Data Processing at the Speed of 100 Gbps using Apache Crail

Patrick Stuedi
IBM Research
The CRAIL Project: Overview

Crail Store

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

Spark-IO  Albis  Pocket

FS  Streaming  KV  HDFS

TCP  RDMA  NVMeF  SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM  NVMe  PCM  ······ GPU
The CRAIL Project: Overview

Data Processing Framework (e.g., Spark, TensorFlow, \( \lambda \) 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 \( \mu \)sec

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

shuffle/broadcast acceleration

fast sharing of ephemeral data

100 Gbps
10 μsec
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**
  - 100 Gbps RoCE

- **Storage**
  - TCP
  - RDMA
  - NVMeF
  - SPDK

- **Drives**
  - DRAM
  - NVMe
  - PCM
  - GPU

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

- **Performance**
  - 100 Gbps
  - 10 μsec
The CRAIL Project: Overview

- **Crail Store**
- **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**

- **100 Gbps 10 μsec**
- **Data sharing for serverless applications**
- **Fast network, e.g., 100 Gbps RoCE**
- **DRAM**, **NVMe**, **PCM**, **GPU**

**Descriptions**
- Efficient storage of relational data
- Shuffle/broadcast acceleration
- Fast sharing of ephemeral data
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

100x

Throughput [Gbit/s]

0 20 40 60 80 100

Elapsed time (seconds)

0 100 200 300 400 500

Hardware limit

Graph showing performance comparison between 2008 and 2018+ technologies.
#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><strong>360</strong></td>
<td><strong>495</strong></td>
</tr>
</tbody>
</table>

Heavily layered software run using a tight CPU budget.

Data copies, context switches, cache pollution.
Diverse hardware technologies / complex programming APIs
Ephemeral data has unique properties (e.g., wide range of I/O size)
CRAIL Approach

• Performance: leverage user-level APIs
  - Zero-copy to/from application buffers (even in JVM)
  - Careful software design for the µsec era

• Programming: high-level storage abstraction
  - Most communication patterns can implemented using storage
  - Must be careful to retain hardware semantics

• Ephemeral data: design for short-lived data
  - Fault-tolerance, data size

• Diversity: pluggable storage backends
  - RDMA, NVMf, ReFlex/SPDK, TCP
Crail Store: Architecture

- hierarchical namespace
- multiple data types
- distributed storage over DRAM and flash
Crail Store Properties

- Transmit not more data than what is read/written
  - Holds for RDMA, not yet true for NVMf

- Non-blocking and asynchronous read/writes
  - No background threads need, hardware does the job

- Horizontal tiering
  - Spill to flash if DRAM across the storage space is exhausted

- CPU efficient
  - Storage servers don't do much, hardware does most of the job
Crail Store: Deployment Modes

- Application compute
- DRAM storage server
- Flash storage server
- Metadata server
- Compute/storage co-located
- Compute/storage disaggregated
- Flash storage disaggregation
- Application compute
Crail Store: Read Throughput

Performance of a single client running on one core only

Crail reaches line speed at for an I/O size of 1K
Crail Store: Read Latency

Remote DRAM

Remote NVMe SSD (3D XPoint)

Crail remote read latencies (DRAM and NVM) are very close to the hardware latencies
Metadata server scalability

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

Fast Network, e.g., 100 Gbps RoCE

- TCP
- RDMA
- NVMeF
- SPDK

DRAM

NVMe

PCM

GPU

100 Gbps
10 μsec
Sorting 12.8 TB on 128 nodes

Network utilization for Crail/Spark was around 70Gb/s, compared to 5-10Gb/s for Spark
### How fast is this?

![Spark/Crail Winner 2014 Winner 2016](image)

<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>

[www.sortingbenchmark.org](http://www.sortingbenchmark.org)

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

FS
Streaming
KV
HDFS

Crail Store

TCP
RDMA
NVMeF
SPDK

Fast Network, e.g., 100 Gbps RoCE

DRAM
NVMe
PCM

100 Gbps
10 μsec
Reading Relational Data

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, NVMf, 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](http://crail.apache.org)
Contributors

Animesh Trivedi, Jonas Pfefferle, Bernard Metzler, Adrian Schuepbach, Ana Klimovic, Yawen Wang, Michael Kaufmann, Yuval Degani, ...
Backup
Crail Store: Deployment Modes
CPU cores
Network interfaces
user-mapped network queues
RPC processing
Network interfaces
Meta data
Metadata servers
Lock-free fast map