SPARKUCX – RDMA ACCELERATION PLUGIN FOR SPARK

Peter Rudenko, software engineer
100s of 1000s of data scientists and over 16,000 enterprises use Spark

Spark is 100x faster at processing data than Hadoop

1000+ contributors across 250+ companies

Databricks platform alone spins up 1 million virtual machines per day
Spark’s in-memory model completely changed how shuffle is done

- In both Spark and Hadoop, map output is saved on the local disk
- In Hadoop, map output is then copied over the network to the destined reducer’s local disk
- In Spark, map output is fetched from the network, on-demand, to the reducer’s memory

Memory-to-network-to-memory? RDMA/RoCE is a perfect fit!
SPARK’S SHUFFLE BASICS

Input → Map → Map output → File

Map:
- Reduce task → Fetch blocks
- Reduce task → Fetch blocks
- Reduce task → Fetch blocks
- Reduce task → Fetch blocks

Reduce:
- Spark Master

Publish Map Status

© OpenFabrics Alliance
Shuffling is very expensive in terms of CPU, RAM, disk and network

Spark users try to avoid shuffles as much as they can

Speedy shuffles can relieve developers of such concerns, and simplify applications
2017 SparkRDMA shuffle plugin open sourced [https://github.com/Mellanox/SparkRDMA]
  * Based on disni library (thin wrapper over verbs)
  * Initial customers POC, collected requirements and feedback.

2019 SparkUCX shuffle plugin [https://github.com/openucx/sparkucx]
  * Java wrapper for UCX library implementation
  * Fixes architectural bottlenecks in SparkRDMA

2020 Nvidia Rapids for Spark (to be open sourced)
  * Based on UCX java library for communication
  * GPU + RDMA acceleration
UCX

- UCX is a framework for network APIs and stacks

- UCX aims to unify the different network APIs, protocols and implementations into a single framework that is portable, efficient and functional

- UCX doesn’t focus on supporting a single programming model, instead it provides APIs and protocols that can be used to tailor the functionalities of a particular programming model efficiently

- When different programming paradigms and applications use UCX to implement their functionality, it increases their portability. As just implementing a small set of UCX APIs on top of a new hardware ensures that these applications can run seamlessly without having to implement it themselves
# UCX GOALS

## Unified API
Applications driven, simple, extendable, HW-agnostic

## Focus on performance
Fast, scalable, highly optimized low latency high bandwidth messaging framework

## Production quality
Multi-tier testing, used by top Mellanox customers in production

## Open source
Collaboration between industry, laboratories, and academia

## Innovation
Concepts and ideas from research in academia and industry

## Multi arch/transports
RoCE, InfiniBand, Cray, TCP, shared memory, GPUs, x86, ARM, POWER

---

### Co-design of Network APIs
UCX OVERVIEW

- **APIs**
  - Socket-like stream send/receive, RPC
  - Remote memory access and atomic operations
  - Client/server connection establishment
  - Fully non-blocking

- **Advanced features**
  - Full support for GPU and GPU Direct
  - Multi-rail and fault tolerance
  - Direct verbs for minimal software overhead
  - Thread-safety with separate resources per-thread
  - Interrupt and polling-based progress
  - Smart data transfer protocols (eager, rendezvous, bcopy, zcopy, …)
APACHE SPARK + UCX = ACCELERATED SHUFFLE
Spark allows for external implementations of ShuffleManagers to be plugged in
  • Configurable per-job using: “spark.shuffle.manager”

The plugin interface allows proprietary implementations of Shuffle Writers and Readers, and essentially defers the entire Shuffle process to the new component

SparkUCX utilizes this interface to introduce RDMA in the Shuffle process
Initialization:
- Spark driver allocates global metadata buffer per shuffle stage, to hold addresses and memory keys of data and index files on mappers.

Mapper phase:
- `mmap()` and register index and data files
- Publish `{address, rkey}` to driver metadata buffer (`ucp_put`).

Reduce phase:
- Fetch metadata from driver (`ucp_get`)
- For each block:
  - Fetch offset in data file, from index file (`ucp_get`).
  - Fetch block contents from data file (`ucp_get`).
1. Instantiate ucp context:

```java
UcpContext context = new UcpContext(new UcpParams().requestRmaFeature());
```

2. Register memory on context:

```java
UcpMemory memoryRegion = context.memoryMap(new UcpMemMapParams().setLength(length).allocate());
```

3. Instantiate ucp worker:

```java
UcpWorker worker = context.newWorker(new UcpWorkerParams().setCpu(0).requestWakeupRMA());
```

4. Instantiate ucp endpoint:

```java
UcpEndpoint endpoint = worker.newEndpoint(new UcpEndpointParams().setSocketAddress(InetSocketAddress("1.2.3.4:1234"));
```

5. Perform get/put/send/recv operation on endpoint:

```java
UcxRequest request = endpoint.getNonBlocking(remoteAddress, remoteKey, localBuffer, callback);
```

6. Progress request until it's completed:

```java
worker.progressRequest(request)
```
Accelerating Spark
- Lower Block transfer times (latency and total transfer time)
- Lower Memory consumption and management
- Lower CPU utilization
- GPU Direct

Easy to deploy and configure
- Packed into a single JAR file
- Plugin is enabled through a simple configuration handle
- Allows finer tuning with a set of configuration handles

Configuration and deployment are on a per job basis
- Can be deployed incrementally
SPARK SHUFFLE PERFORMANCE (CPU)
Using default TCP vs SparkUCX (RoCE)

- **Benchmarks:** Terasort + Pagerank
  - [https://github.com/Intel-bigdata/HiBench](https://github.com/Intel-bigdata/HiBench)
- **Terasort:**
  - 1.2 TB input, 10K mappers, 15k reducers
- **Pagerank:**
  - Bigdata Hibench workload (600 Gb), 5K mappers, 15K reducers
- **15 nodes:** Broadwell @ 2.60GHz, 250GB RAM, 500GB HDD
- **ConnectX-5:** Infiniband: 100G EDR. TCP device: IPoIB 100G
- **Red Hat Enterprise Linux Server release 7.5 (Maipo)**
  (kernel: 3.10.0-862.el7.x86_64)
- **MLNX_OFED_LINUX-4.6-1.0.1.1.**
- **Spark-2.4.3, Hadoop-2.9.2, UCX v1.8.0**
- **Deployment guide:**
  [https://docs.mellanox.com/pages/releaseview.action?pageId=19819236](https://docs.mellanox.com/pages/releaseview.action?pageId=19819236)
SPARK SHUFFLE PERFORMANCE (CPU)
Using default TCP vs SparkUCX (RoCE)

![Graph showing performance comparison between TCP and UCX]

<table>
<thead>
<tr>
<th>Stage Id</th>
<th>Description</th>
<th>Time, seconds</th>
<th>Total Time</th>
<th>Reducer Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>count at TeraSort.scala:153</td>
<td>2.3 min</td>
<td>15000/15000</td>
<td>1777.0 GB</td>
</tr>
<tr>
<td>UCX</td>
<td>count at TeraSort.scala:153</td>
<td>1.4 min</td>
<td>15000/15000</td>
<td>1777.0 GB</td>
</tr>
</tbody>
</table>
Inventory Pricing Queries: >5X time reduction

ETL for Logistical Regression Model: >2X time reduction

THANK YOU

Peter Rudenko, software engineer

Mellanox Technologies