Handling 100 Gb/s RDMA, and NVMe in Apache Crail and Pocket

Patrick Stuedi

Hardware Changes since 2010

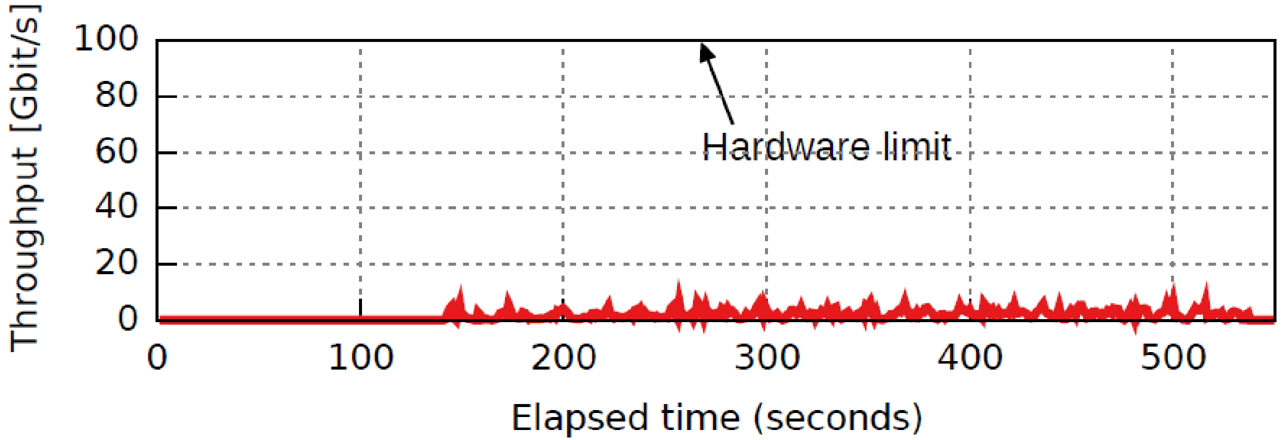
~ 2014: starting Crail project

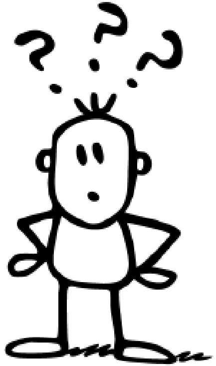
	2010	2015	2020	
Storage	50 MB/s (HDD)	500 MB/s (SSD)	16 GB/s (NVMe)	10x
Network	1 Gb/s	10 Gb/s	100 Gb/s	10x
CPU	~3 GHz	~3 GHz	~GHz	

Reynold keynote, https://databricks.com/session_na20/wednesday-morning-keynotes

Challenges

- Difficult to leverage modern networking and storage hardware
- Example (2016): sorting 12 TB on a 128 node cluster, all data in DRAM, 100 Gb/s full bisection network

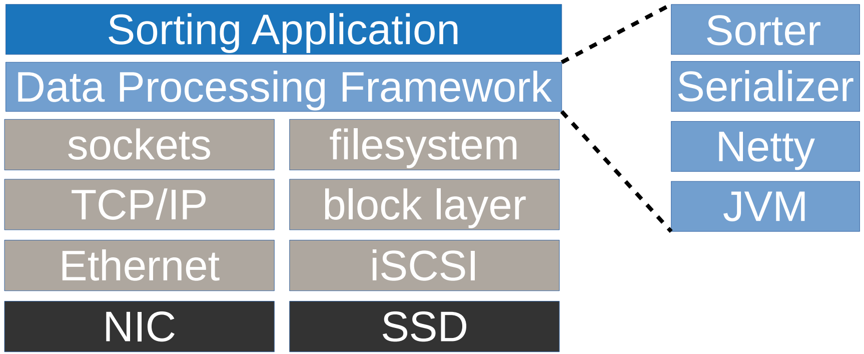


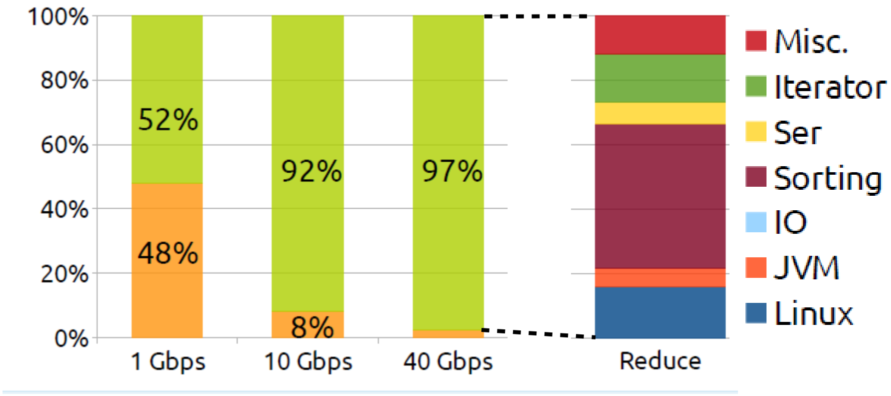


Software Overheads

	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495

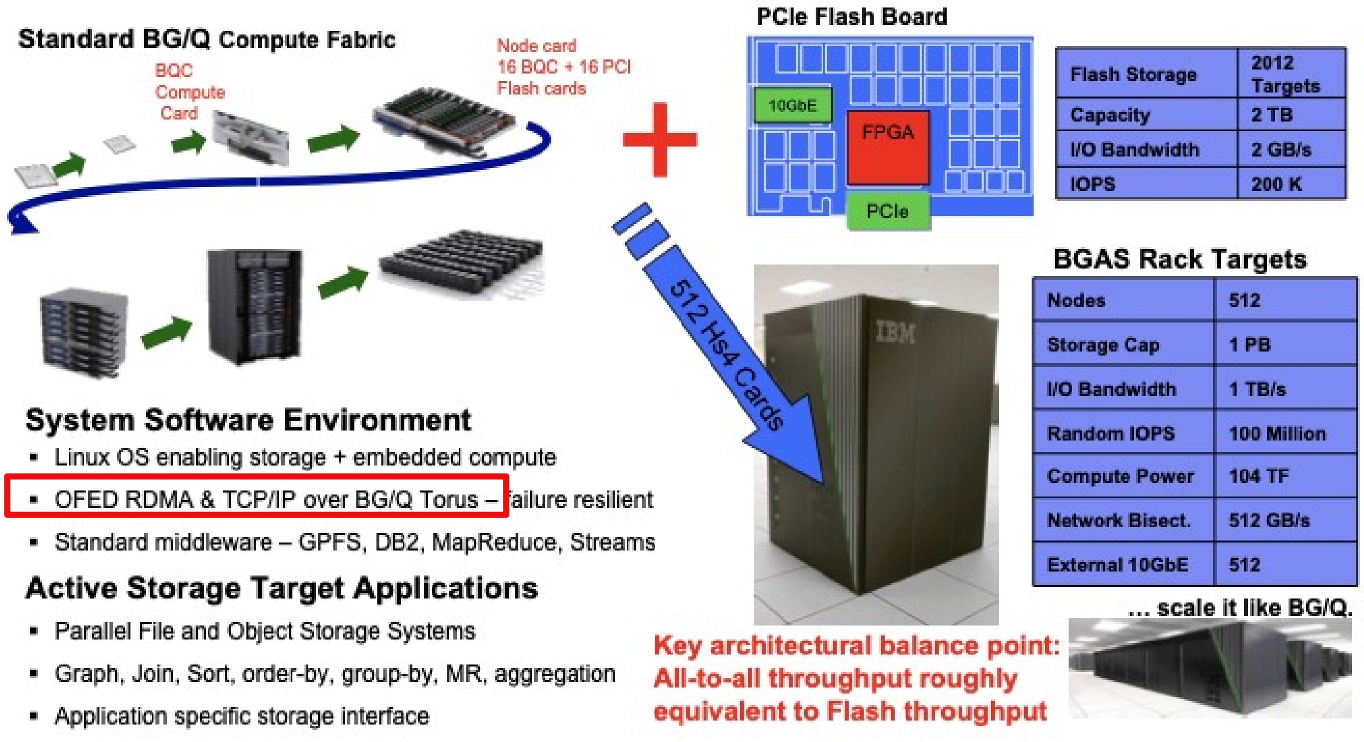
software overhead are spread over the entire stack





HotNets'16

How do Supercomputers solve this?



IBM BlueGene Active Store Project (2012)

RDMA on Azure

Azure VM Types

	General Purpose	Compute Optimized	Memory Optimized	Storage Optimized	GPU	High Performance Compute
Type	Av2, B, DCsv2, Dv2, Dsv2, Dv3, Dsv3, Dav4, Dasv4, Ddv4, Ddsv4,Dv4, Dsv4	Fsv2	M, Mv2, Dv2, DSv2, Ev3, Esv3, Eav4, Easv4, Ev4, Esv4, Edv4, Edsv4	Lsv2	NC, NCv2, NCv3, ND, NDv2, NV, NVv3, NVv4	H, HBv2, HC, HB
Description	Balanced CPU and memory	High ratio of compute to memory	High ratio of memory to compute	High disk throughput and IO	Specialized with single or multiple NVIDIA GPUs	High memory and compute power – fastest and most powerful
Uses	Testing and development, small-medium databases, low-medium traffic web servers	Medium traffic web servers, network appliances, batch processing, app servers	Relational database services, analytics, larger caches	Big Data, SQL, NoSQL databases	Compute intensive, graphics-intensive, visualization workloads	Batch processing, analytics, molecular modeling, fluid dynamics, low latency RDMA networking

Microsoft Azure
Linux RDMA

RDMA Networking

- User-level network architecture
- Kernel bypass
  - NIC queues accessible from user-space
- Transport stack offloading
  - Infiniband, RoCE, iWARP

RDMA Networking: Benefits

- User-level network architecture
- Kernel bypass
  - NIC queues accessible from user-space
- Transport stack offloading
  - Infiniband, RoCE, iWARP

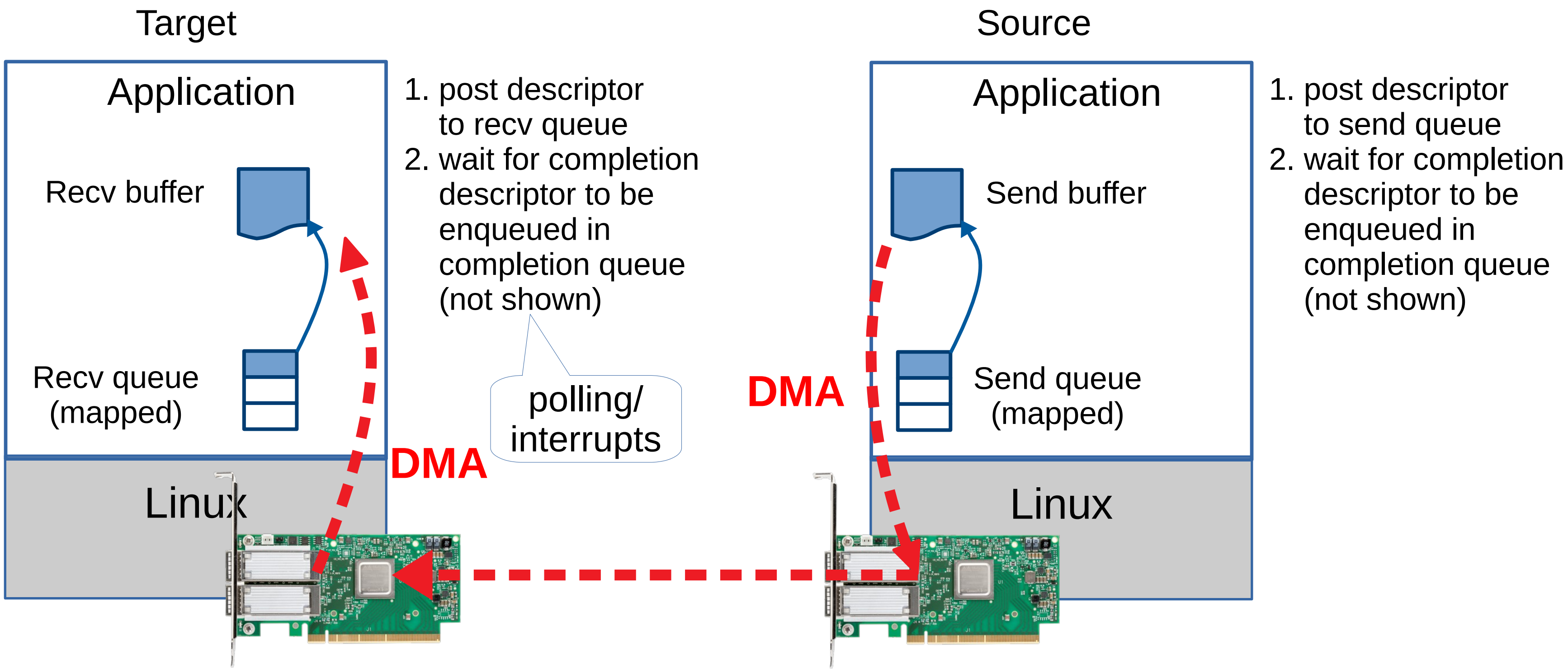
Low latency
  no sycalls
Zero-copy
  directly DMA from/to
  userspace buffers
Low CPU usage
  transport offloading
High bandwidth

RDMA Networking: Benefits

- User-level network architecture
- Kernel bypass
  - NIC queues accessible from user-space
- Transport stack offloading
  - Infiniband, RoCE, iWARP

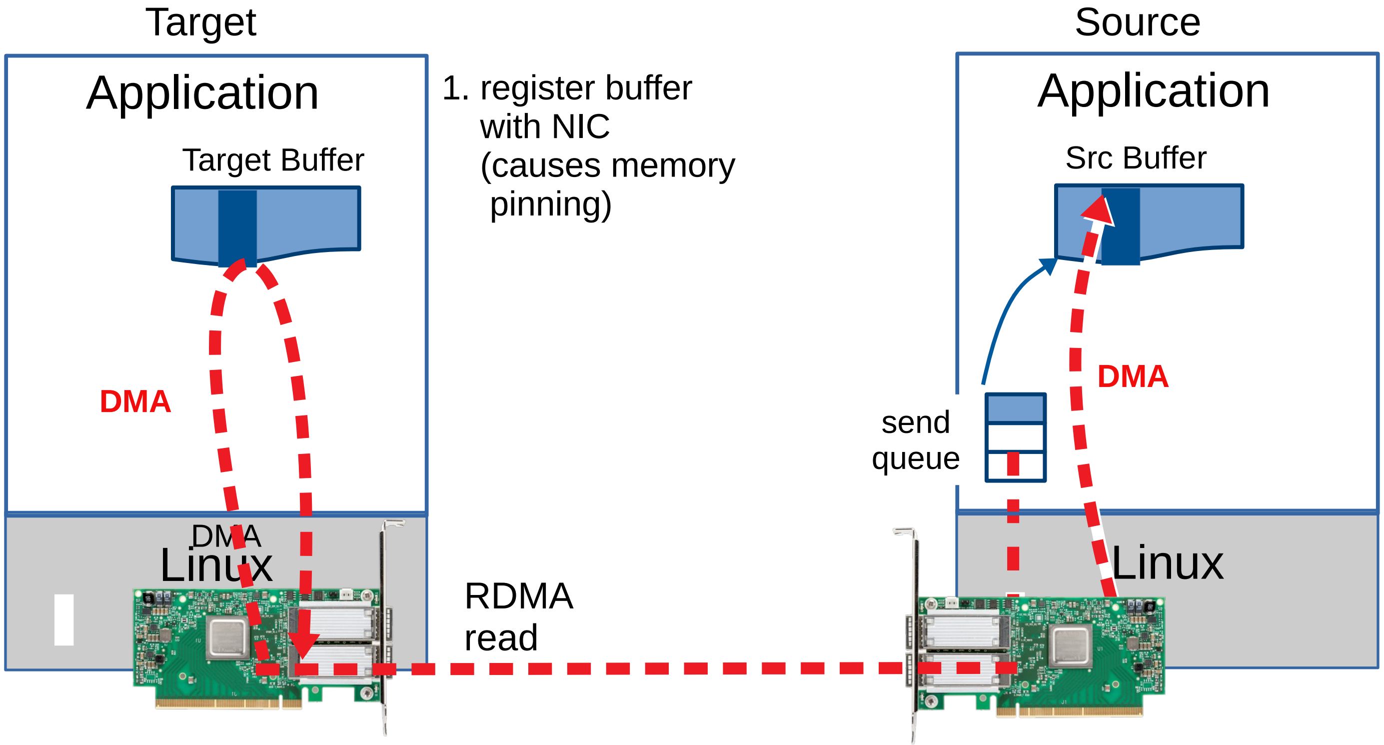
Low latency
    no sycalls
Zero-copy
    directly DMA from/to
    userspace buffers
Low CPU usage
    transport offloading
~~High bandwidth~~
High bandwidth / core

RDMA Two-Sided Operations



Source does not need to know buffer address at target. Receiver is notified.

RDMA One-Sided Operations



1. register buffer
   with NIC
   (causes memory
   pinning)

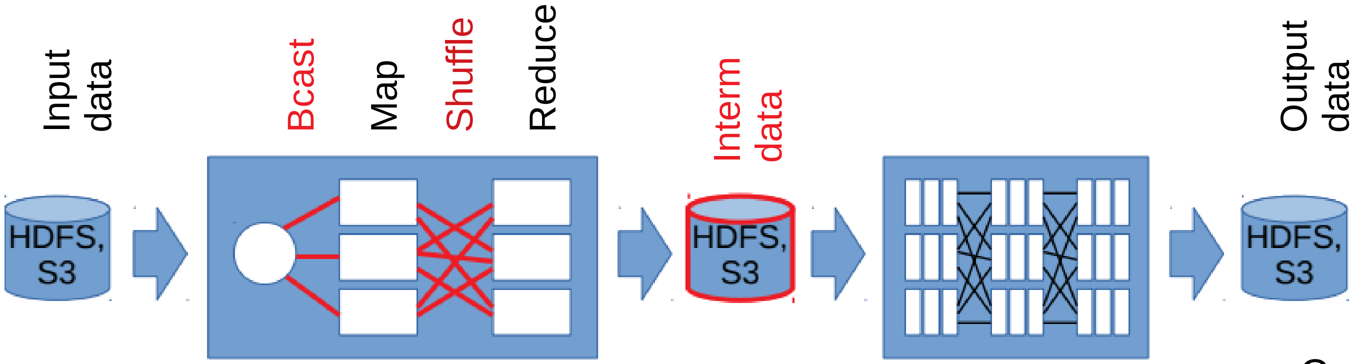
1. post read operation descriptor to send queue
2. wait for completion descriptor to be enqueued in completion queue (not shown)

Source needs to know target address. Target is not notified.

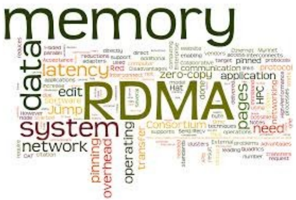
NVM Express (NVMe)

- Host-controller interface for PCI attached SSDs
- Enables user-level access for storage
  - Map device queues into user-space
  - SPDK, NVMe-over-Fabrics

Integrating User-level I/O with Data Processing Systems

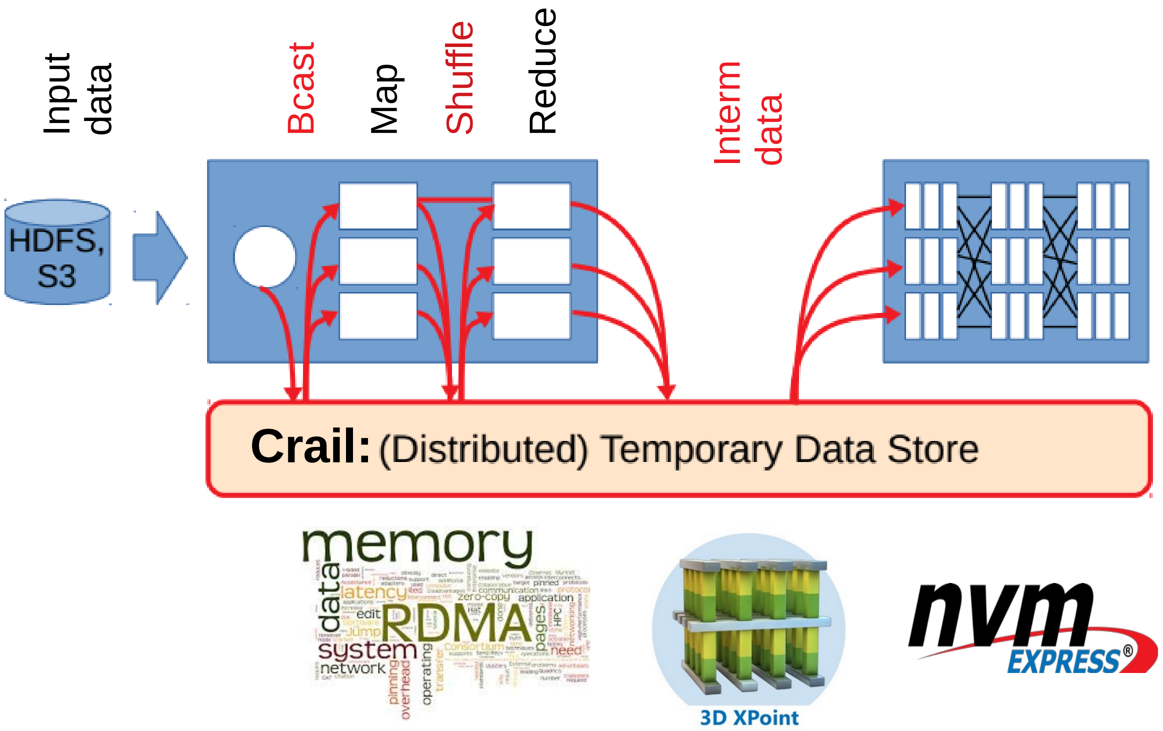


Can’t implement every operation for all the different hardware, framework and deployment options



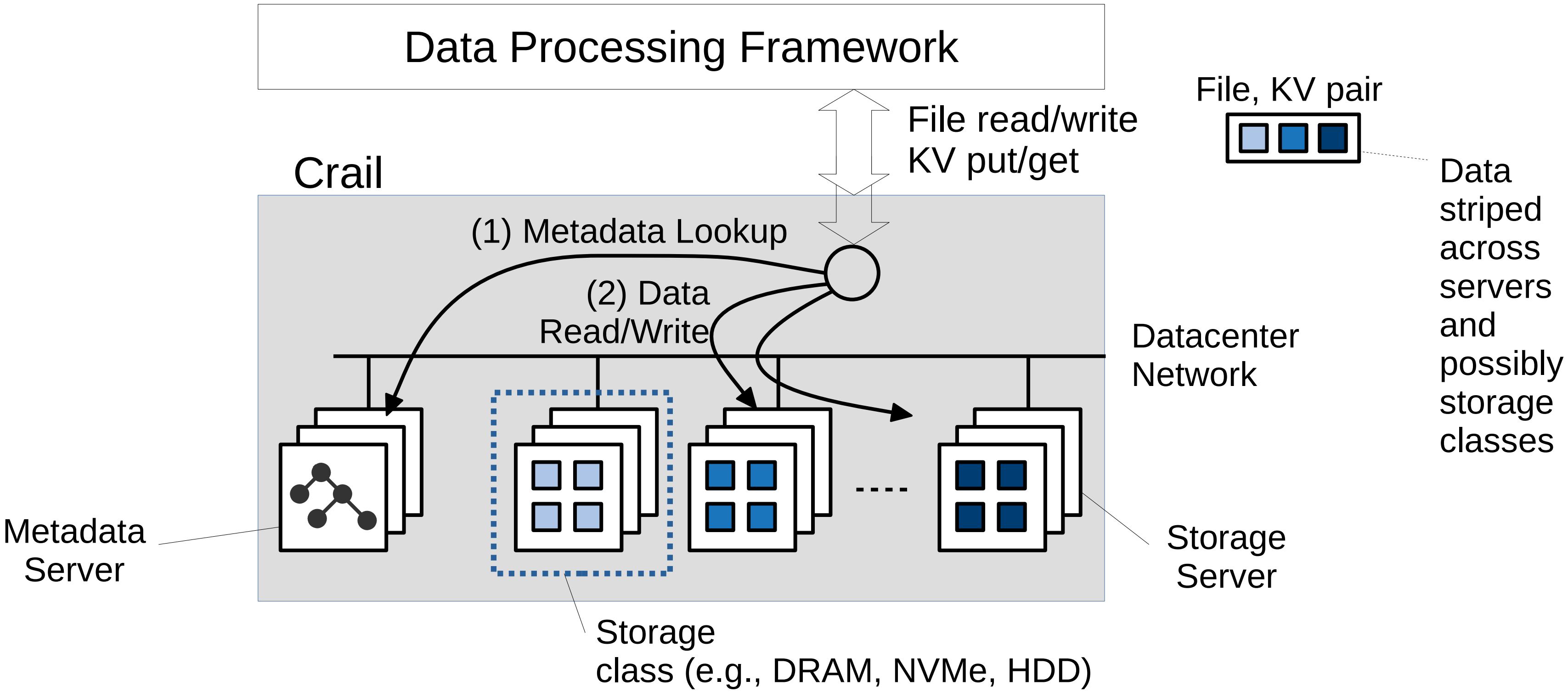
3D XPoint

Integrating User-level I/O with Data Processing Systems (2)



Implement hardware support once and support different operations and frameworks

Crail Architecture

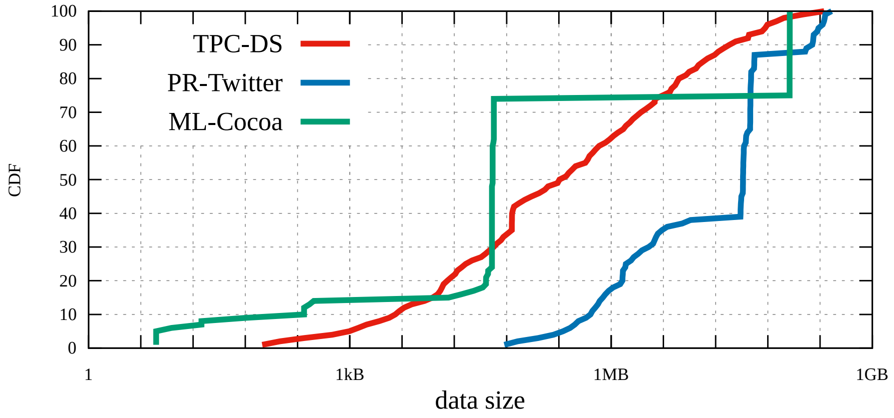


Performance Challenges

Performance Challenges

1. Must handle millions of storage operations per second on a large number files with a wide range of data sizes

   - Example: the Spark shuffle engine built on top of Crail creates #partition files per core in the cluster. With 128 machines each running 3 executors with 5 cores each that is 11M files (!)
   - Files have a wide range of data sizes



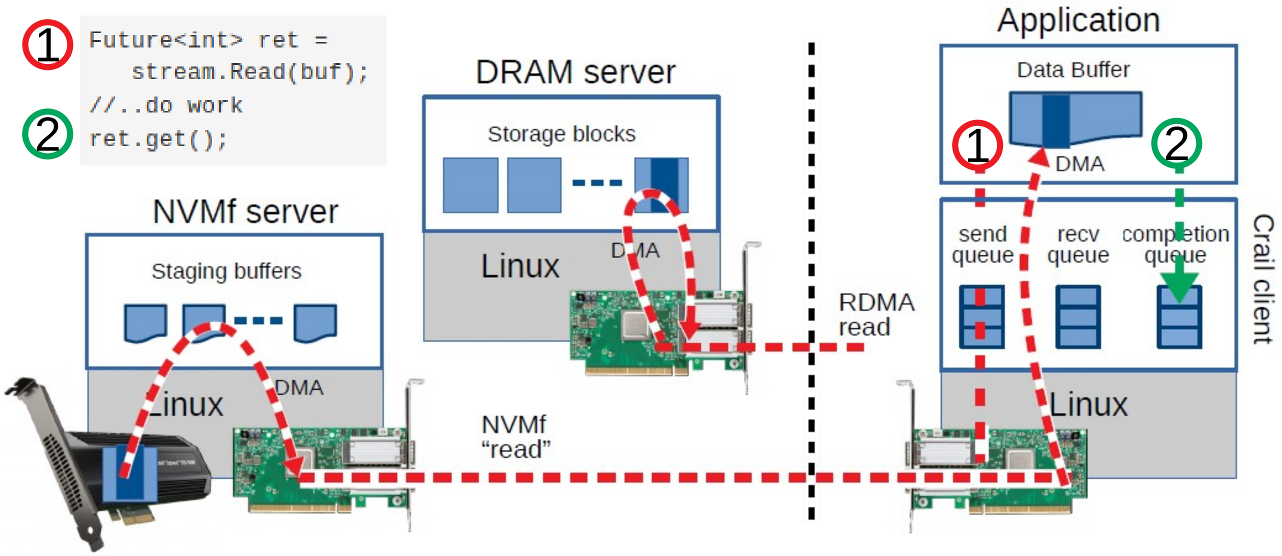
Performance Challenges

1. Must handle millions of storage operations per second on a large number files with a wide range of data sizes
   - Example: the Spark shuffle engine built on top of Crail creates #partition files per core in the cluster. With 128 machines each running 3 executors with 5 cores each that is 11M files (!)
   - Files have a wide range of data sizes
2. Should be able to read/write at line speed (e.g., 100 Gb/s) using a single core (for a reasonable I/O size)
3. Must support reading/writing of tiny files in a few microseconds
4. Overall CPU consumption of the storage system should be kept low
5. Must be able to store data volumes > cluster DRAM

Performance/Design Decisions

Performance/Design Decisions

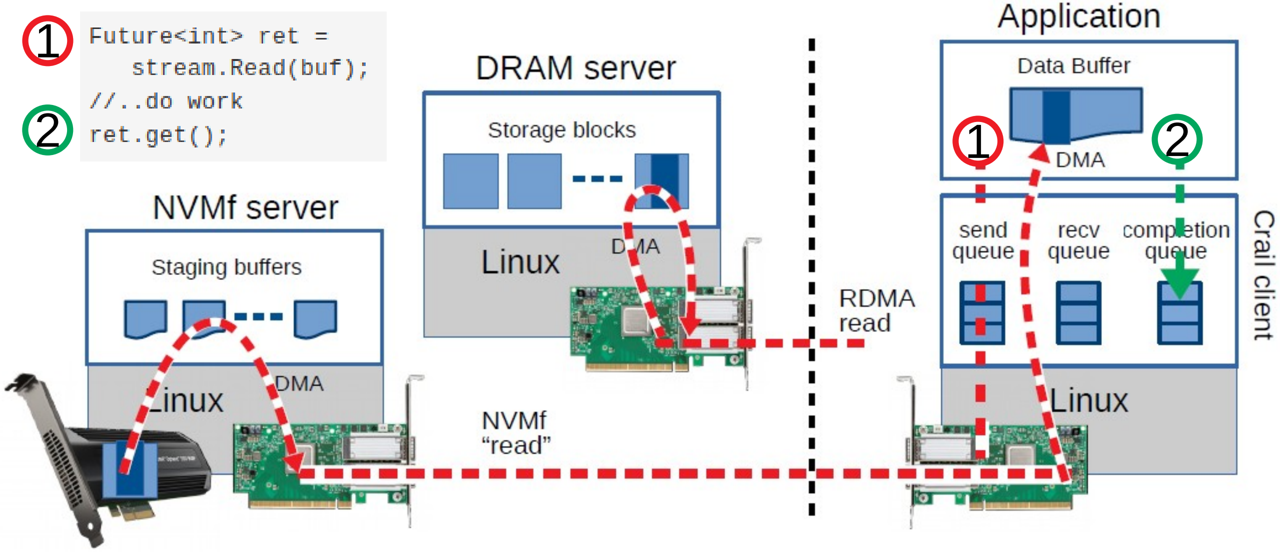
1. Fast data path using one-sided RDMA
   - Metadata needs to include target address for read/write (not shown)
   - Have the NIC DMA to/from actual application buffers



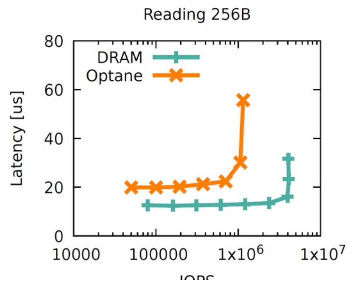
Performance/Design Decisions

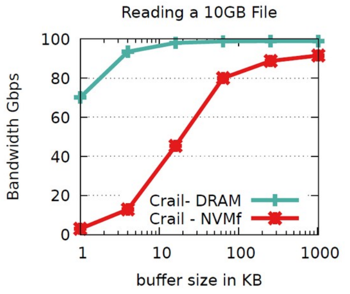
1. Fast data path using one-sided RDMA
   - Metadata needs to include target address for read/write (not shown)
   - Have the NIC DMA to/from actual application buffers

```
Future<int> ret =
    stream.Read(buf);
//..do work
ret.get();
```



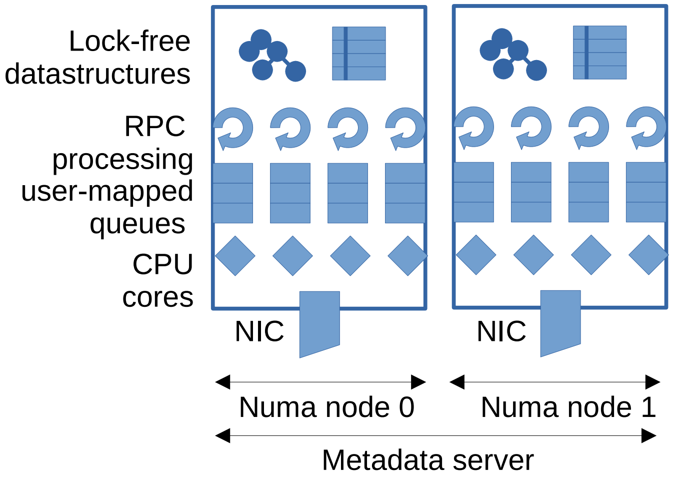
very close to HW limits





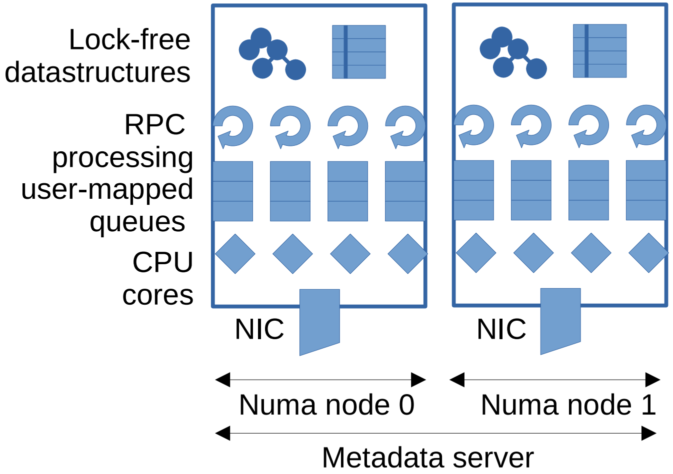
Performance/Design Decisions

1. Fast data path using one-sided RDMA
   - Metadata needs to include target address for read/write
   - Have the NIC DMA to/from actual application buffers
2. Scale metadata RPC using two-sided RDMA
   - Keep RPC request/response messages small (< 128 bytes)
   - Process RPC in-place on receiving core
   - Avoid NUMA remote memory access

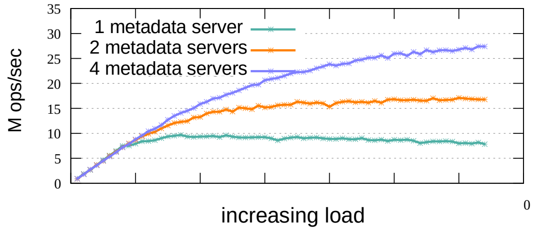


Performance/Design Decisions

1. Fast data path using one-sided RDMA
   - Metadata needs to include target address for read/write
   - Have the NIC DMA to/from actual application buffers
2. Scale metadata RPC using two-sided RDMA
   - Keep RPC request/response messages small (< 128 bytes)
   - Process RPC in-place on receiving core
   - Avoid NUMA remote memory access



1 metadata server can serve ~10 million requests per second



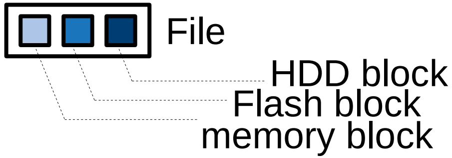
Performance/Design Decisions

1. Make data path fast using one-sided RDMA
   - Metadata needs to include target address for read/write
   - Have the NIC DMA to/from actual application buffers
2. Scale metadata RPC using two-sided RDMA
   - Keep RPC request/response messages small (< 128 bytes)
   - Process RPC in-place on receiving core
   - Avoid NUMA remote memory access
3. No threads, execute all I/O in the process context of the app
4. Avoid interrupts for small data transfers and RPCs
5. Per NUMA node pre-pinned buffer pool for application memory

Performance/Design Decisions

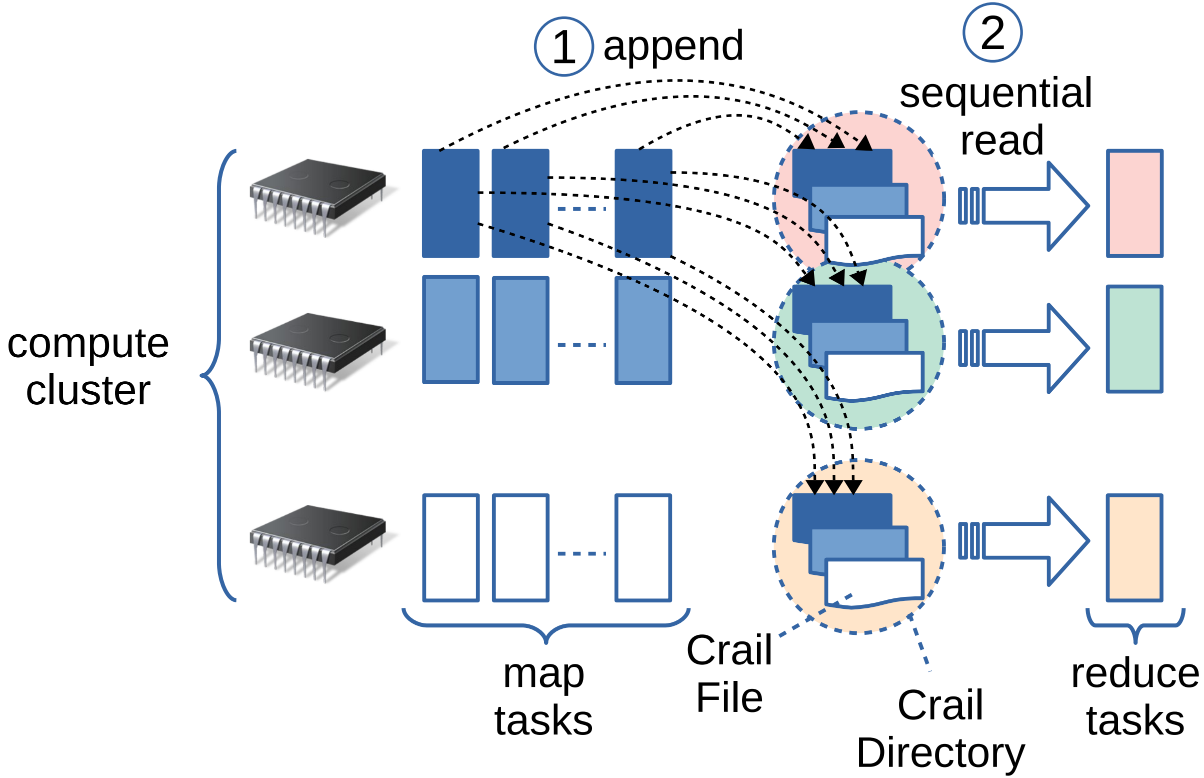
6. Horizontal tiering

- Store data in Flash **iff** all DRAM in the cluster is exhausted



- Remote Flash ≈ Local Flash

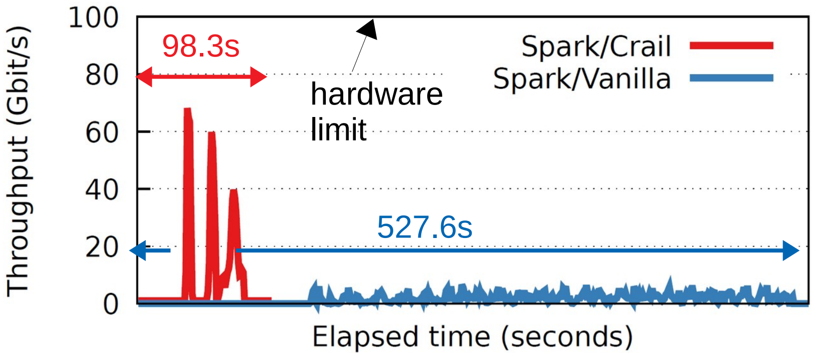
Example: Spark Shuffle using Crail



How to avoid large numbers of small files?

1. Don't: Crail performs well for small files
2. Re-use per core files
3. Interleaved reading

Sorting 12TB on 128 Node Cluster



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

www.sortingbenchmark.org

Pocket: Ephemeral Storage for Serverless Analytics

Serverless Analytics

- Serverless frameworks are increasingly being used for **interactive analytics**

PyWren
(SoCC'17)

ExCamera
(NSDI'17)

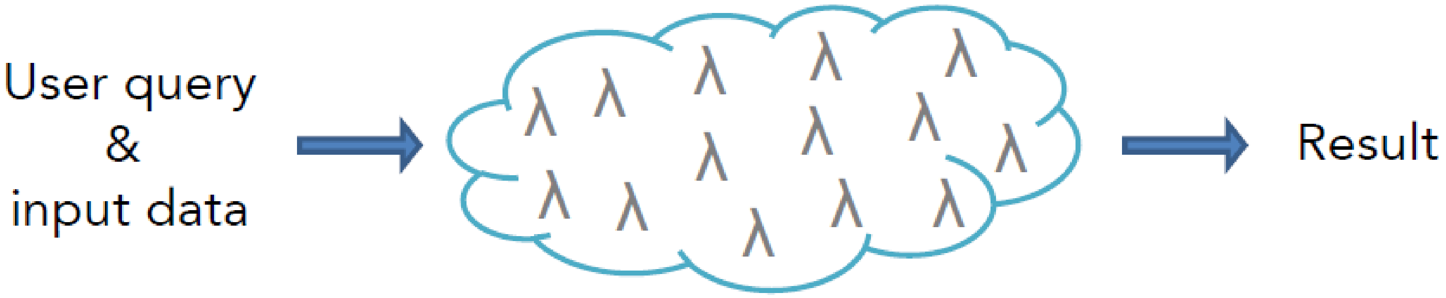
databricks
serverless

gg: The Stanford Builder

Amazon Aurora
Serverless

Serverless Analytics

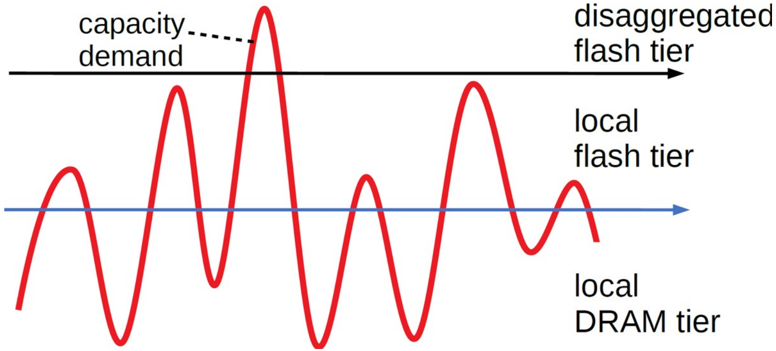
- Serverless frameworks are increasingly being used for **interactive analytics**
  - Exploit massive parallelism with large number of serverless tasks



Challenge: Data Sharing

- Serverless analytics involve multiple stages of execution
  - Serverless tasks need an efficient way to communicate **intermediate data** between different stages
- Today: such data sharing is 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?

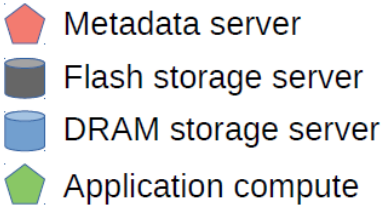
Crail Deployment Modes



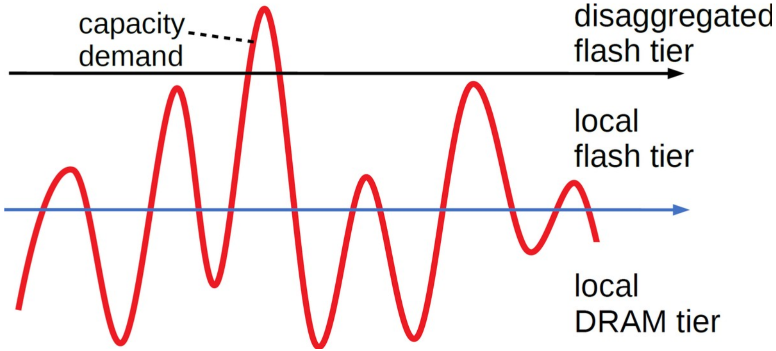
compute/storage
co-located

storage
disaggregation

flash storage
disaggregation



Crail Deployment Modes



compute/storage
co-located

storage
disaggregation

flash storage
disaggregation

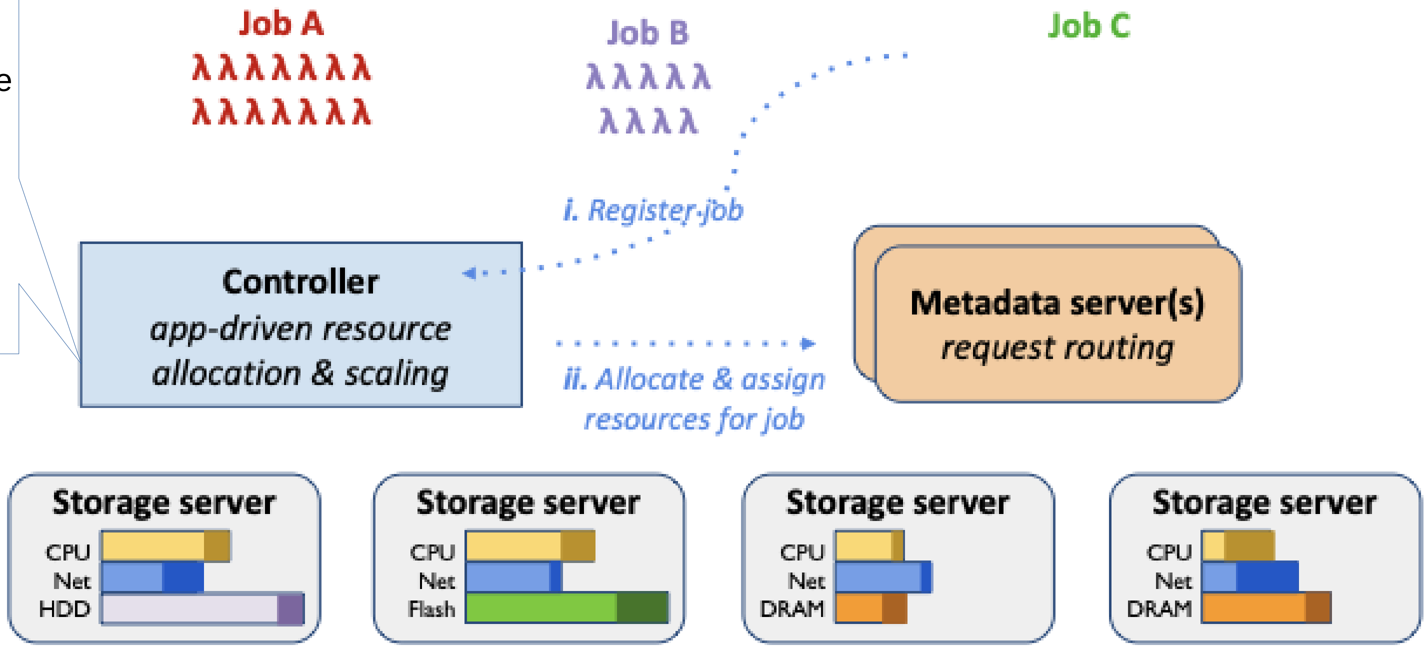
Not enough. We also
need elastic scaling.

Pocket Overview

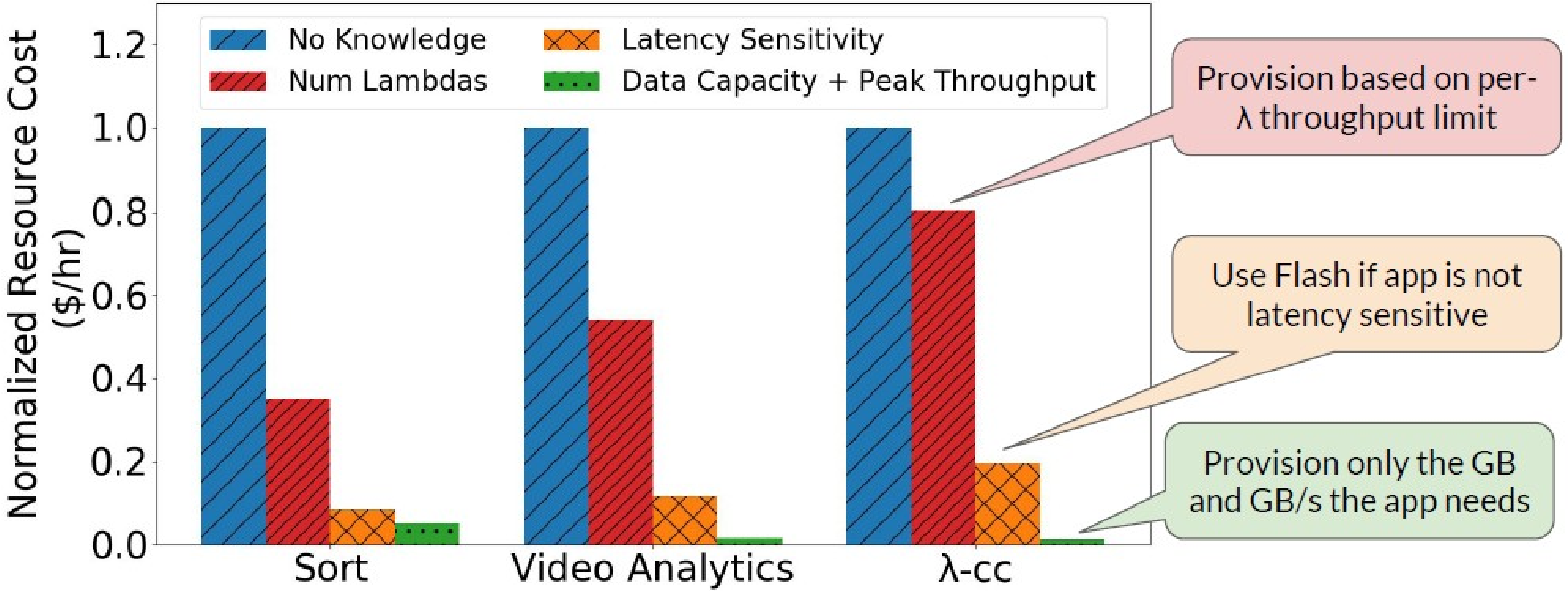
Brain of the system, decides:

1. Type of storage resources to use for a job based on job properties

2. When to scale up/down

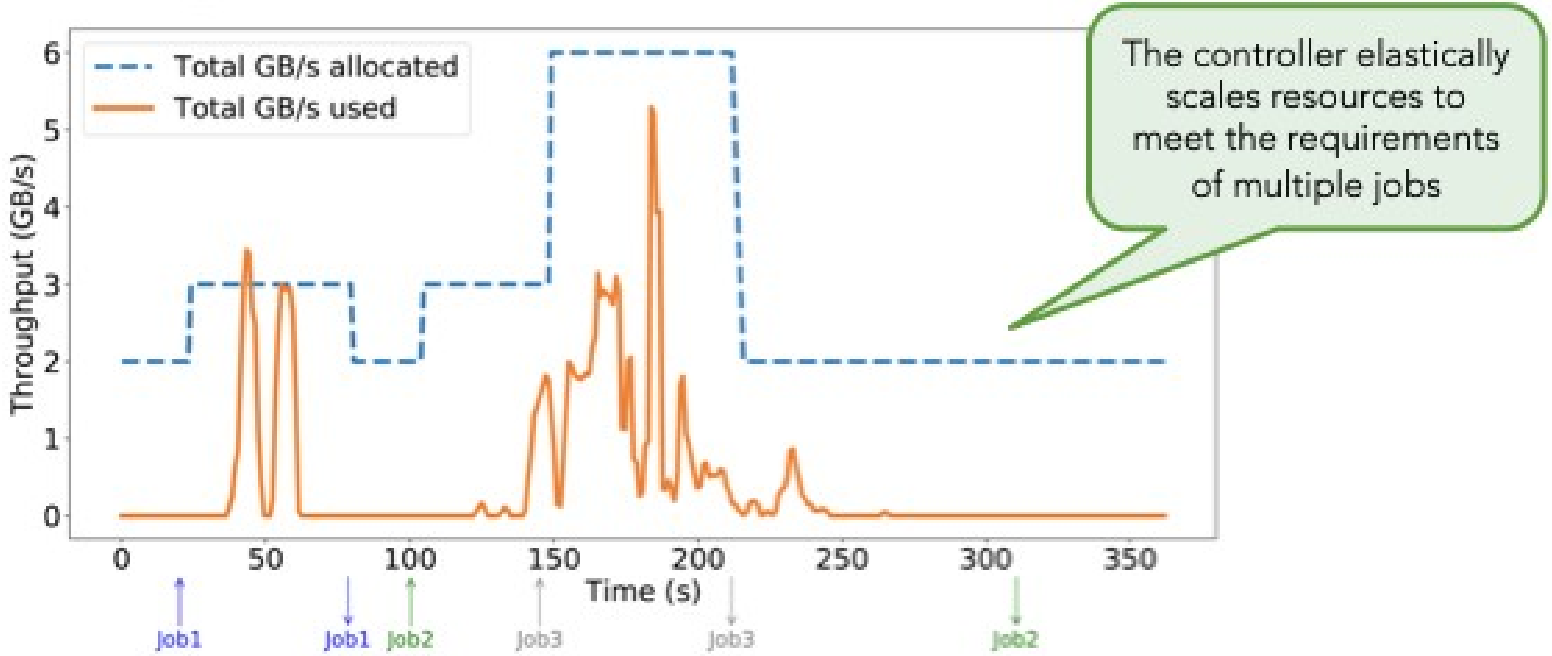


Pocket: Resource Utilization



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

Pocket: Autoscaling



References

- crail.apache.org
- github.com/apache/incubator-crail (Java)
- github.com/patrickstuedi/crailnative (C++)
- Pocket: Elastic ephemeral storage for serverless analytics, OSDI'2019
- github.com/stanford-mast/pocket
- Wimpy nodes with 10 GbE: Leveraging One-sided RDMA operations to boost Memcached, USENIX ATC'12

Contributers

Crail: Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Bernard Metzler, Adrian Schuepbach, Ana Klimovic, Yuval Degani

Pocket: Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Christos Kozyrakis