OPENFABRICS ALLIANCE

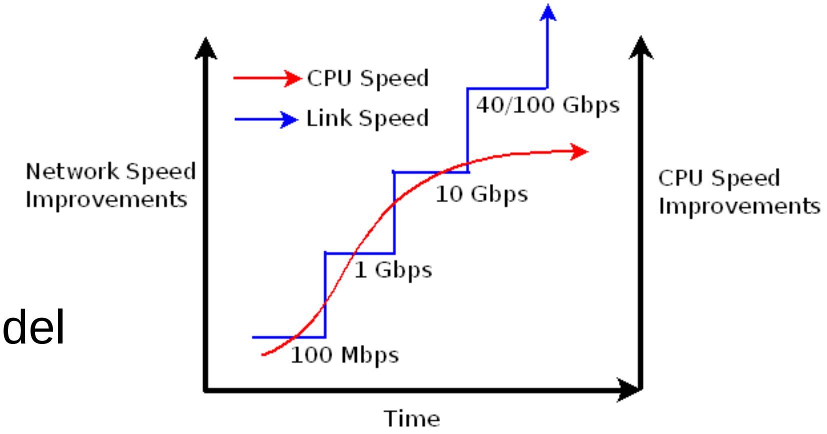
jVerbs: Java/OFED Integration for the Cloud

Authors: Bernard Metzler, Patrick Stuedi, Animesh Trivedi.
IBM Research – Zurich

Date: 03/27/12

Motivation

- The commodity Cloud is
  - Flexible computing at large scale
  - Network heavy
  - Built out of commodity hardware
  - Virtualized
  - Using Java as a main programming model
- Cloud interconnect
  - Commodity 10 GbE is there, more to come
  - Low latency/high throughput puts burden on end hosts CPU
    - Today's Java network stack less efficient than native C program using sockets
    - Cloud performance becomes I/O bound
    - RDMA typically requires dedicated costly hardware
- Lets put things together
  - Commodity RDMA stack + RDMA enabled Java
  - Accelerate given Java applications and enable new RDMA inspired communication patterns

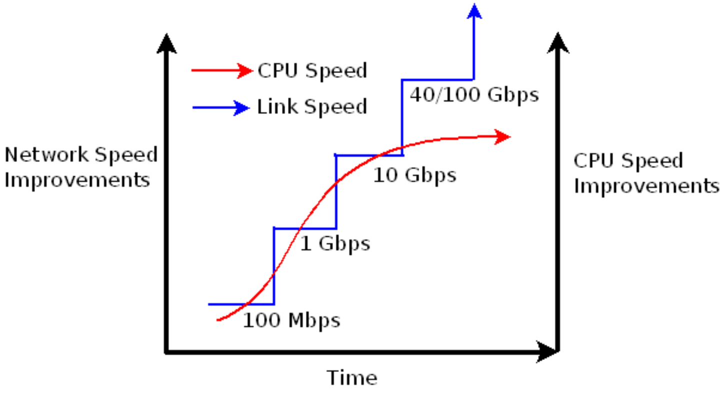


Java and OFED/RDMA in Cloud

RDMA

+ NW stack efficiency
- Explicit resource management
- Cost/dedicated hardware

?



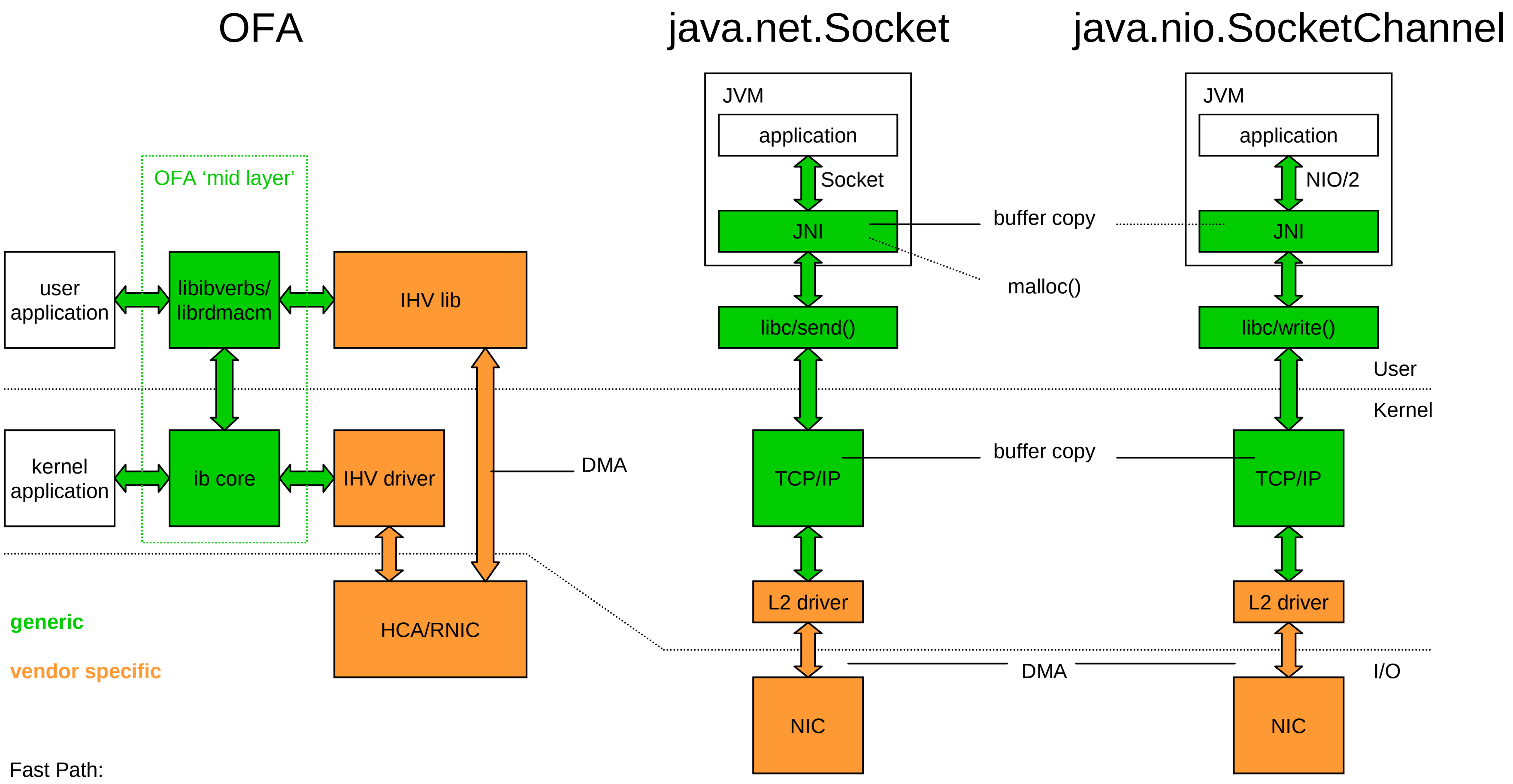
THE CLOUD

- Large scale
- Cost efficient
- Dynamic
- Flexible
- Virtualized

+ Flexibility
+ Portability
+ Core cloud programming model
- Network performance

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates. Other product and service names might be trademarks of IBM or other companies.

Some Network Stacks



OFA

java.net.Socket

java.nio.SocketChannel

OFA 'mid layer'

user application

libibverbs/ librdmacm

IHV lib

kernel application

ib core

IHV driver

HCA/RNIC

DMA

generic

vendor specific

JVM

application

Socket

JNI

libc/send()

TCP/IP

L2 driver

NIC

buffer copy

malloc()

buffer copy

DMA

JVM

application

NIO/2

JNI

libc/write()

TCP/IP

L2 driver

NIC

User

Kernel

I/O

Fast Path:

- Zero copy, no CPU involved

- Copy (1) from heap and (2) in TCP/Socket
- Potential malloc()
- CPU intensive

- Using directBuffers may avoid one copy
- Asynchronous, extended in NIO2
- Less CPU intensive

Levels of RDMA/Java Integration

Avoiding the Effort

1. Use SDP Sockets

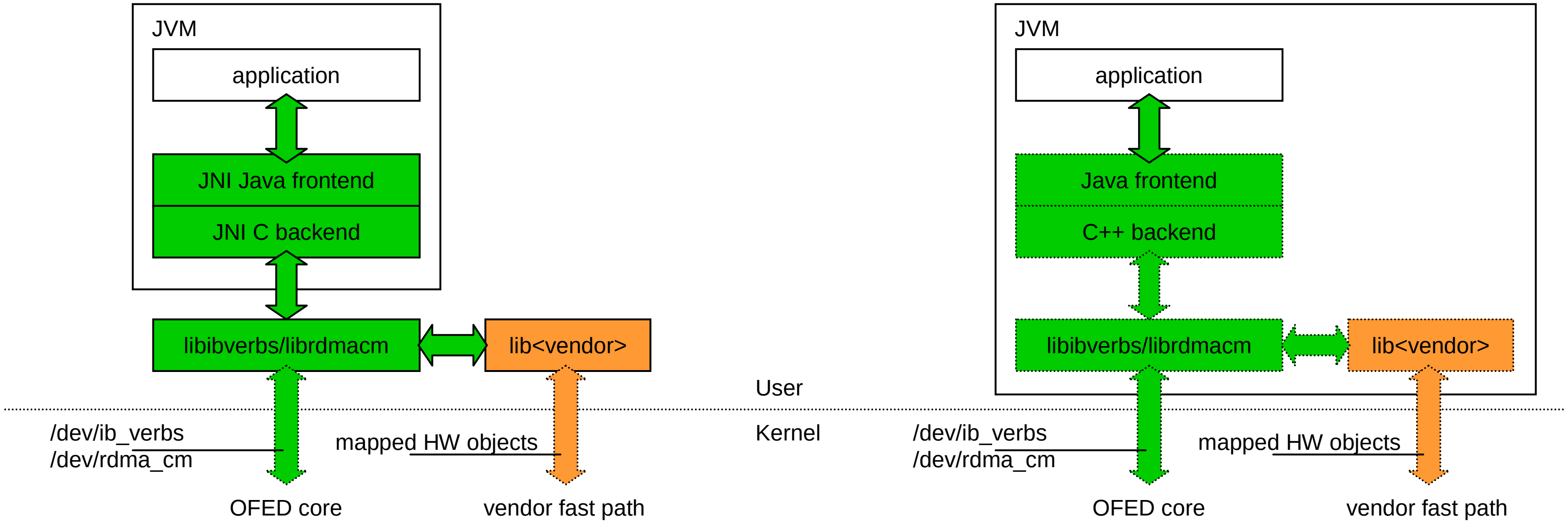
    - Implicit RDMA deployment only
        - Some RDMA benefits for all applications using NIO and java.net
        - Application: no RDMA semantic available and no changes

Doing the Integration

2. Use JNI to attach to libibverbs/librdmacm
3. Full JVM integration of libibverbs/librdmacm
4. **jVerbs**: Re-write user space OFED mid layer as a jar library to seamlessly integrate with unchanged JVM

    - Explicit RDMA deployment with one sided operations possible
    - Availability of RDMA semantics can be tailored at API
        1. Using Java sockets or NIO translates to implicit RDMA calls
        2. NIO2: Match async. API semantics with native RDMA calls
        3. New native RDMA: Provide RDMA verbs-like native API

JNI vs full JVM Integration of OFED



JNI:

- Java frontend providing verbs, and
- C backend to call ibverbs

+ Use OFED environment, no IHV dependencies

- Performance: JNI internal buffer copy, call marshalling

JVM extension:

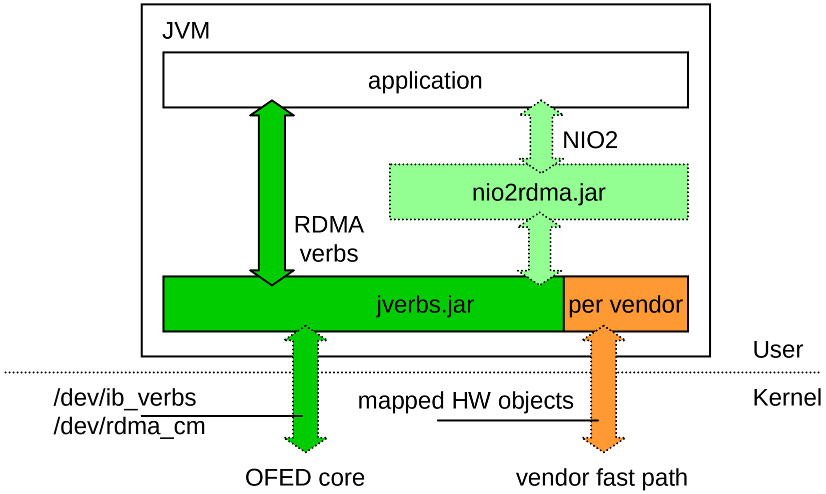
- Java Verbs frontend
- Code directly calling libibverbs added to JVM

+ Performance: full integration into JVM

- JVM changes which are platform dependent

- Provider specific code in JVM

jVerbs Basics



jVerbs:

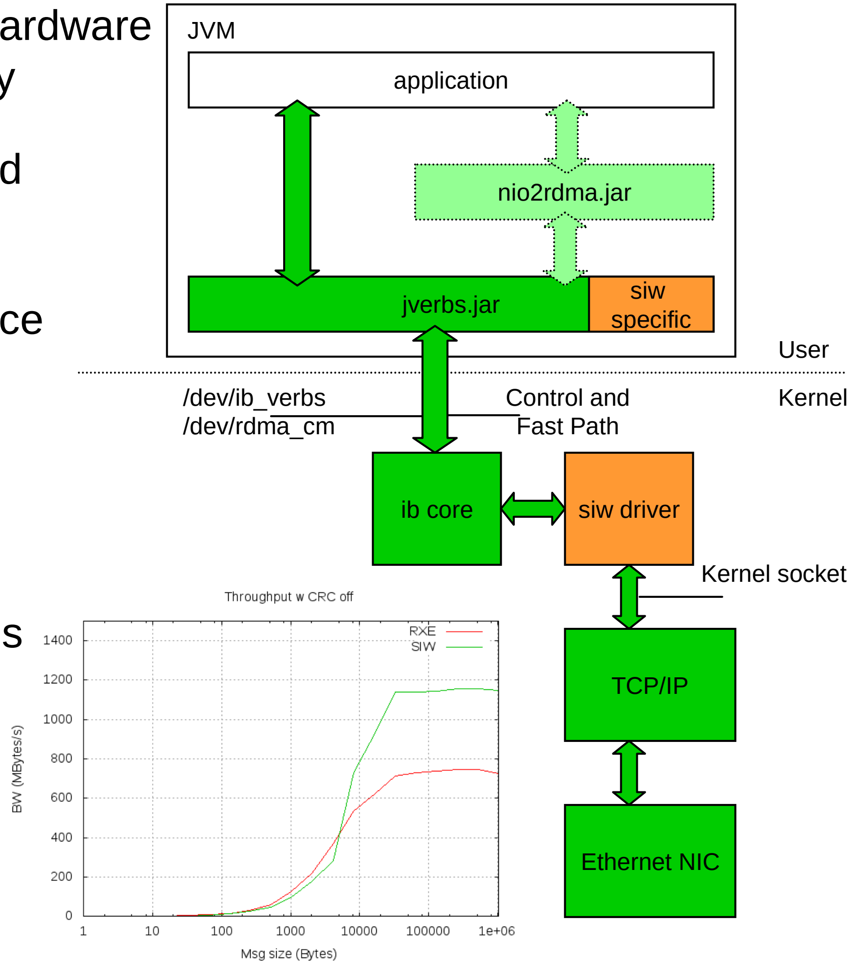
- Regular Java library
- Implements functionality of libibverbs, librdmacm and lib<vendor>
- Provides verbs interface to application

+ No intermediate layers
+ Zero copy if application uses direct buffers
- Vendor specific code (as with OFED user code)

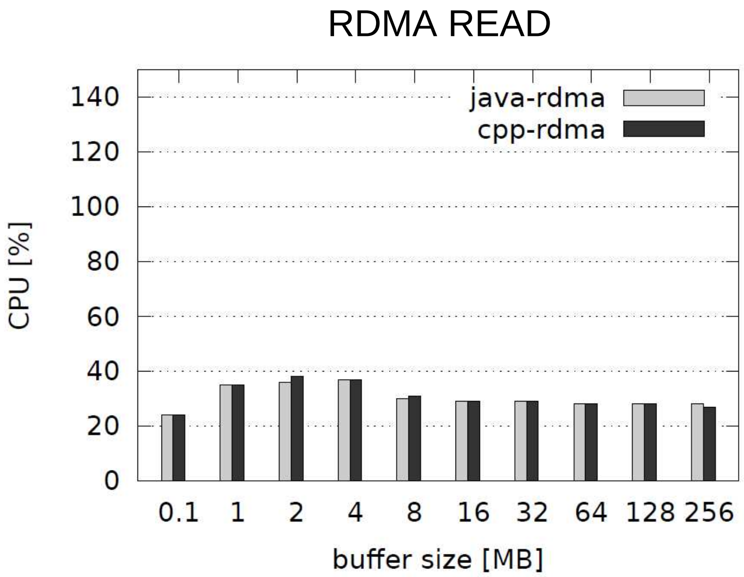
- jVerbs OFED Interface
  - Implements OFED’s device I/O protocol
  - Replaces generic libibverbs, librdmacm
  - Contains vendor specific code
    - Extends BaseDriverClass and overrides some methods
    - Resource allocation/queue mapping
    - Fast path to HW
  - Generic fast path through /dev/ib_verbs if supported by vendor
- Complete RDMA verbs API
  - Native RDMA semantic available
  - New Java applications leveraging RDMA
  - Zero copy when using direct buffers
- jVerbs can implement NIO2 interfaces
  - ‘nio2rdma’ library
  - Direct mapping to async. CM and one-sided RDMA operations
  - Allows seamless RDMA support for NIO2 applications

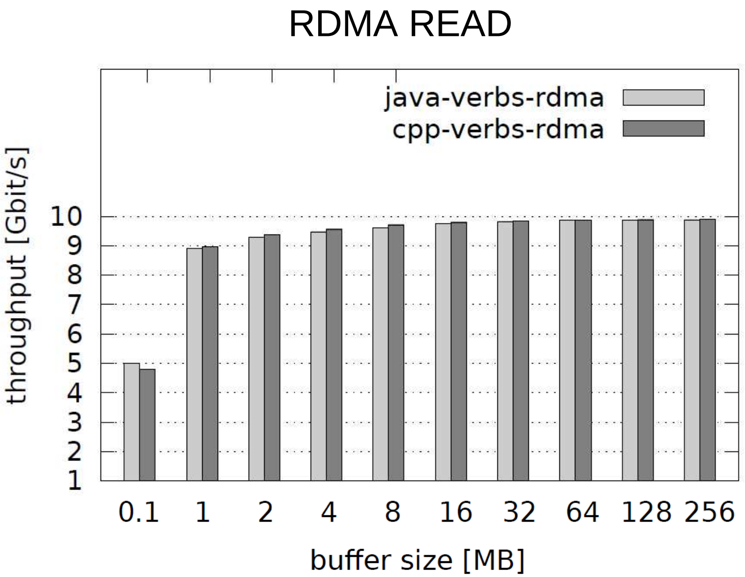
Prototype Implementation

- Software-only RDMA Stack fits Cloud needs
  - Cheap and integrative for heterogonous hardware
  - Flexible host resource mgmt (lazy memory registration etc. possible)
  - Good virtualization support - host-local and host-to-host
- SoftiWARP or SoftRoCE?
  - On given setup, siw with better performance with large packets
    - Plain 10GbE infrastructure (no CEE)
    - GSO/GRO, checksum offload
    - TCP better suited for (today's cloud) non-CEE networks?
  - siw with prototype lazy memory mgmt
  - rxe better performance for small messages (siw lacks user mapped queues)
- Fast path via generic OFED calls
  - No QP/CQ mapping
  - Minimum driver specific code
  - How bad is 1 us extra for posting/reaping work for a Java application?



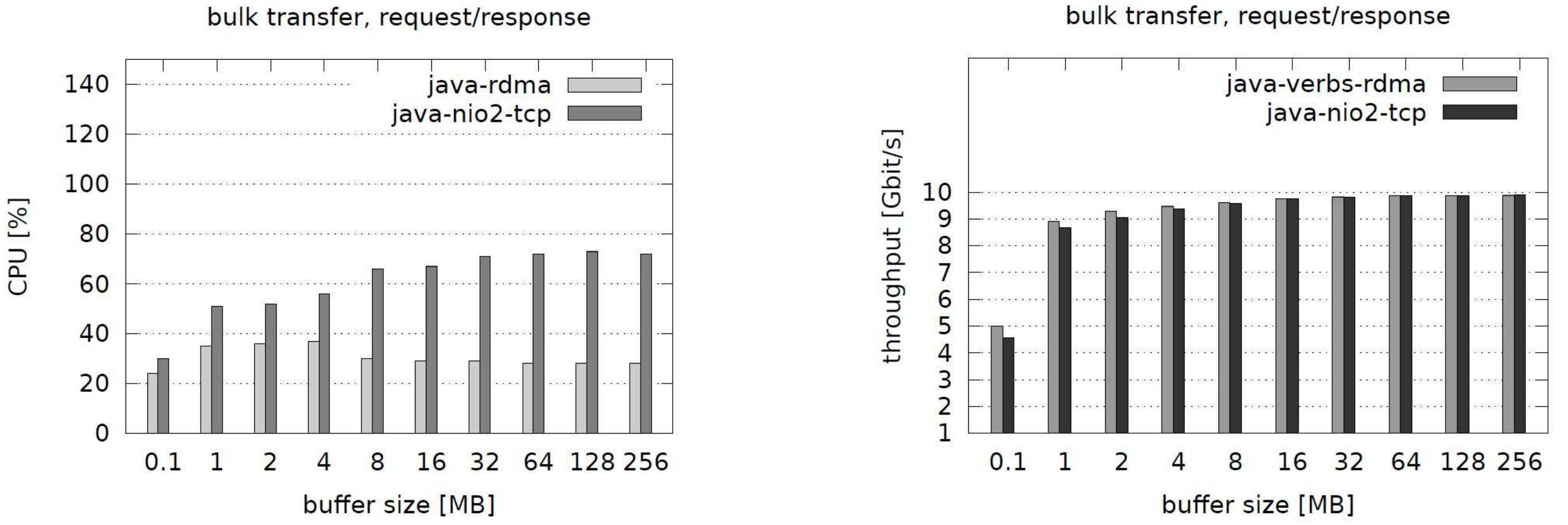
jVerbs versus ibverbs





- Setup:
  - siw as verbs provider (gitorious.org/softiwarp)
  - Xeon E5540 @ 2.53GHz
  - from java directBuffer: zero copy send application
  - Bulk transfer tests (req/resp using RDMA READ's)
- jVerbs performs on par with native C++ application using ibverbs

jVerbs versus TCP in Java

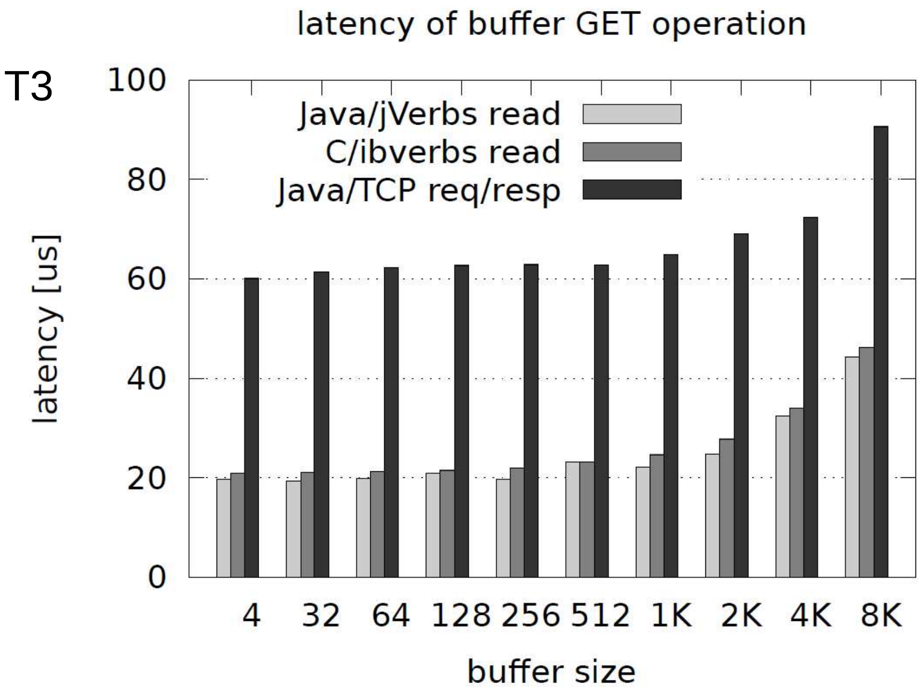


Java-only tests: Either via NIO2/TCP or NIO2/jVerbs

- Same throughput for large messages
- Significant CPU savings using jVerbs
  - Zero copy transmit
  - Caching of memory registration for direct buffers
  - RDMA via siw

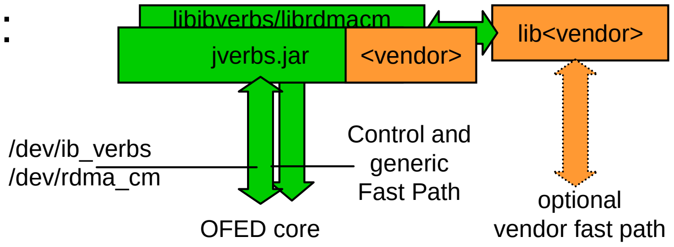
Current and future Activities

- Reaching out for real RDMA hardware
  - Started implementing IHV's private fast path
    - Looking into user mapped objects
  - Estimate for jVerbs stack overhead
    - First encouraging results for Chelsio T3
      - Generic fast path through mid layer
      - Some clever optimizations for call marshalling
      - ‼ *Performance win* compared to libibverbs
    - Mellanox mlx4 is next
- Hadoop™ Distributed File System as an application
  - Written in Java with Java API
  - Large block transfers
  - Allow explicit usage of RDMA semantics



Findings and Suggestions

- Using generic fast path for SQ/RQ/CQ (post/reap):
  - Some 100ns for empty system call might be well invested overhead for additional protection
  - Minimizes HW dependencies
  - Scaling: conserves host resources (avoids mmap() and extra pinned memory etc.)
  - Only costly for polling CQ
  - All HW vendors might support it
- Current generic fast path from user space:
  - Aims at translating it into kernel application call
  - Creates ‘struct ib_send_wr’ out of user cmd
  - Example post_send():
    - kmalloc()’s and kfree()’s for transient objects
      - for the current user WR
      - repeated malloc() for each WR in an array of kernel WR’s
    - opcode specific copy of parameters into WR
    - Some discussion in the past (“RFC kernel path optimizations”) – status?
- Fast path could be optimized for IHV private opaque pass-through of user level WR’s



```
ib_uverbs_write(*filep, __user *buf, …) {
   copy_from_user(&hdr, …)
   ib_uverbs_post_send(buf + sizeof hdr, …) {
       copy_from_user(&cmd, …)
       u_wr = kmalloc(…)
       for (cmd.wr_count) {
            copy_from_user(u_wr, buf, …)
            k_wr = kmalloc(…)
            copy_params(k_wr, u_wr)
            copy_from_user(k_wr->sg_list, …)
            append(k_wr, k_wr_list)
       }
       device->post_send(k_wr_list, …)
       for (cmd.wr_count)
            kfree(k_wr)
       kfree(u_wr)
   }
}
```

Summary

- Integration of OFED/Java
  - Proposed another user space OFED ‘mid layer’ for Java
    - Decouple OFED kernel from user components
    - ‘cVerbs’ *and* ‘jVerbs’ OFED’s coexisting user space components..?
  - No changes to the JVM
  - Native Java-RDMA applications possible
  - Async. NIO2 good match for RDMA communication
  - Performance comparable to native libibverbs app’s.
  - SW based RDMA stacks + jVerbs good fit for Cloud
- Current work
  - Real HCA/RNIC
  - Mapping RDMA provider specific resources
  - Looking at Java cloud applications (HDFS)