sRDMA: A General and Low-Overhead Scheduler for RDMA

Xizheng Wang, Shuai Wang, Dan Li

wang-xz22@mails.tsinghua.edu.cn

June 29, 2023
RDMA is a Trending Topic in HPC and Cloud

Designed for performance

- Lower latency, Higher bandwidth, Lower CPU utilization

RDMA

Octopus’17
XSTORE’20
Hermes’20
A1’20
ScaleRPC’19
TH-DPMS’16

FaRM’14
Wukong’16
FaSST’16
DrTM+R’16
Dare’15

DrTM+H’18
HERD’14
RDMP-KV’20
DSL’18

NAM-DB’17
RAMCloud’15
ccNUMA’18
Grappa’15

CoRM’21
Catfish’19
Derecho’19
C-Hint’14
DaRPC’22
DrTM’15
Hyperloop’18
APUS’17
Data Center Application with RDMA

Distributed applications alleviate performance bottlenecks by leveraging RDMA\textsuperscript{[1,2]}, e.g., Tensorflow and Spark

- Replacing the underlying hardware
- Wrapping upper interfaces


Scheduling Requirements

Net Flow Scheduling helps to improve the performance of distributed applications

• Frequent data interchange during the execution of distributed application workflows
• Data transfer for multi-round iterative tasks should ideally follow the order of iteration rounds.
When Scheduling Encounters RDMA

Scheduling RDMA messages appears to be challenging

- RDMA data plane requires very little CPU intervention
- Modifying hardware protocol stack would be challenging

![Diagram of RDMA scheduling]

- **Application**
  - Buffers
  - Sockets API

- **User**
  - Buffers
  - RDMA Verbs API

- **Kernel**
  - Sockets
  - TCP
  - IPv4/IPv6
  - Network Device

- **Device Driver**
  - Kernel Bypass

4
Existing RDMA Flow Scheduling

Couple scheduling logic to the specific applications[1,2]

• CEFS creates multiple RDMA connections with various priorities
• Argus embeds algorithm into Spark to schedule RDMA message belonging to different iterations


Insight of RDMA Workflow

Lightweight WRs/WCs are used to handle data transmission

- Requester posts WRs to initiate data transmission
- Data transmission follows the arrangement order of WRs
- Requester handles WC(CQE) to finish the data transfer
- Message-level scheduling can be more suitable in certain cases, e.g., DL

Can we schedule data transmissions by sorting WR/WC?
Overview of sRDMA

Message Scheduling by manipulating WRs/WCs at host driver

- Scheduling mechanism: reorganize WRs according to the priorities and policies
- Slicing mechanism: split WRs/WCs to further improve scheduling space and alleviate HOL

General APIs

- Minor modifications to RDMA verbs to convey application priorities
sRDMA Message Scheduling

Rearrange the WRs in the SQ/RQ

- Scheduling early but without embedding in the applications

Priority scheduling and small-message-first scheduling
sRDMA Message Slicing

Partitioning WRs into fixed-sized sub-WRs to slice large message

- Slicing WRs based on the data payload and a fixed threshold
- Enabling mutual retrieval of original WR and sub-WRs through ID
- Further expand scheduling space and alleviate the HOL blocking situation
General sRDMA Scheduling at Driver Layer

Original RDMA process: handle WRs – data transfer – handle WCs

sRDMA Process:
General sRDMA Scheduling at Driver Layer

Original RDMA process: handle WRs – data transfer – handle WCs

sRDMA Process: slicing WRs(not necessary)
General sRDMA Scheduling at Driver Layer

Original RDMA process: handle WRs – data transfer – handle WCs

sRDMA Process: slicing WRs (not necessary) – scheduling WRs/sub-WRs – data transfer
General sRDMA Scheduling at Driver Layer

Original RDMA process: handle WRs – data transfer – handle WCs

sRDMA Process: slicing WRs (not necessary) – scheduling WRs/sub-WRs – data transfer – splice sub-WCs (not necessary) – handle WCs
General sRDMA APIs

Minor changes to the original RDMA interfaces

• Add 8-bit priority to enable `ibv_post_send()` transfer desired order from applications

```c
int ibv_post_send(struct ibv_qp *qp, struct ibv_send_wr *wr, struct ibv_send_wr **bad_wr)
```

```c
struct ibv_send_wr {
    uint64_t wr_id;  /* User defined WR ID */
    struct ibv_send_wr *next; /* Pointer to next WR in list. NULL if last WR */
    struct ibv_sge *sg_list; /* Pointer to the s/g array */
    int num_sge; /* Size of the s/g array */
    enum ibv_wr_opcode opcode; /* Operation type */
    unsigned int send_flags; /* Flags of the WR properties */
...
    uint8_t sRDMA_priority; /* Priority of the WR used in sRDMA */
};
```
Possibility of Inter-Connection Scheduling

Priorities give room for complex scheduling

Manipulating priorities globally enables simple global scheduling

• Connection grouping: different priority segments and token ring
Evaluation: Testbed Setup

Experimental Environment

• 10 GPU servers: 1 NVIDIA K40 GPU, 32 CPU cores, 1 Mellanox CX-6 100Gbps NIC

comparative solutions

• Original RDMA, sRDMA and sRDMA’ (only scheduling)

Microbenchmark

• Perftest and open-source code[1]

Application

• DL model: AlexNet, VGG-19, ResNet-50(Tensorflow)
• Graph task on RGraph

[1] https://github.com/efficient/rdma_bench
Microbenchmarks

Operation latency
• Scheduling and slicing WRs only causes time cost of no more than 100 nanoseconds

CPU overhead
• sRMDA incurs a modest 3% increase in CPU overhead compared to RDMA, even with fully out-of-order WRs

Memory overhead
• Sub-WRs and sub-WCs consume more memory, but it remains within the same order of magnitude
Application-Level Performance

Minimal decrease in throughput and FCT evaluated by perftest

- One-to-many connections: Using perftest to generate continuous flows
- Simulating periodic scheduling or specific out-of-order situations

<table>
<thead>
<tr>
<th></th>
<th>per 1ms</th>
<th>per 0.1ms</th>
<th>per 0.001ms</th>
<th>partial disorder</th>
<th>complete disorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput(Gbps)</td>
<td>80.832</td>
<td>80.827</td>
<td>80.82</td>
<td>80.838</td>
<td>80.825</td>
</tr>
<tr>
<td>FCT(μs)</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
</tr>
</tbody>
</table>
Application-Level Performance

Distributed deep learning model on Tensorflow
- sRDMA and sRDMA’ achieved efficient improvements in AlexNet and VGG-19 models

Distributed social relationship task on Rgraph
- sRDMA achieved 11% shorter application completion time
Conclusion & Future Work

Conclusion

• RDMA revolutionizes network performance for data center applications, yet scheduling challenges arise due to special features
• sRDMA achieves RDMA message scheduling through WR/WC scheduling and slicing, ensuring low overhead, effectiveness, and generality of scheduling

Future Work

• Measure more corner cases
• Design more complete scheduling mechanisms
• Design mature global scheduling mechanisms
Thank you for listening!

Q&A