13th ANNUAL WORKSHOP 2017

CRAIL

A HIGH-PERFORMANCE I/O ARCHITECTURE FOR THE APACHE DATA PROCESSING ECOSYSTEM

Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Bernard Metzler,
Ioannis Koltsidas, Radu Stoica and Nikolas Ioannou

IBM Zurich Research

[March 2017]

AGENDA

- Problem statement
- Introducing Crail
  - The Crail Data Store
  - Crail Modules
  - Clever Storage Tiering
- Applying the Crail approach to Big Data
  - TeraSort
  - Storage Disaggregation
  - NVMeF Tier
  - SQL
- Summary and discussion

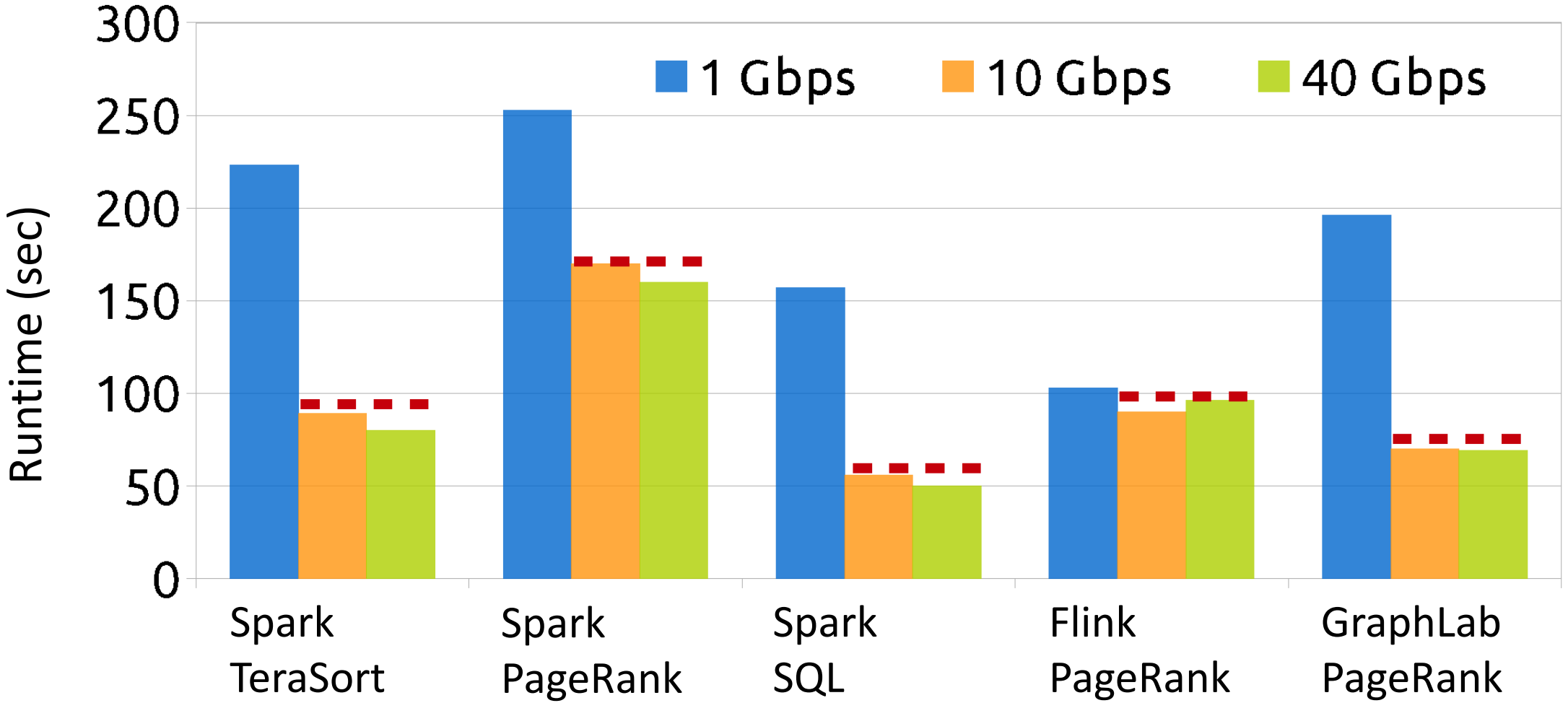
DO BIG DATA PROCESSING FRAMEWORKS BENEFIT FROM MODERN IO TECHNOLOGY?

- **Network interconnects have evolved**
  - From 1Gbs to 100Gbs
  - From 100µs delay to 1µs delay
- **Storage technology has evolved**
  - Factor of 100x – 1000x
- **IO APIs have evolved**
  - From sockets to RDMA verbs
  - From block IO to NVMe
- **Large scale data processing frameworks matured**
- **How do these frameworks benefit from modern IO?**
  - Large scale data processing done in distributed systems
  - IO intensive tasks should benefit
    - Shuffle, inter-job data sharing,…
- **Do we see faster analytics query response times?**

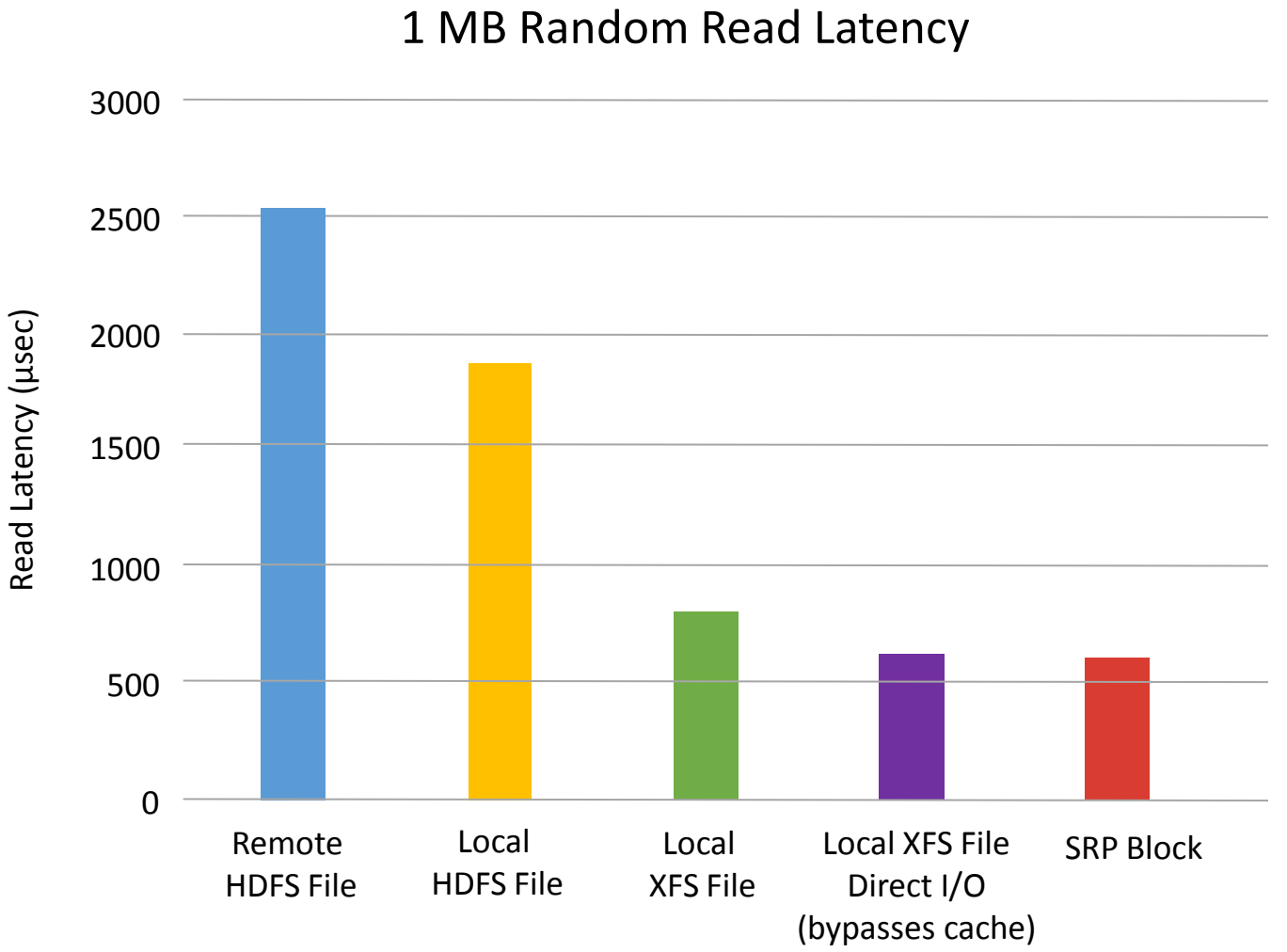
Network IO Performance	RTT (µs)	Bandwidth (Mbit/s)
1GbE (sockets)	82	942
10GbE (sockets	17	9896
100GbE(sockets)	17	63636
100GbE (verbs)	2.4	92560

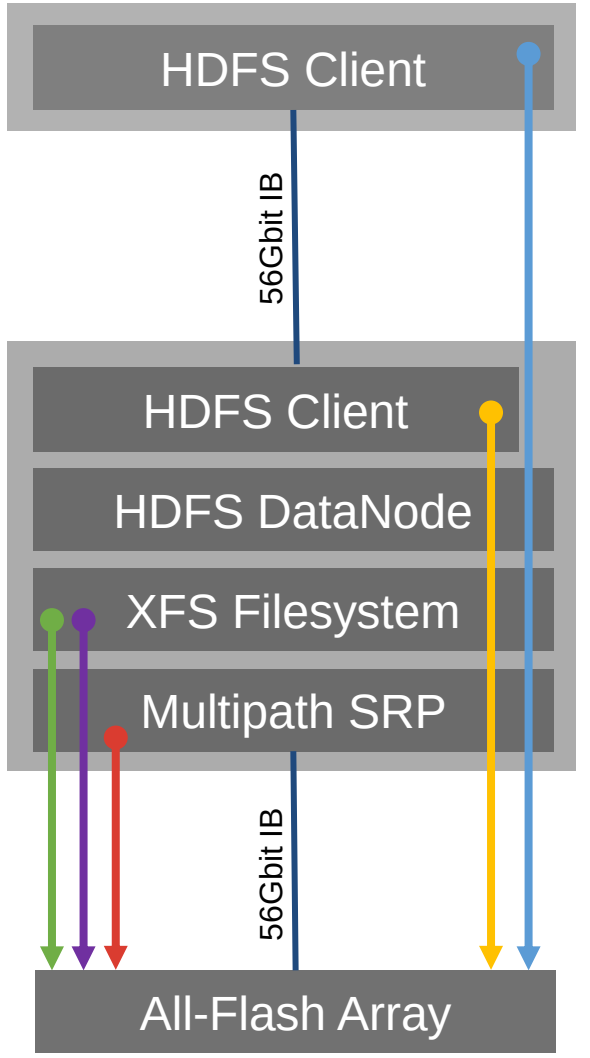
Storage IO Performance	Latency (µs)		Bandwidth (Mbyte/s)	
	read	write	read	write
Disk Block device	5978	5442	136.8	135.3
NVMe Flash SPDK	54.1	8.59	3433.3	2051.7

BIG DATA SYSTEMS CHALLENGED BY HIGH-PERFORMANCE NETWORKS

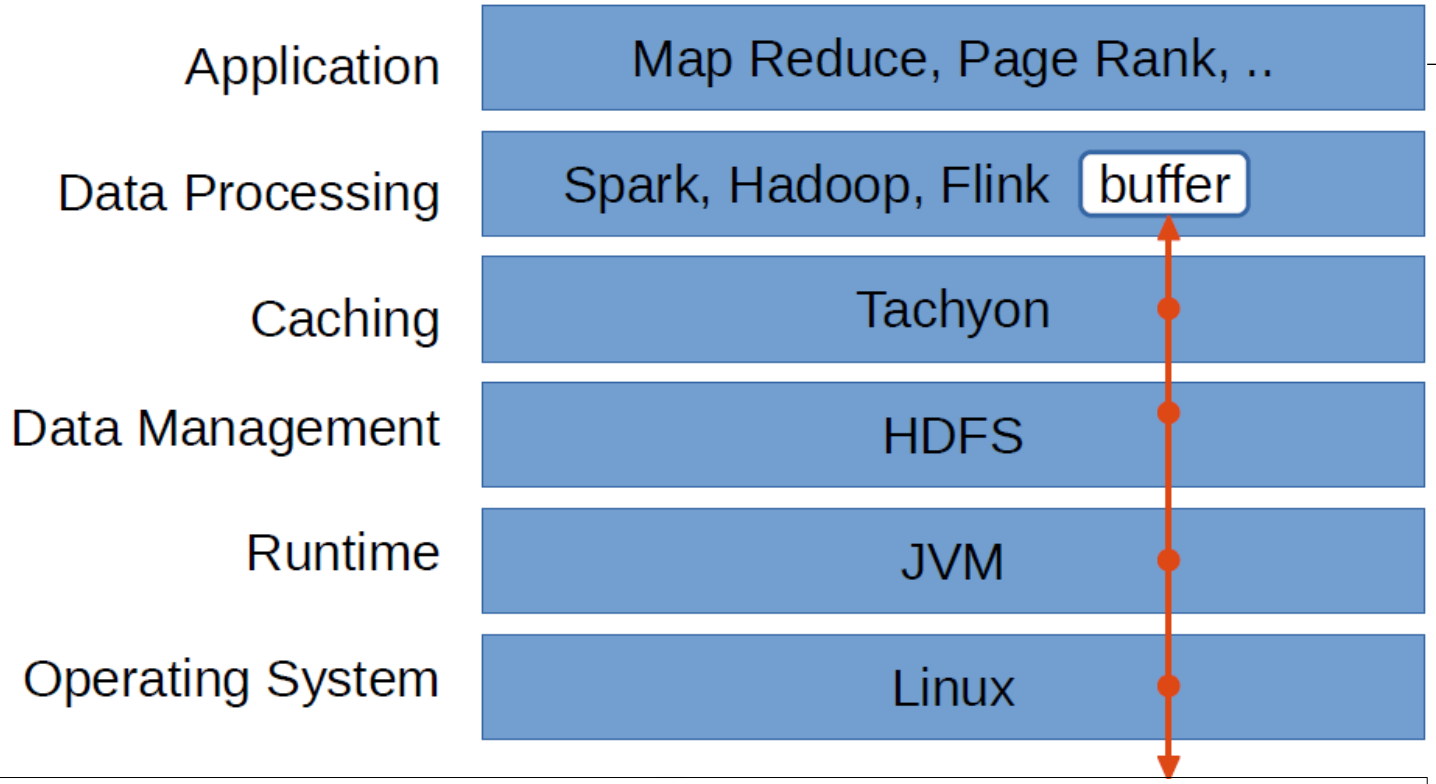


BIG DATA SYSTEMS CHALLENGED BY HIGH-PERFORMANCE STORAGE





HARDWARE PERFORMANCE BURIED IN THE BIG DATA STACK

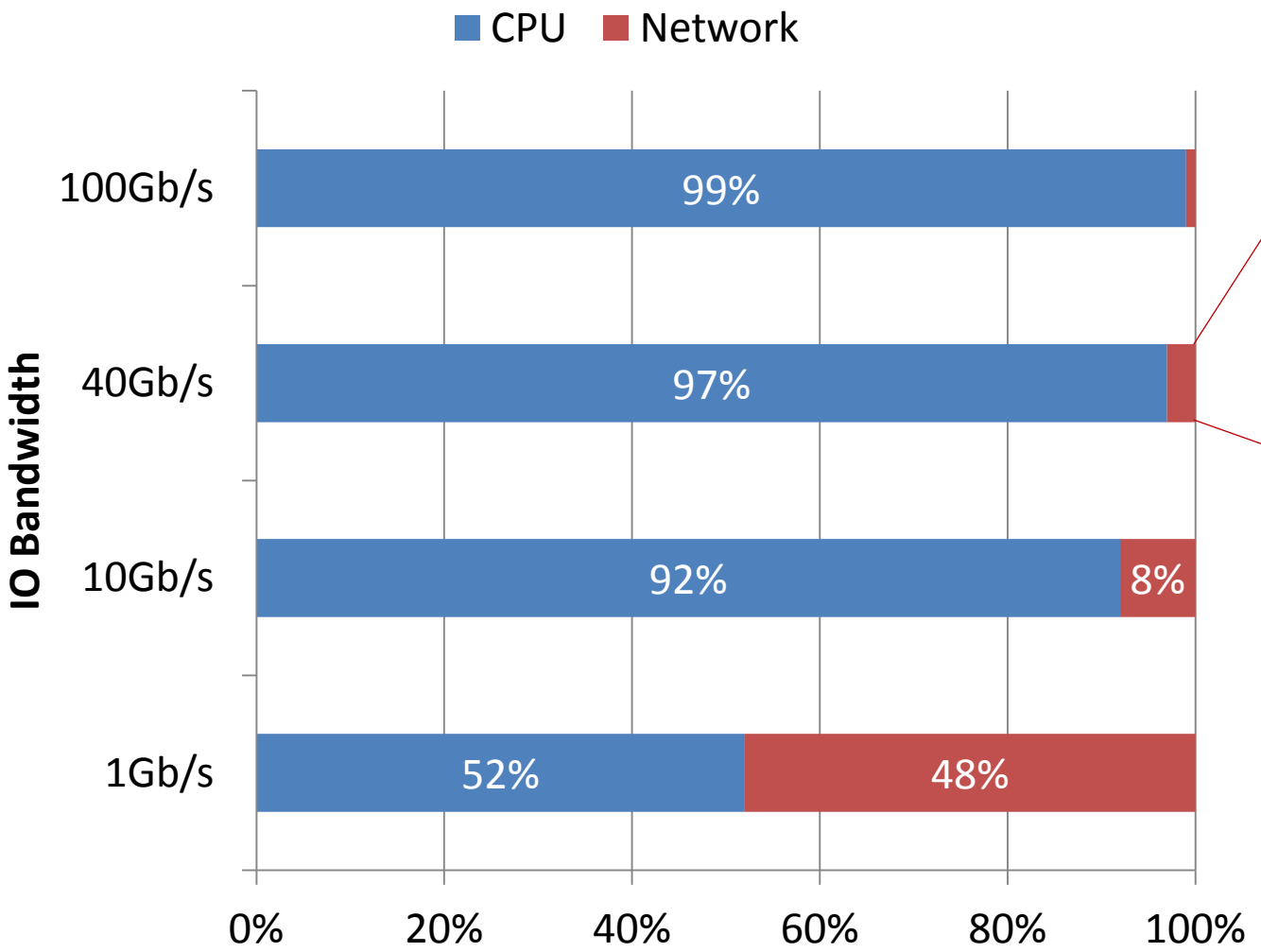


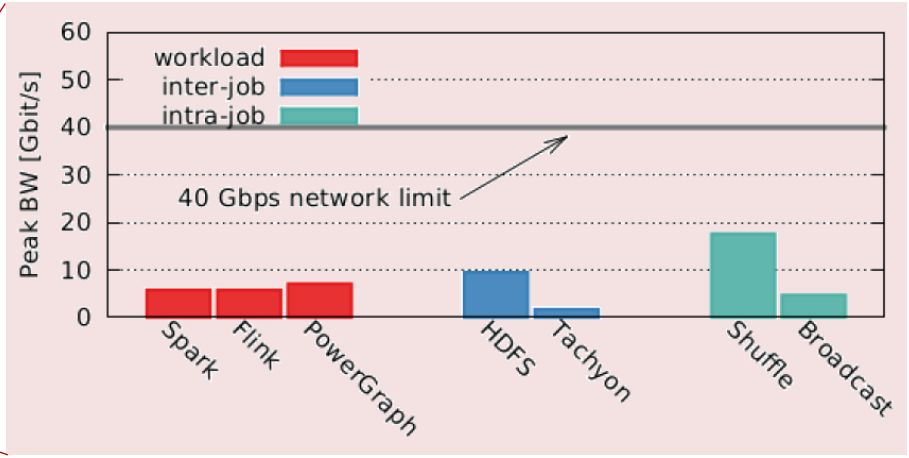
- Data copies
- Context switches
- Cache pollution
- Deep callstack
- Legacy IO interfaces

→ Ineff. local IO stack
→ Remotely even worse
→ Data locality at (almost) any price

Fast I/O Hardware

I/O GAINS ARE SHADOWED BY CPU



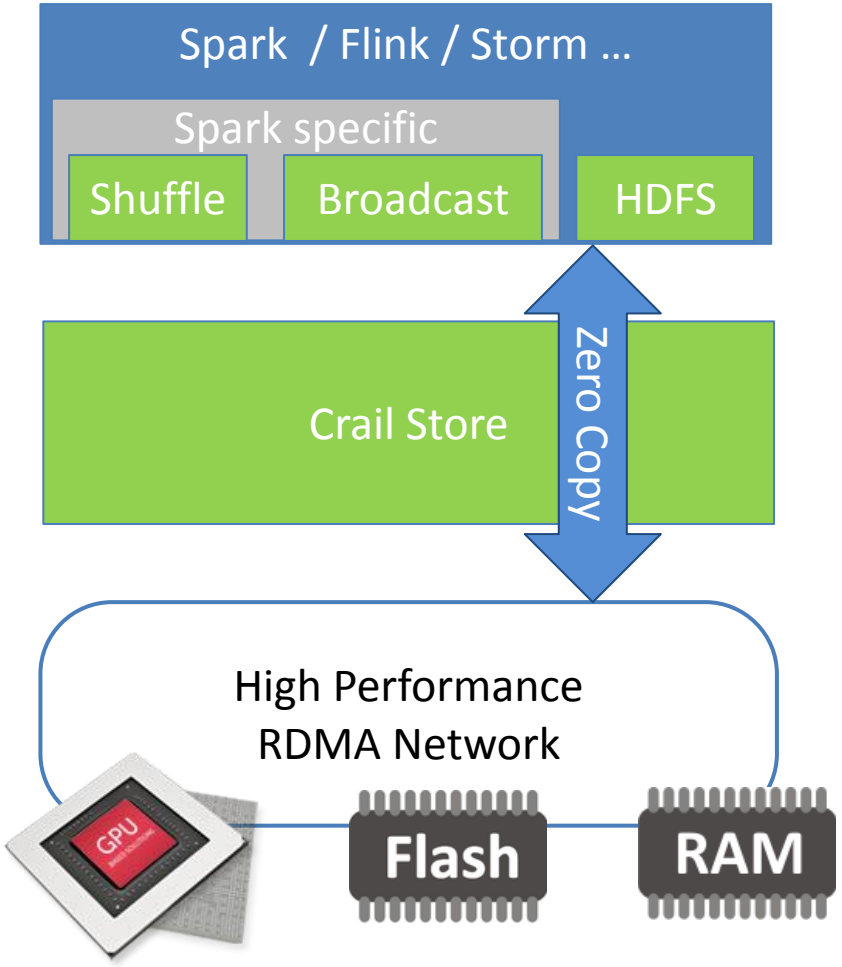


- CPU time is not just application time!
  - Includes all the IO stack overhead
- Faster IO hardware does not help
- IO stack redesign needed

CRAIL: AN ARCHITECTURE TO ENABLE HIGH PERFORMANCE IO IN BIG DATA SYSTEMS

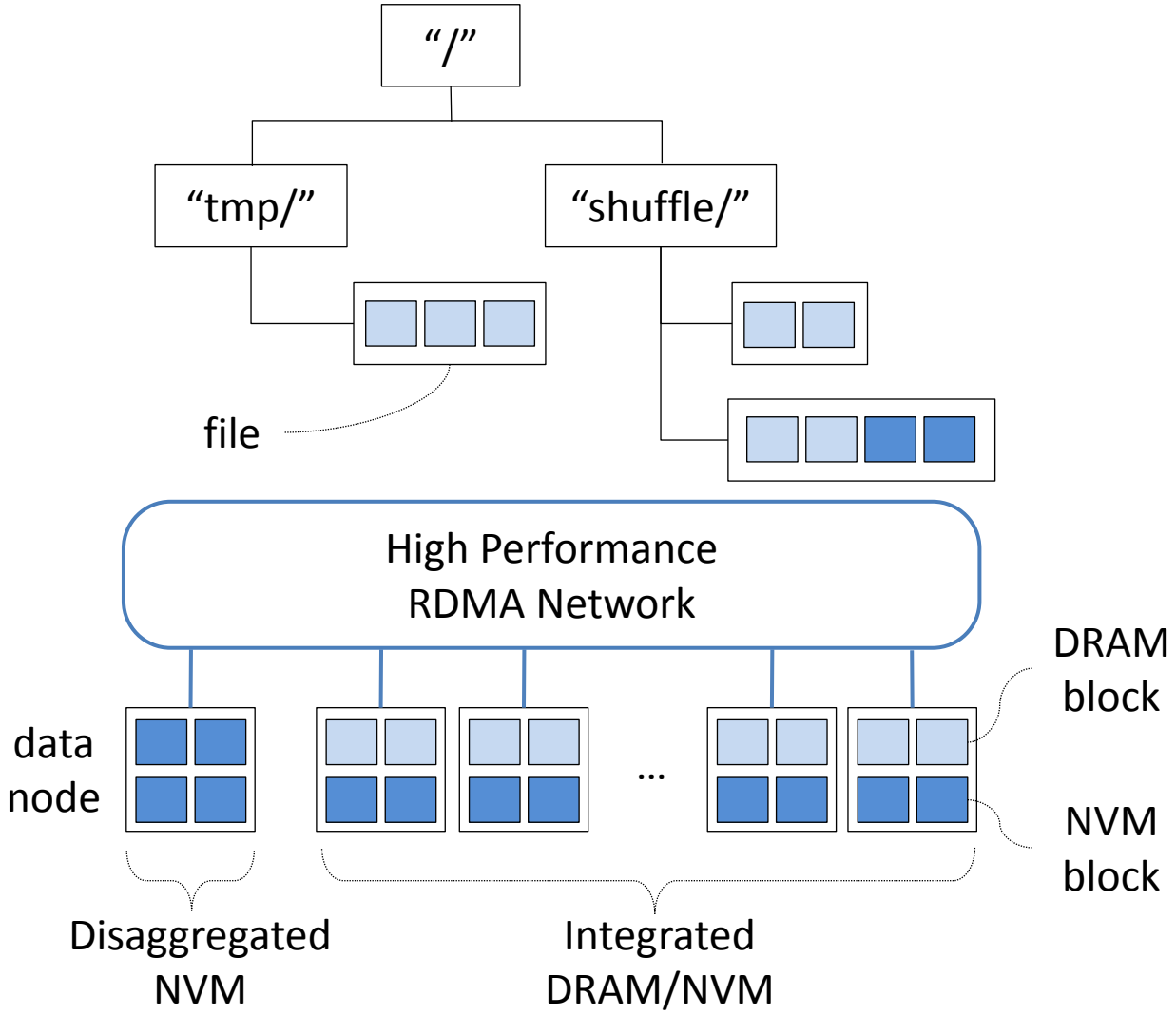
Crail

- ✓ **Architecture to deliver the performance of high-end storage and networking hardware to the application**
- ✓ **Relies on the principles of user level IO**
  - ✓ Separation between control path and data path
  - ✓ User-space direct-access I/O architecture/layer cut-through
  - ✓ Based on open source DiSNI user level IO stack for JVM (aka jVerbs)
- ✓ **Builds on a distributed, shared data store**
- ✓ **Used by multiple nodes, jobs or stages in a job to store and share data**
- ✓ **Comes with two flavors of interfaces for integration**
  - ✓ Direct access to store
  - ✓ Data processing tailored plug-in modules
- ✓ **No changes to overall data processing framework**
- ✓ **Is optimized to serve short-lived working set data**



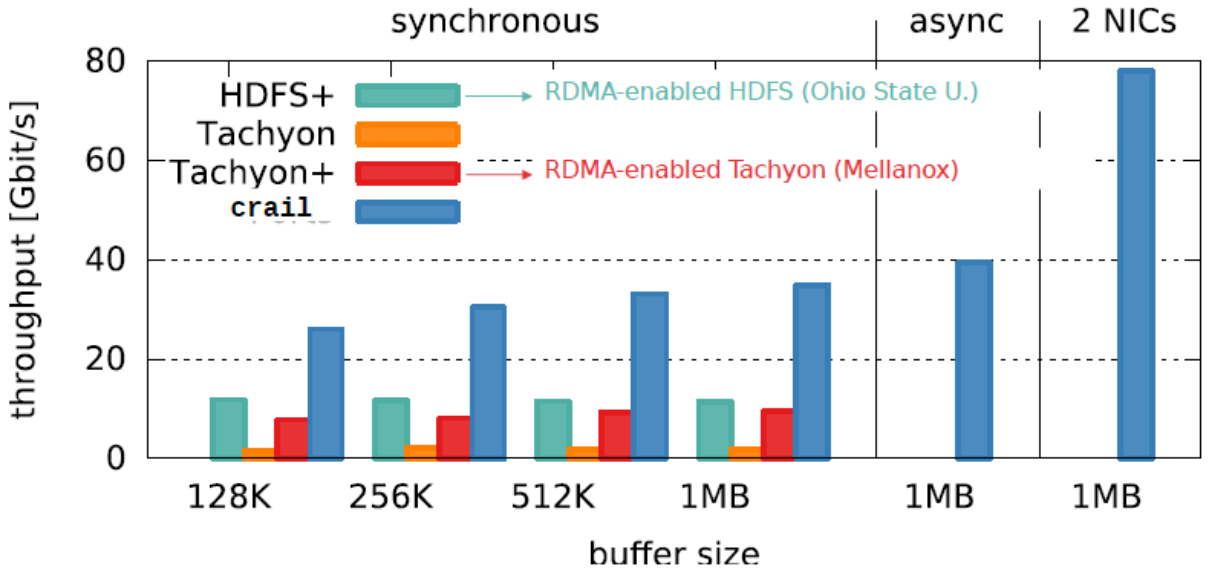
CRAIL STORE: A DISTRIBUTED FILE SYSTEM NAME SPACE

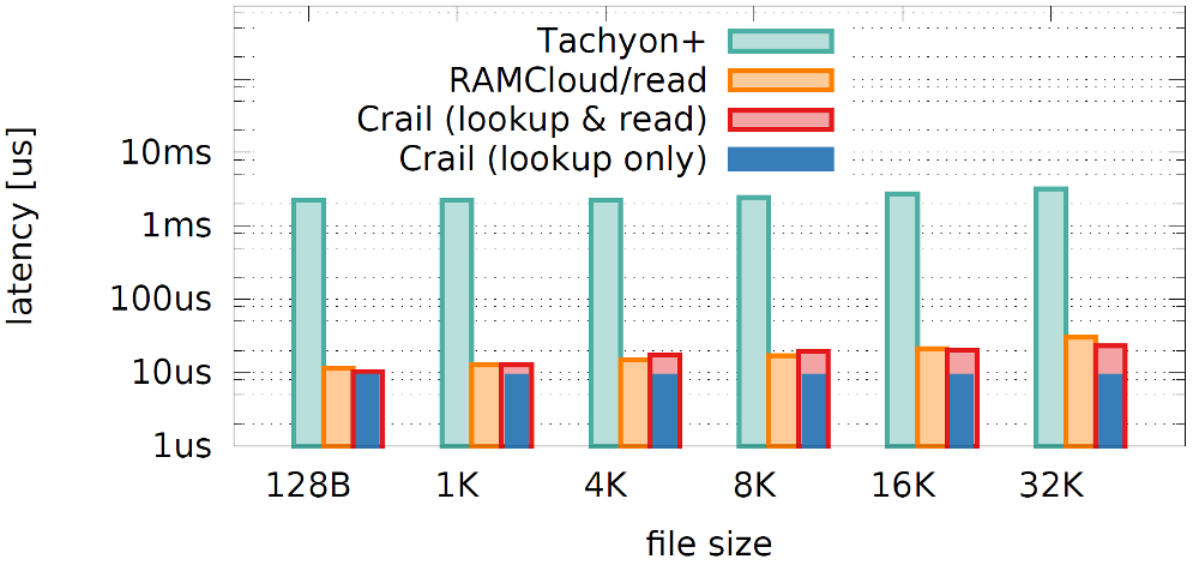
- **File system name space across cluster**
- **RDMA interconnect**
  - I/O buffers are pre-registered for RDMA
- **File access translates to asynchronous IO**
- **Multiple storage tiers**
  - DRAM
  - Shared volume (block device via SRP etc)
  - NVMeF
  - (more planned, such as GPU memory)
- **Store comprises compute, data, name node**
- **Data nodes may be**
  - Co-located with compute, or
  - Disaggregated
- **Data affinity control**
  - Location
  - Storage Tier



MICORBENCHMARKS: CRAIL STORE FILE ACCESS (READ)

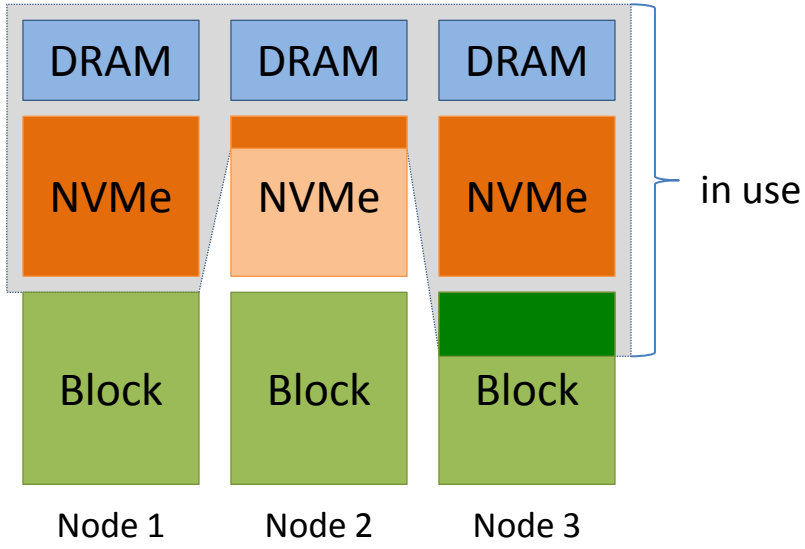
- Comparing with some high-performance in-memory tiers
- Crail file read performance bare metal
- Crail file access latencies among 'best of breed'

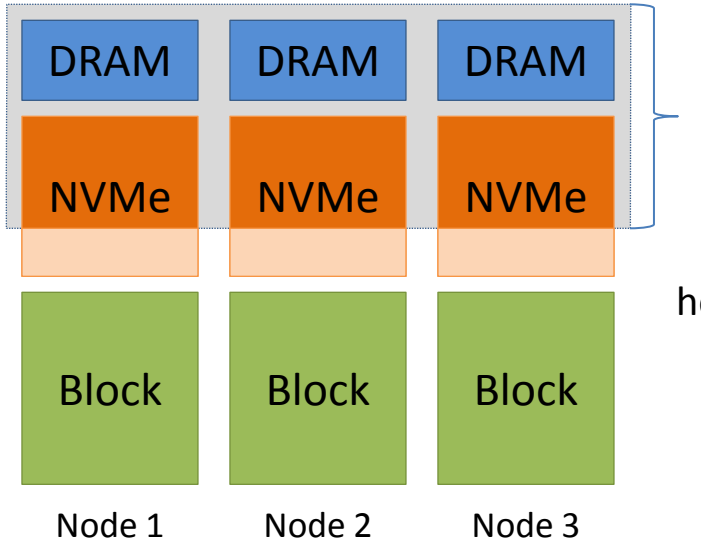




CRAIL STORE: STORAGE TIERING

Traditional
per node vertical tiering
(such as Tachyon)





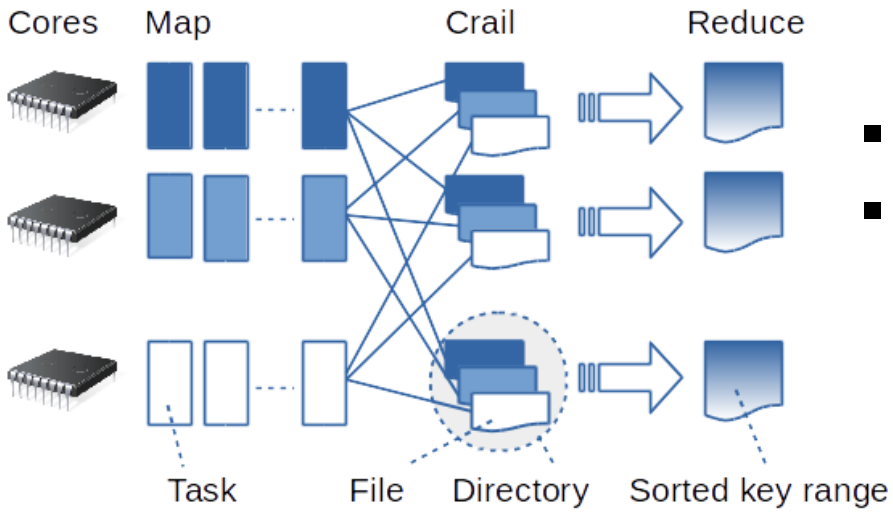
Crail
horizontal per resource
type tiering

- **Crail supports multiple storage tiers: memory, various types of Flash, disk**
- **Multiple tiers can be aggregated into a single address space, and**
- **Higher performance tiers are filled up across the cluster prior to using lower tiers**
  - Assumption of 'local always faster than remote' is not true anymore
  - Priority strategy selectable (application may choose 'storage affinity')
  - E.g., if set to DRAM, NVM store only used if DRAM completely filled up (automatic spill over)

CRAIL PLUGIN MODULES CURRENTLY AVAILABLE

Shuffle

- Spark specific plugin
- Maps key ranges to Crail dir's
- Map task appends k/v pairs to files in matching key directory
- Selects storage affinity (DRAM) for best performance



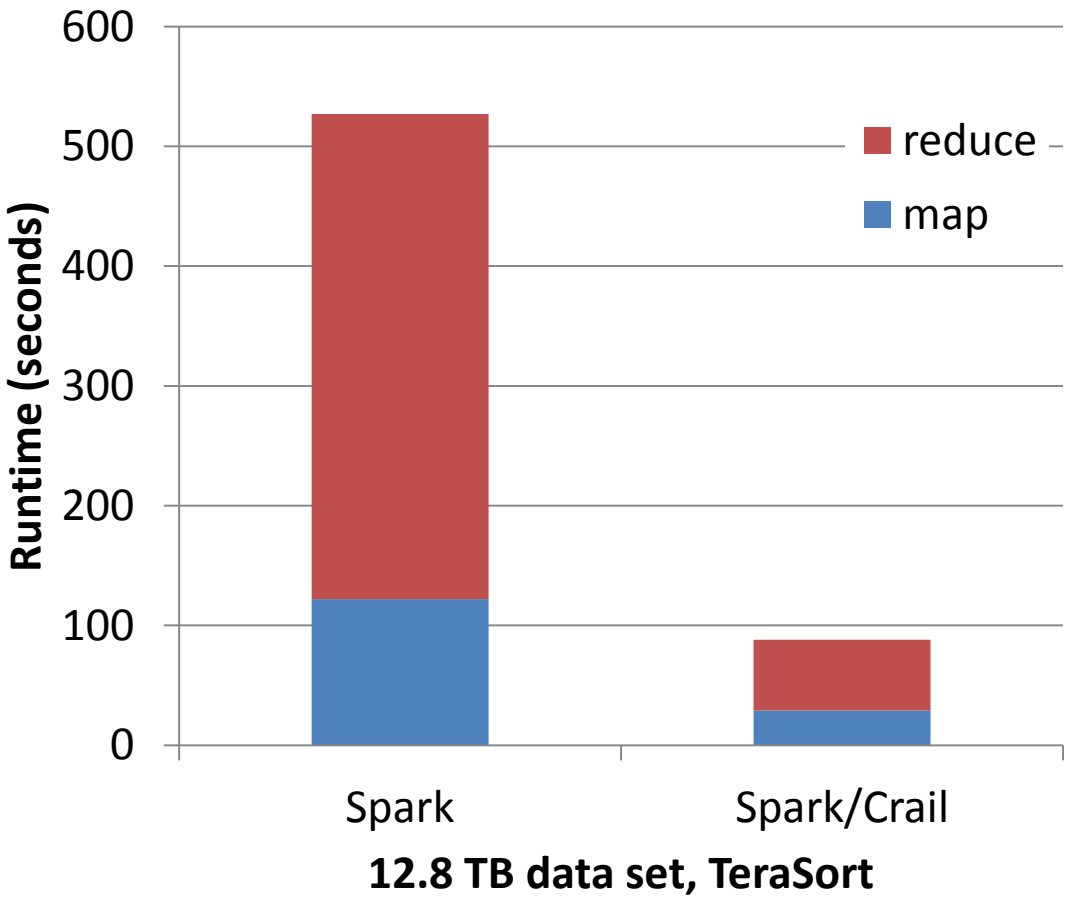
Broadcast

- Spark specific plugin
- Stores broadcast variables in Crail files
- Variables are inherently globally shared
- Unselects location affinity to ensure variables are equally distributed
- Very efficient in DRAM tier
- File write (~line speed) + 10µs publish + file read (~line speed)

HDFS Adaptor

- Generic plugin
- Exports HDFS file system
- Mapping between sync. HDFS calls and async. Crail operations
- Crail specific extra functionality exposed via admin tools per directory
  - Storage affinity, etc.
- Available for all Big Data frameworks using HDFS

EVALUATION - TERASORT



12.8 TB data set, TeraSort

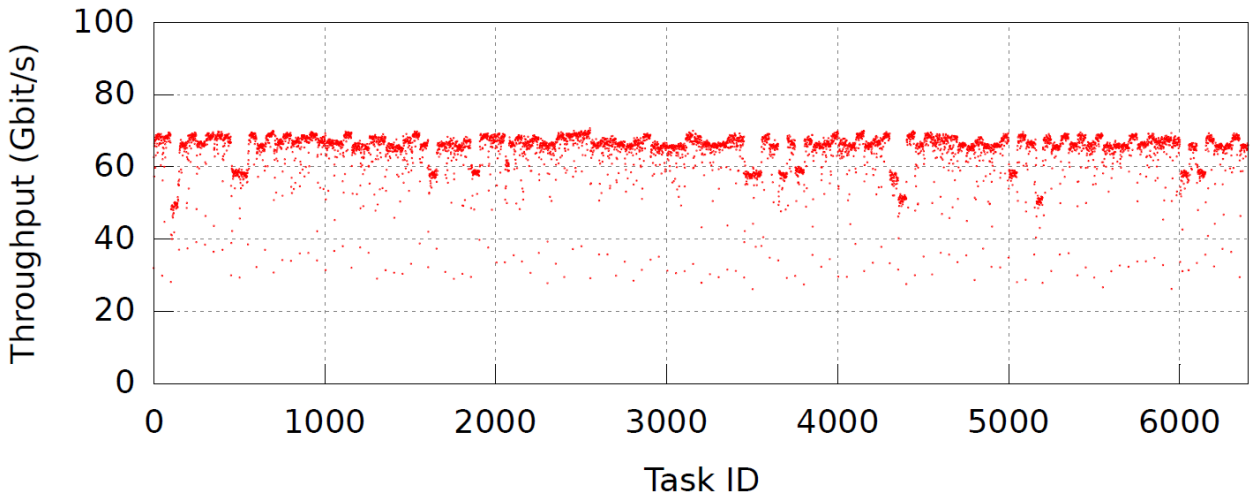
128 nodes OpenPOWER cluster

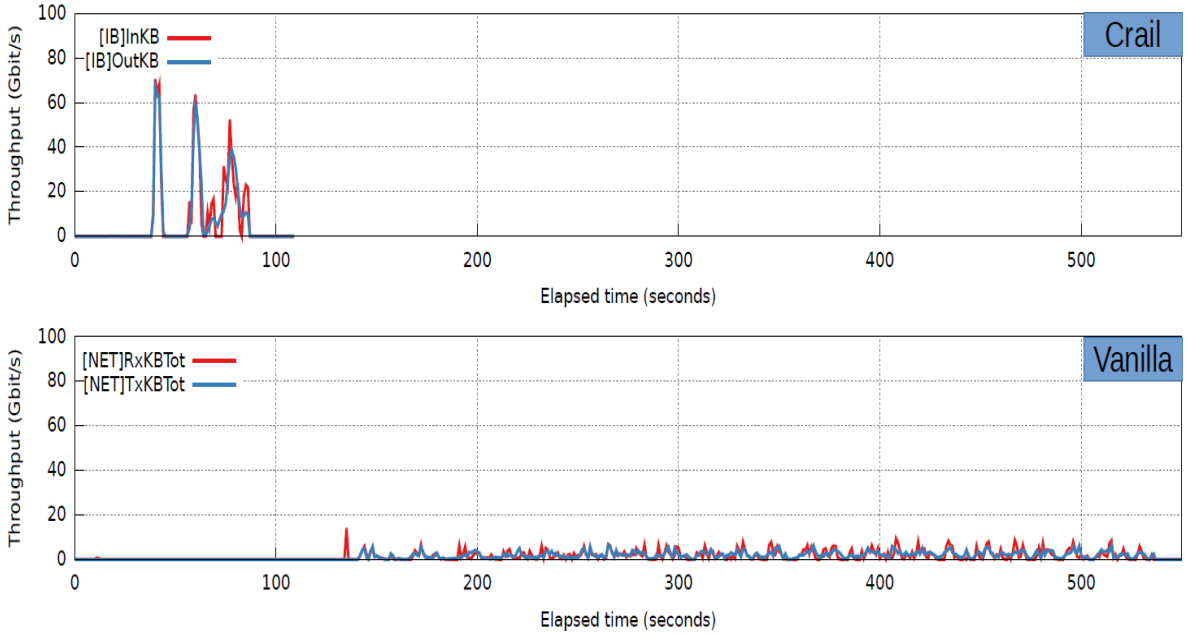
- 2 x IBM POWER8 10-core @ 2.9 Ghz
- DRAM: 512GB DDR4
- 4 x 1.2 TB NVMe SSD
- 100GbE Mellanox ConnectX-4 EN (RoCE)
- Ubuntu 16.04 (kernel 4.4.0-31)
- Spark 2.0.2

Performance gain: 6x

- Most gain from reduce phase:
  - Crail shuffler much faster than Spark build-in
  - Dramatically reduced CPU involvement
  - Dramatically improved network usage
- Map phase: almost all activity local
  - Still faster than vanilla Spark

EVALUATION – TERASORT: NETWORK IO

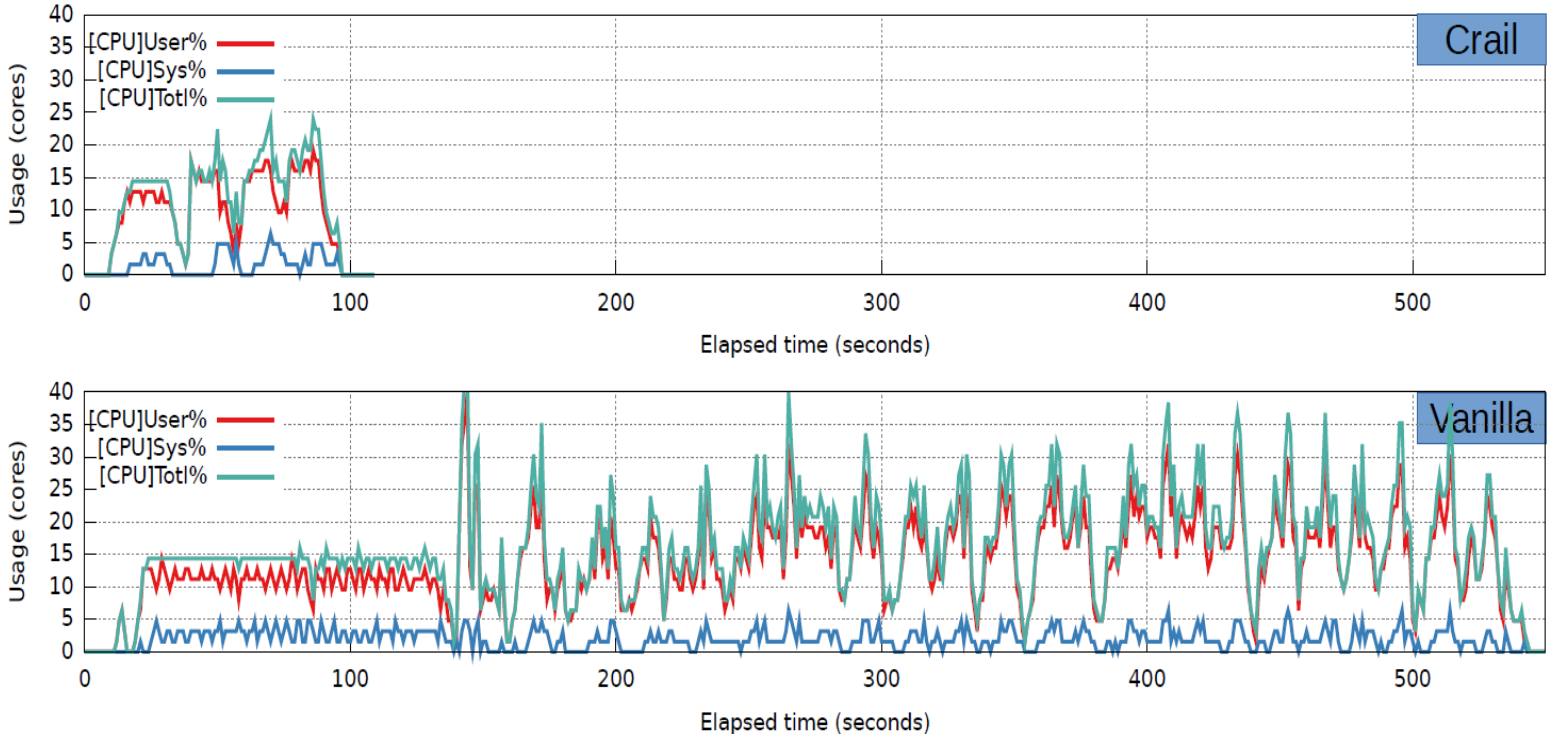




- Vanilla Spark runs on 100GbE
- Spark/Crail runs on 100Gb RoCE/RDMA

- Vanilla Spark peaks at ~10Gb/s
- Spark/Crail shuffle delivers ~70Gb/s per node

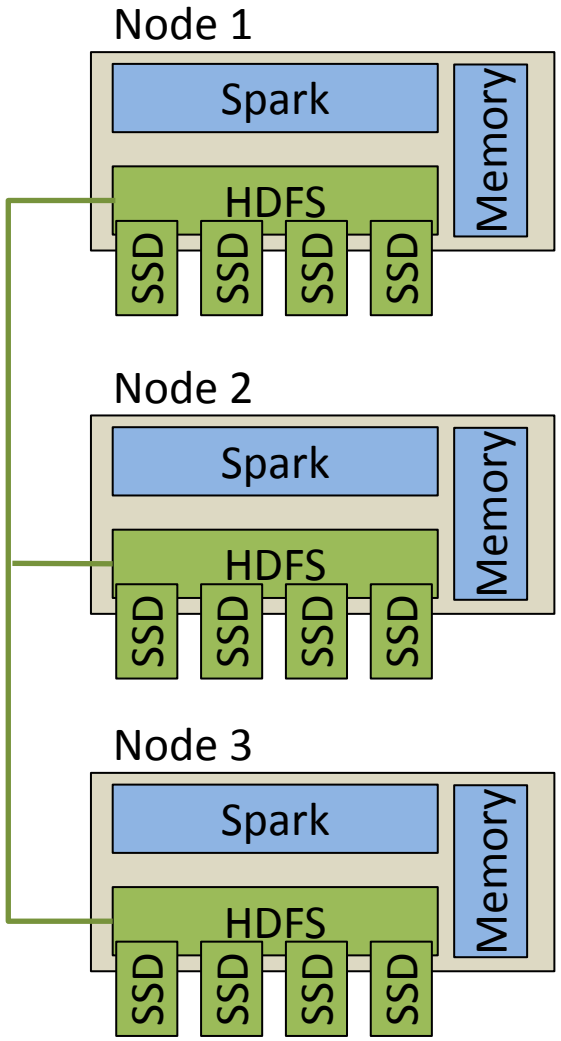
EVALUATION – TERASORT
CPU EFFICIENCY



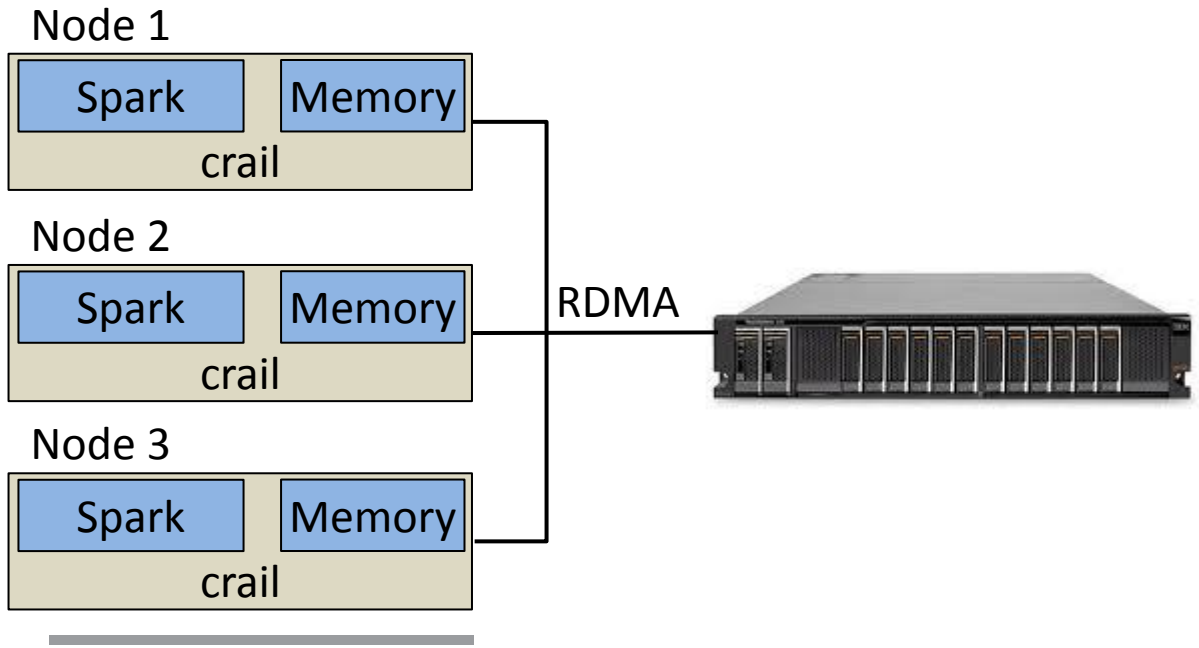
	Spark + Crail	Spark 2.0.2	Winner 2014	Winner 2016
Size TB	12.8		100	
Time sec	98	527	1406	98.6
Cores	2560		6592	10240
Nodes	128		206	512
NW Gb/s	100		10	100
Rate TB/min	7.8	1.4	4.27	44.78
Rate/core GB/min	3.13	0.58	0.66	4.4

- Spark/Crail completes much faster despite comparable CPU load
- Spark/Crail CPU efficiency is close to 2016 sorting benchmark winner: **3.13 vs. 4.4 GB/min/core**
- 2016 winner runs native C code!

EVALUATION – STORAGE DISAGGREGATION SUPPORT



- **Why storage disaggregation?**
  - Independent scaling of compute and storage
  - Higher utilization due to less fragmentation
  - Easier maintenance
- **Challenge**
  - Systems like Hadoop/Spark have been designed for local storage!



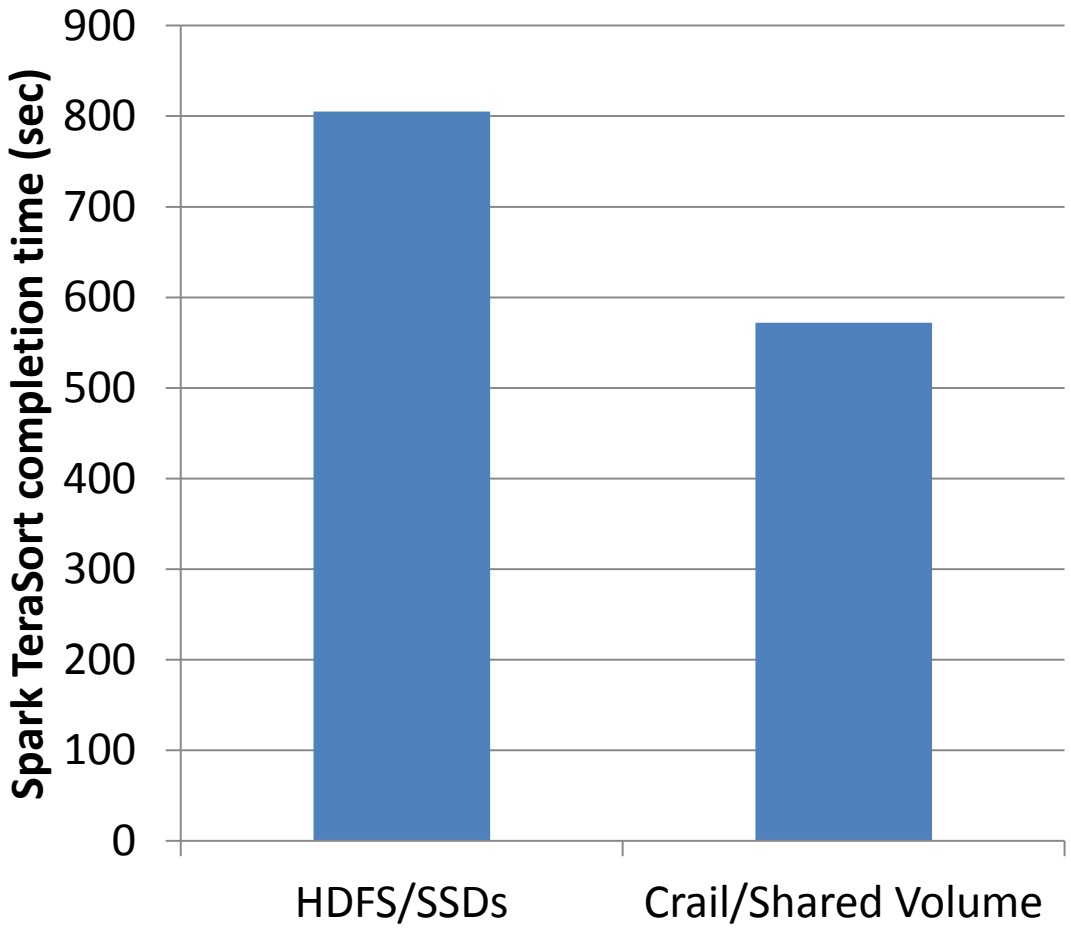
EVALUATION – STORAGE DISAGGREGATION

Experiment:

- Sorting 400GB of data
- 10 nodes cluster
- 56 Gb/s InfiniBand network
- HDFS: local attached flash
  2 x 1TB SSD/node, 2-way replication
  uses host memory (via OS page cache)
- Crail: SRP attached flash
  1 x FlashSystem 840 (8 cards, 23TB use)
  does not use host memory
- About the same Flash BW and capacity

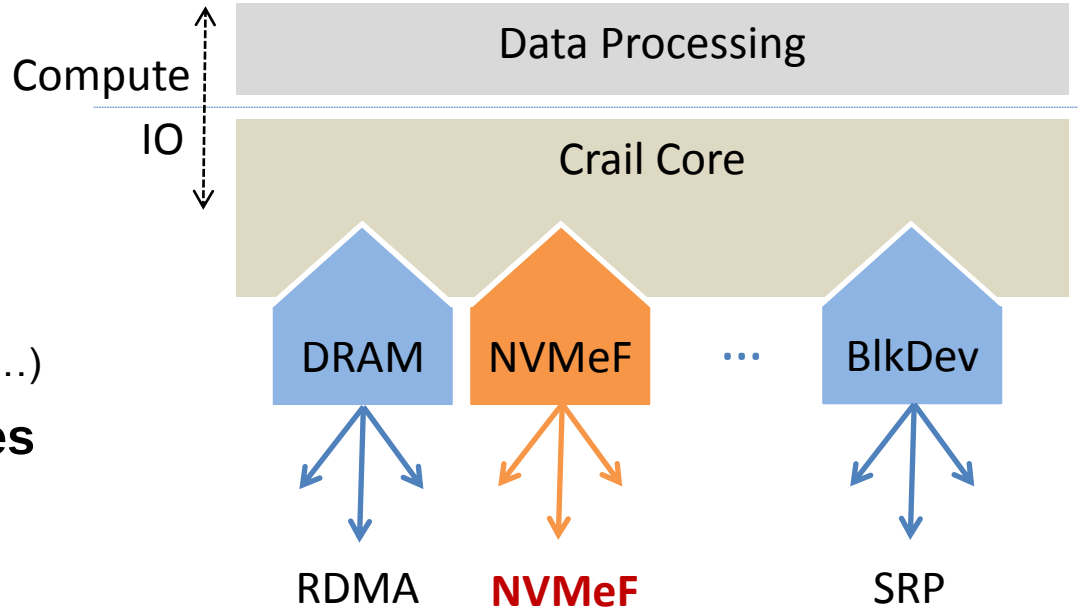
Result:

- Crail + disaggregated flash achieves over 20% speedup
- More efficient to access remote flash in Crail than local flash in Spark!

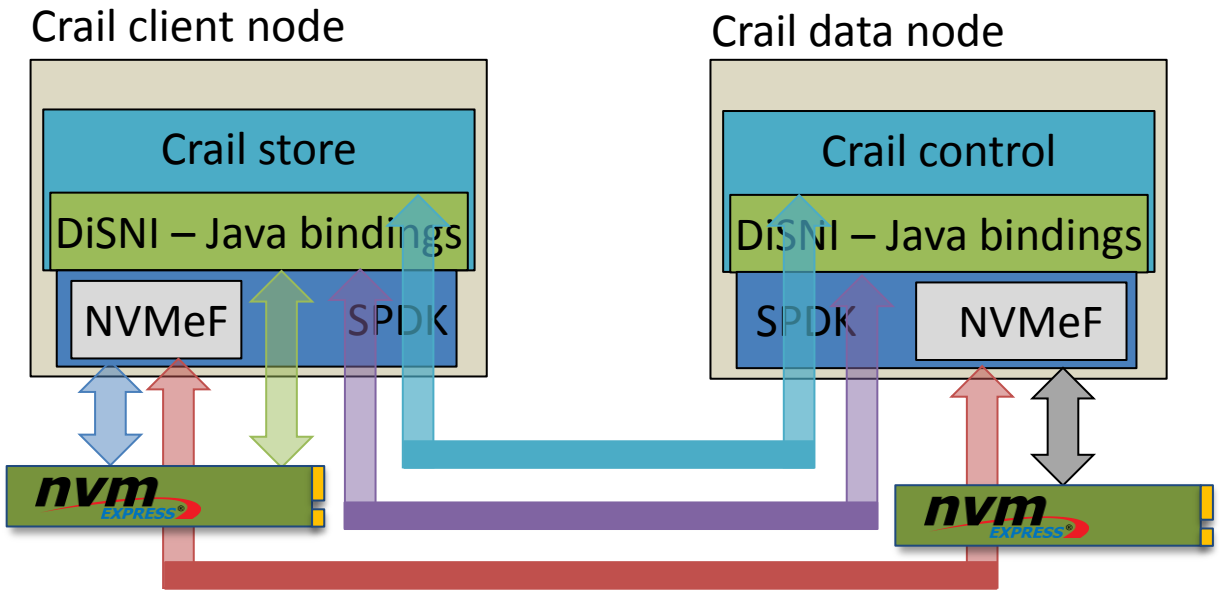


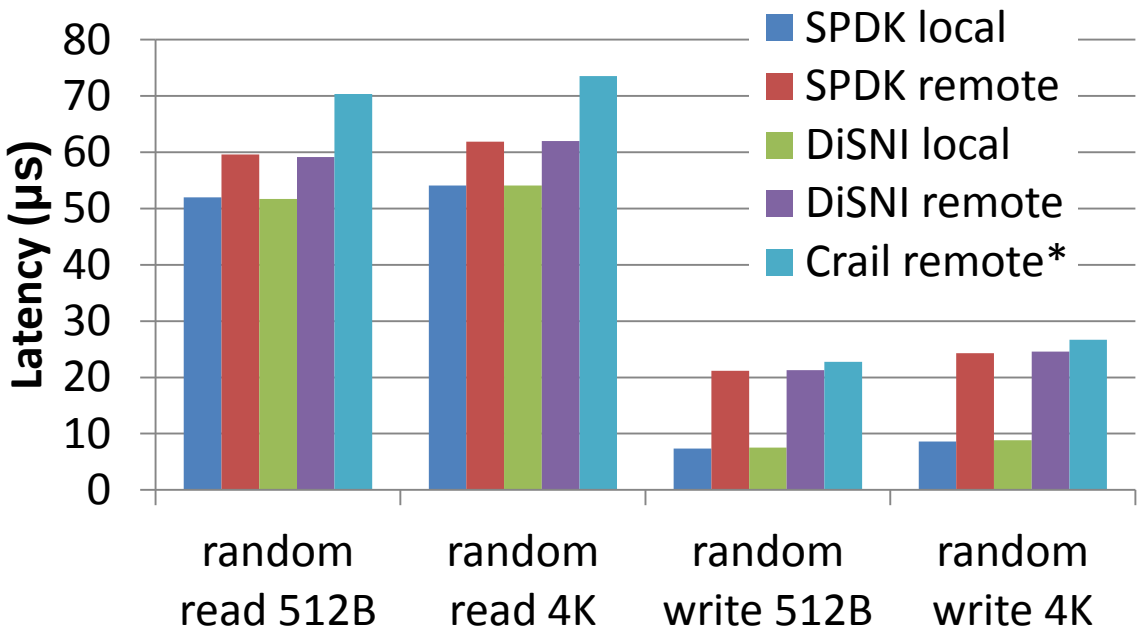
ADDING THE CRAIL NVMEF TIER

- **NVMeF technology maturity**
  - Adds little latency to local NVMe device access
  - Promising technology for storage disaggregation
- **Adding a new (NVMeF) tier to Crail:**
  - Storage tiers are just plugin's
  - Implementing some abstract functions of a shared data node class (init, createEndpoint, run, close..) and an endpoint class (read, write, close, …)
- **Crail NVMeF client and target code uses SPDK libraries**
  - JNI wrapper for SPDK control and data transfer functions
  - Integrated with DiSNI (former 'jVerbs') IO layer within Crail
- **Crail NVMeF data node**
  - Calls into SPDK NVMeF target code
- **Crail NVMeF client**
  - Calls into SPDK NVMeF initiator code
  - Unfortunately, no direct buffer passing possible (yet)
    - Data copy within SPDK library
  - Unfortunately, no clear split with DPDK library (always carried around, but not fully needed here)
    - Changes were needed to allow for non-root usage



NVMEF TIER PROTOTYPE RESULTS: ACCESS LATENCY





Setup

- Samsung 960 Pro
- Chelsio T5 40Gb
- SPDK git master early March'17
- Investigating local and remote NVMeF access
- QD: 1 (latency), 128 (BW)

Results

- DiSNI/Java adds very little to plain SPDK
  - True for local and remote
- Crail store adds some 10µs to native latency
- Crail file read/write to NVM is close to bare metal IO
- Bandwidth always at device limit (not shown)

NVMEF TIER PROTOTYPE RESULTS: TERASORT

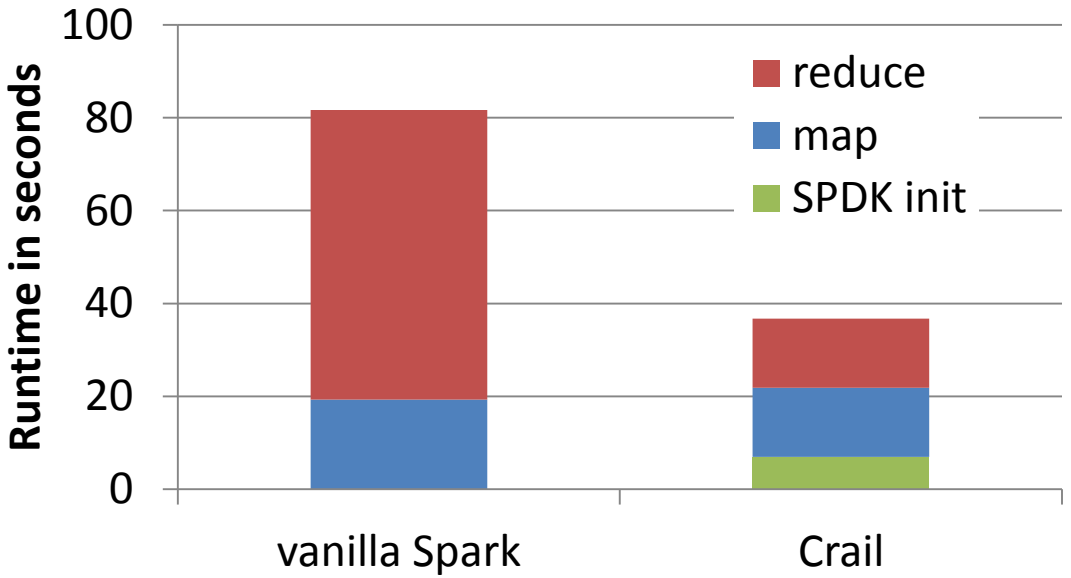
TeraSort run for 200GB

Setup:

- 10 machines
- NVMe: 2 x 960 Pro 512GB per machine, ext4 fmt.
- Spark/Yarn: 10 executors, 8 cores each
  →80 tasks parallel

Run:

- Spark
  - Input/output to NVMeF (HDFS)
  - Shuffle to tmpfs (DRAM, basically)
  - Best result found at partition size of 64MB
- Crail
  - Input/output to Crail NVMeF tier
  - Shuffle to Crail DRAM tier
  - Best result found at partition size of 256MB



Results

- Crail beats vanilla Spark by more than 50%
- Why only reduce phase contributes to perf. gain?

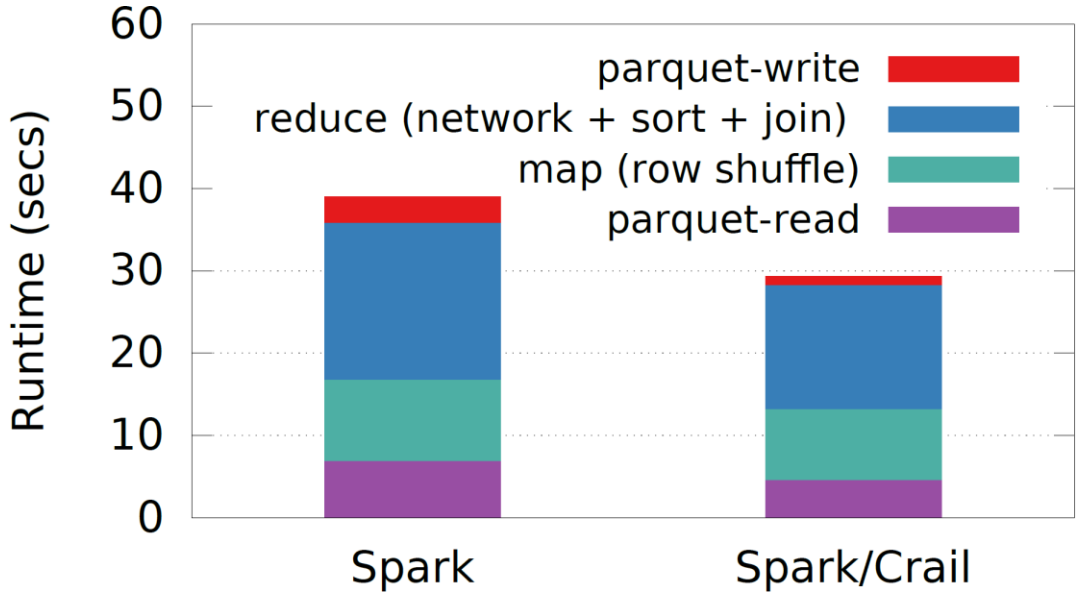
→`SPDK initialization hurts Crail

- 7 seconds to set up/register buffers etc.
- Includes DPDK initialization

EVALUATION - SQL

- **Spark/SQL evolved into fast, powerful SQL system**
- **Experiment on 8 nodes:**
  - Compare performance of SQL join operation of two large tables
    - Cartesian product of two RDDs
    - Re-arrange RDDs partitions, execute predicate on desired columns
  - Experimental join operation: EquiJoin
    - Matching rows based on same values
    - Implemented as MapReduce job
  - Two Parquet data sets joined, 128GB from 128 million rows each
  - Vanilla Spark vs Crail + Crail Shuffle
  - Parquet files stored in HDFS or Crail/HDFS
  - Map: Tables read from distributed store and key ranges written to worker nodes
  - Reduce: Each worker collects two sets of rows, scans and joins, and writes out matching rows

Performance of an SQL join



- 25% performance gain with simple workload
- Gains from all four IO phases
- Reduce gains overshadowed by heavy local de-serialization, sorting and join

CONCLUSION

Today's open source analytics stacks

- Existing analytics stacks designed for yesterday's commodity hardware
- Performance on high-end hardware inhibited by heavy-layered stack architecture

The Crail approach

- Radical re-design of I/O (network & storage) for analytics by exploiting modern hardware
- RDMA, NVMe & NVMe over fabrics
- Extends Spark operation by plugins to take advantage of Crail
- Enable high-performance disaggregated storage for analytics
- Crail reduces Time-to-Insight and cost/performance
- Crail is open source: **www.crail.io**
- Stay tuned: Crail will get presented at 2017 Spark Summit this June

DISCUSSION

- **Crail is not "tied" to Spark**
  - Independent of the compute framework
  - The focus is on high-performance data sharing
- **Crail is not about "just" putting RDMA in Spark**
  - Focus is on the high-performance, user-level IO in a distributed system
  - Expands beyond RDMA to storage (e.g., NVMeF)
  - Even allows to execute on non-RDMA networks – TCP/IP tier available
- **Crail is not just an academic prototype**
  - Tried and tested on 100+ nodes, multiple architectures, and RDMA networks
  - Independent modules (tiering, RPC, buffer management, scheduling, etc.)
  - Active development targeting GPUs, FPGAs, and diverse workloads
  - Check out www.crail.io (all sources are available at github.com)

OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

THANK YOU

Bernard Metzler

IBM Zurich Research

[LOGO HERE]