Improving Spark Performance with Zero-copy Buffer Management and RDMA

Hu Li, Charley Chen and Wei Xu
Institute for Interdisciplinary Information Sciences
Tsinghua University, China
Latency matters in big data

Big Data: Not only capable, but also interactively

[Kay@SOSP13]
Overview of our work

• NetSpark: A reliable Spark package that takes advantage of the *RDMA over Converged Ethernet (RoCE)* fabric

• A combination of **memory management optimizations** for JVM-based applications to take advantage of RDMA more efficiently

• Improving **latency-sensitive** task performance, while staying fully **compatible** with the off-the-shelf Spark
Background: Remote Direct Memory Access (RDMA)

Lower CPU utilization and lower latency
An overview of NetSpark transfer model

Machine A

- Executor
  - JVM heap
  - Object
  - Serialization
  - Byte Array
  - DMA Read
  - RNIC

Machine B

- Executor
  - JVM heap
  - Object
  - Deserialization
  - Byte Array
  - DMA Write
  - RNIC

Network transfer

User Space

JVM off-heap
Zero-copy network transfer

Traditional Way

- Object
- Serialize
- Byte Array
- Network API (Copy)
- System call (Copy)

Our Way

- Object
- Serialize
- Byte Array
- DMA READ
- RNIC

JVM Heap

JVM Off-heap

Kernel Space
Implementation: SPARK executors

Executor (Spark)

BlockManager

SendingConnections
BlockTransferService (TCP)
ReceivingConnections

Executor (NetSpark)

BlockManager

SendingConnections
BlockTransferService (RDMA)
ReceivingConnections

BufferManager
RDMA buffer management

- RDMA require a fixed physical memory address
  - for Java: off-heap
- Significant allocate/de-allocate cost
- Need to register to RDMA
  - High overhead

Simple solution: Pre-allocate RDMA buffer space to avoid allocation / register overhead
RDMA Buffer Management (cont’d)

- A small number of large-enough fixed-size off-heap buffers
  - Like the Linux kernel buffer, but @ user space
- But … need to copy from heap to off-heap
Serializing directly into the off-heap RDMA buffer

• Rewrite Java InputStream and OutputStream to take advantage of the new buffer manager

• Details in the paper
Evaluation: Testbed

1. 3 switches, 34 servers
2. RoCE, 10GE
3. Using priority flow control for RDMA to avoid packets loss

Network topology of our testbed
Evaluation: Experiment Setup

Compared four different executor implementation

1. Java NIO
2. Netty
3. Naive RDMA
4. NetSpark

(Spark version: 1.5.0)
Group-by performance on small dataset

- Spark example
- 2.5GB data shuffled

About 17% improvement over the naive RDMA
Why do we have an improvement?

- CPU block time
- Measurements from SPARK log
- Following Kay@NSDI15
Group by on larger data - entire reduce stage

A larger dataset about 107.3GB for shuffle

~40% faster over Netty
PageRank on a large graph

- Twitter Graph Dataset
  [Kwak@www2010]
  - 41 million nodes
  - 1.5 billion edges
  - 20% faster than Netty
  - 10% faster than naive RDMA

![Box plot of time distribution for a single iteration with nio, netty, naive rdma, and NetSpark]
Conclusion

• NetSpark: A reliable Spark package that takes advantage of the RDMA over Converged Ethernet (RoCE) fabric

• A combination of memory management optimizations for JVM-based applications to take advantage of RDMA more efficiently

• Improving latency-sensitive task performance, while staying fully compatible with the off-the-shelf Spark

Wei Xu  weixu@tsinghua.edu.cn