Data Processing at the Speed of 100 Gbps using Apache Crail

Patrick Stuedi
IBM Research
The CRAIL Project: Overview

Data Processing Framework
(e.g., Spark, TensorFlow, λ Compute)

Spark-IO  Albis  Pocket

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Crail Store

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  . . .  GPU

100 Gbps
10 μsec
The CRAIL Project: Overview

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

Spark-IO  Albis  Pocket

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Crail Store

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  . . .  GPU

100 Gbps
10 μsec

fast sharing of ephemeral data
The CRAIL Project: Overview

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

Spark-IO  Albis  Pocket

Crail Store

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  GPU

100 Gbps
10 μsec

shuffle/broadcast acceleration

fast sharing of ephemeral data
The CRAIL Project: Overview

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)
- Spark-IO
- Albis
- Pocket

Fast Network, e.g., 100 Gbps RoCE
- 100 Gbps
- 10 μsec

Crail Store
- FS
- Streaming
- KV
- HDFS

Fast Network, e.g., 100 Gbps RoCE
- TCP
- RDMA
- NVMeF
- SPDK

DRAM
NVMe
PCM
GPU

- efficient storage of relational data
- shuffle/broadcast acceleration
- fast sharing of ephemeral data
The CRAIL Project: Overview

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

- Spark-IO
- Albis
- Pocket

Crail Store

- FS
- Streaming
- KV
- HDFS

Fast Network, e.g., 100 Gbps RoCE

- TCP
- RDMA
- NVMeF
- SPDK

100 Gbps
10 μsec

data sharing for serverless applications

- efficient storage of relational data
- shuffle/broadcast acceleration
- fast sharing of ephemeral data

DRAM
NVMe
PCM
GPU
Outline

• Why CRAIL
• Crail Store
• Workload specific I/O Processing
  - File Format, shuffle engine, serverless
• Use Cases:
  - Disaggregation
  - Workloads: SQL, Machine Learning
#1 Performance Challenge (1)

- **2008**
  - 100MB/s
  - 10ms

- **2018+**
  - 10GB/s
  - 10μs

![Graph showing throughput over elapsed time](image-url)

- Hardware limit

Elapsed time (seconds)

Throughput [Gbit/s]
#1 Performance Challenge (2)

<table>
<thead>
<tr>
<th></th>
<th>1 Gbps</th>
<th>HDD</th>
<th>100 Gbps</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>117 MB/s</td>
<td>140 MB/s</td>
<td>12.5 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>cycle/unit</td>
<td>38,400</td>
<td>10,957</td>
<td>360</td>
<td>495</td>
</tr>
</tbody>
</table>

Sorting Application
- Sorter
- Serializer

Data Processing Framework
- Serializers
- Networking
- Operating System
- Networking Stack

sockets
TCP/IP
Ethernet
NIC

filesystem
block layer
iSCSI
SSD
#1 Performance Challenge (2)

<table>
<thead>
<tr>
<th></th>
<th>1 Gbps</th>
<th>HDD</th>
<th>100 Gbps</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>117 MB/s</td>
<td>140 MB/s</td>
<td>12.5 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>cycle/unit</td>
<td>38,400</td>
<td>10,957</td>
<td>360</td>
<td>495</td>
</tr>
</tbody>
</table>

Sorting Application
Data Processing Framework
sockets
TCP/IP
Ethernet
NIC
filesystem
block layer
iSCSI
SSD

Sorter
Serializer
Netty
JVM

Process chunk
In reduce task

Fetch chunk
Over the network

HotNets’16
#1 Performance Challenge (2)

<table>
<thead>
<tr>
<th></th>
<th>1 Gbps</th>
<th>HDD</th>
<th>100 Gbps</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>117 MB/s</td>
<td>140 MB/s</td>
<td>12.5 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>cycle/unit</td>
<td>38,400</td>
<td>10,957</td>
<td>360</td>
<td>495</td>
</tr>
</tbody>
</table>

Sorting Application
Data Processing Framework
sockets
TCP/IP
Ethernet
NIC
filesystem
block layer
iSCSI
SSD
JVM
Netty
Serializer
Sorter

HotNets’16
#1 Performance Challenge (2)

### Software Overhead

Software overhead are spread over the entire stack.

### Bandwidth Table

<table>
<thead>
<tr>
<th></th>
<th>1 Gbps</th>
<th>HDD</th>
<th>100 Gbps</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>117 MB/s</td>
<td>140 MB/s</td>
<td>12.5 GB/s</td>
<td>3.1 GB/s</td>
</tr>
<tr>
<td>cycle/unit</td>
<td>38,400</td>
<td>10,957</td>
<td>360</td>
<td>495</td>
</tr>
</tbody>
</table>

### HotNets’16
#2 Diversity

Diverse hardware technologies / complex programming APIs / many frameworks
#3 Ephemeral Data

Ephemeral data has unique properties (e.g., wide range of I/O size)
CRAIL Approach (1)

Abstract hardware via high-level storage interface
CRAIL Approach (1)

Abstract hardware via high-level storage interface

hardware specific plugins (storage tiers)

most I/O operations can conveniently be implemented on a storage abstraction
CRAIL Approach (1)

Abstract hardware via high-level storage interface

most I/O operations can conveniently be implemented on a storage abstraction

what is the right API? (FS, KV, ?)

hardware specific plugins (storage tiers)
CRAIL Approach (2)

Filesystem-like interface:

- Hierarchical namespace
  - Helps to organize data (shuffle, tmp, etc) for different jobs

- Separate data from metadata plane
  - Reading/writing involves block metadata lookup
  - Cheap on a low-latency network (few usecs)
  - Flexible: data objects can be of arbitrary size

- Specific data types
  - KeyValue files: last create wins
  - Shuffle files: efficient reading of multiple files in a directory

- Let applications control the details
  - Data placement policy: which storage node or storage tier to use
CRAIL Approach (3)

Careful software design:

• Leverage user-level APIs
  – RDMA, NFm, DPDK, SPDK

• Separate data from control operations
  – Memory allocation, string parsing, etc.

• Efficient non-blocking operations
  – Avoid army of threads, let the hardware do the work

• Leverage byte-address storage
  – Transmit no more data than what is read/written
Crail Store: Architecture

- Hierarchical namespace
- Multiple data types
- Distributed storage over DRAM and flash

Crail API (Java & C++)

Metadata server

Storage server

Drum block
NVMe block
Crail Store: Architecture

- Hierarchical namespace, multiple data types
- Distributed storage over DRAM and flash
- Files may spawn multiple storage tiers
- Can be read like a single file

Crail API (Java & C++)

Distributed storage over DRAM and flash

Metadata server

Client library

Storage server

- Metadata
- DRAM block
- NVMe block
Crail Store: Deployment Modes

- Application compute
- DRAM storage server
- Flash storage server
- Metadata server
- Compute/storage co-located
- Compute/storage disaggregated
- Flash storage disaggregation
Performance of a single client running on one core only

Crail reaches line speed at for an I/O size of 1K
Crail remote read latencies (DRAM and NVM) are very close to the hardware latencies.
A single metadata server can process 10M Crail lookup ops/sec
Running Workloads: MapReduce

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

Spark-IO  Albis  Pocket

Crail Store

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  ...  GPU

100 Gbps 10 μsec
val pairs = sc.parallelize(1 to tasks, tasks).flatMap(_ => {
  var values = new array[(Long,Array[Byte])](numKeys)
  values = initValues(values)
}).cache().groupByKey().count()
Sorting 12.8 TB on 128 nodes

Sorting time [sec]

Spark
Spark/Crail

6x

Network Throughput [Gb/s]

Task ID

www.sortingbenchmark.org

<table>
<thead>
<tr>
<th></th>
<th>Spark/Crail</th>
<th>Winner 2014</th>
<th>Winner 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (TB)</td>
<td>12.8</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>98</td>
<td>1406</td>
<td>134</td>
</tr>
<tr>
<td>Total cores</td>
<td>2560</td>
<td>6592</td>
<td>10240</td>
</tr>
<tr>
<td>Network HW (Gbit/s)</td>
<td>100</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Rate/core (GB/min)</td>
<td>3.13</td>
<td>0.66</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Native C distributed sorting benchmark

Sorting rate of Crail/Spark only 27% slower than rate of Winner 2016
Using Crail, a Spark 200GB sorting workload can be run with memory and flash disaggregated at no extra cost.
Running Workloads: SQL

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

Spark-IO     Albis     Pocket

Crail Store

FS          Streaming          KV          HDFS

TCP          RDMA          NVMeF          SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM     NVMe     PCM     GPU
None of the common file formats delivers a performance close to the hardware speed.
Revisiting Design Principles

• Traditional Assumption: CPU is fast, I/O is slow
  – Use compression, encoding, etc.
  – Pack data and metadata together
  – Avoid metadata lookups

• Albis: new file format designed for fast I/O hardware

• Albis design principles
  – Avoid CPU pressure, i.e., no compression, encoding, etc.
  – Simple metadata management
Albis/Crail delivers 2-30x performance improvements over other formats
TPC-DS using Albis/Crail

Albis/Crail delivers up to 2.5x performance gains
Running Workloads: Serverless

Data Processing Framework (e.g., Spark, TensorFlow, λ Compute)

Spark-IO  Albis  Pocket

Crail Store

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  …  GPU

100 Gbps 10 μsec
Serverless Computing

• Data sharing implemented using remote storage
  – Enables fast and fine-grained scaling

• Problem: existing storage platforms not suitable
  – Slow (e.g., S3)
  – No dynamic scaling (e.g. Redis)
  – Designed for either small or large data sets

• Can we use Crail? Not as is.
  – Most clouds don’t support RDMA, NVMe, etc.
  – Lacks automatic & elastic resource management
Pocket

- An elastic distributed data store for ephemeral data sharing in serverless analytics

Pocket dynamically rightsizes storage resources (nodes, media) in an attempt to find a spot with a good performance price ratio.
Pocket: Resource Utilization

Pocket cost-effectively allocates resources based on user/framework hints.
Conclusions

• Effectively using high-performance I/O hardware for data processing is challenging

• Crail is an attempt to re-think how data processing systems should interact with network and storage hardware
  – User-level I/O
  – Storage disaggregation
  – Memory/flash convergence
  – Elastic resource provisioning
References

• Crail: A High-Performance I/O Architecture for Distributed Data Processing, IEEE Data Bulletin 2017
• Albis: High-Performance File-format for Big Data, Usenix ATC’18
• Navigating Storage for Serverless Computing, Usenix ATC’18
• Pocket: Ephemeral Storage for Serverless Analytics, OSDI’18 (to appear)
• Running Apache Spark on a High-Performance Cluster Using RDMA and NVMe Flash, Spark Summit’17
• Serverless Machine Learning using Crail, Spark Summit’18
• Apache Crail, http://crail.apache.org
Contributors

Animesh Trivedi, Jonas Pfefferle, Bernard Metzler, Adrian Schuepbach, Ana Klimovic, Yawen Wang, Michael Kaufmann, Yuval Degani, ...