Elements