Jumpgate: In-Network Processing as a Service for Data Analytics

Craig Mustard Fabian Ruffy Anny Gakhokidze
Ivan Beschastnikh Alexandra Fedorova
University of British Columbia

Abstract

In-network processing, where data is processed by special-purpose devices as it passes over the network, is showing great promise at improving application performance, in particular for data analytics tasks. However, analytics and in-network processing are not yet integrated and widely deployed. This paper presents a vision for providing in-network processing as a service to data analytics frameworks, and outlines benefits, remaining challenges, and our current research directions towards realizing this vision.

1 Introduction

In-network compute capability is growing. Figure 1 illustrates the network-attached compute resources that are (or are becoming) available in modern data centers. New programmable hardware includes packet processors that can efficiently process packets using specialized hardware designs, such as SmartNICs [20, 75], programmable switches [12], and NPUs [13, 58, 80]. In parallel, software data paths are becoming increasingly programmable via tools like the Data Plane Development Kit (DPDK) [69] or the eXpress Data Path (XDP) [27], while network function virtualization platforms are being optimized for flexible high-performance packet processing [62, 65, 83].

Programmable network processors can improve application performance. Prior work has shown the benefits of taking advantage of compute that is conveniently located along the data-path [23, 32, 33, 45–47, 49, 50, 72]. However, existing systems that have used in-network processing have been application specific [45–47, 50] or have only been feasible in the research lab. Often, they require invasive changes to infrastructure or the use of unreliable network protocols [23, 45, 50].

How can we deploy in-network processing in real settings?

Our vision: We propose In-Network Processing as a Service (NPaaS)[1]. End-users continue to write high level programs (e.g., SQL) and use existing data processing frameworks such as Apache Spark [7], but the framework would use NPaaS to offload operations to processing elements along (or near) the data path. A common scenario might be NPaaS initiating the read from storage, pre-processing the data, and directing the processed data to compute nodes of the framework. NPaaS could be operated by cloud providers or as a user-run service that requests lower level resources from the provider. Realizing this vision will require designing solutions to challenges presented by this new type of computing, including changes to the existing system stack (§2). To drive research into NPaaS, we are developing a prototype called Jumpgate (§3).

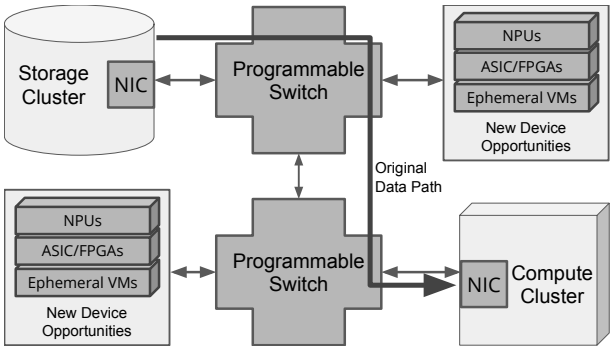

Figure 1: Opportunities for in-network processing. Darker boxes show new programmable opportunities for processing. The solid arrow shows the path that data takes from storage to compute nodes.

1.1 Motivation

Why should data analytics adopt in-network processing?

Data analytics need to leverage new high performance hardware. Data analytics is an expensive but common task. Substantial effort has been spent to accelerate query workflows with high performance devices. We view network processors as another heterogeneous processing device to be adopted in order to improve performance. Just as data analytics systems have integrated GPUs [70], TPUs [2],

[1] We pronounce it similar to 'impasse'.

DPUs [3], and FPGAs [37], they should also coordinate with in-network processors. Since many analytics tasks can be CPU-bound [60], offloading operations to network devices helps to alleviate valuable CPU resources [20] while decreasing workload execution time for the end user. Prior work and our estimates show that in-network processing can accelerate data analytics tasks by orders of magnitude, detailed in §1.2.

We believe that network processing devices, special-purpose and resource constrained as they may be, can act as specialized compute stages as part of data analytics pipelines. For example, Reconfigurable Match Table (RMT) switches [12], programmed in P4 [11], present a very challenging and restrictive programming environment. In spite of this, prior work has used programmable switches to perform join-and-group-by operations [45], aggregations [72], build replicated key-value stores [32, 33], optimize consensus and transaction protocols [46, 47], offload network telemetry queries [25, 57], and implement several network management algorithms [73, 74]. Work like Domino [74] and FlowBlaze [68] provides even higher-level languages for these devices.

We can't move all compute to storage. Existing data processing systems try to place compute near storage nodes to reduce data movement costs [82]. But, modern cloud architectures are decoupling compute and storage to achieve better resource utilization and fault tolerance. Instead of general purpose machines with attached storage, we now have special-purpose data serving machines [26, 38, 56] with very little compute capability. In these deployments, the classic 'move compute to data' optimization is no longer possible.

For instance, Facebook's Bryce Canyon storage server combines 72 high capacity hard drives with two Xeon D-1500 SoCs [16, 17], each with at most 16 cores – *less than half a core per drive.* Since we can no longer place a lot of compute on storage nodes, *we should find ways to place computation near data, and our next option is the network path*.

Data analytics need flexible deployments. Data processing systems behave inefficiently if their nodes are statically allocated in advance and they assume no other compute resources are available. For instance, to filter a large volume of data quickly, we require many nodes. After applying a highly selective filter, the data will be 10-1000x smaller but distributed across the same nodes used to filter it. However, it is often faster to process small datasets on a single machine because it avoids distribution overheads [53]. NPaaS could be used to push the filter operation into the network and ensure all relevant records arrive at a single host.

Ephemeral (serverless) compute service (e.g., AWS Lambda [1]) can address this need for flexibility [35], but *are currently hamstrung by the lack of direct access to the network* [26] and instead rely on inefficient cloud file systems for state exchange [39]. *If they had direct network access, we could treat ephemeral compute services as in-network processors.*

Why provide network processing as a service? As we've seen, in-network processing has a growing number of hardware implementations, but as far as we know, *there are no proposals to manage and expose this plethora of new hardware to end-users*. However, the burden of managing all this hardware should *not* fall to each data analytics framework, as Lerner et al. propose [45]. Instead, analytics systems and in-network processing should be decoupled with an interface that allows the framework to specify desired operations, while NPaaS manages instantiating requested operations on appropriate hardware. NPaaS allows hardware to be managed independently from the data analytics systems while saving analytics developers from implementing device-specific code. We describe potential general interfaces in §2, and our NPaaS prototype in §3.

We want to inspire more network-aware hardware designs. We believe network processors are good hardware accelerators for data analytics. Recent work on near-storage [15, 24, 31, 34, 41] or in-memory processors [3, 10, 22, 81] tightly couples storage with specialized processors, but as we've discussed, storage and compute are better utilized if they are decoupled. We think more data processing devices could be designed assuming data arrives over the network as coordinated by NPaaS, and hope that our work inspires new network-aware data analytics accelerators.

1.2 Performance Potential

In-network processing can reduce the overall work required of end-hosts by performing computation in network, reducing data volume and speeding up applications. To demonstrate, we run an experiment and draw on prior work to show that in-network processing will benefit data analytics.

Experiments. We estimate the effect of offloading common analytics operations to the network by measuring reduction in time and bytes transmitted on a query[2] run using Apache Spark. We offload operations in the query that require no storage (filter, projection, shuffle) or can be bounded (partial aggregation). To measure traffic reduction, we pack each record in a UDP packet and drop or modify packets as dictated by the operation, measuring data sent over the wire before and after. To measure query time, we simulate offloading operators using Spark SQL to query pre-processed files that contain data that would come from the offloaded operators. This assumes operators work as fast as our cluster[3] so our measurements give an upper-bound on the benefits.

Table 1 summarizes our results, averaged over 5 runs. **Filter and Project** reduce traffic commensurate with the amount of rows or columns they remove. For our query test, *shuffle and partial aggregation also apply filter and project*. **Shuffle** eliminates the need for nodes to exchange records by sending

[2] `SELECT item_sk, sum(quantity) FROM store_sales GROUP BY item_sk WHERE item_sk < I`, where `I` selects 50% of records

[3] A 4-node Spark cluster using Azure E4v3 VMs, with 4 cores each.

Operator Offloaded	Traffic Reduc.	Spark Query Time	
		13.7GB	68.9 GB
None	0%	56s ±0.8s	256s ±5s
Filter	50%	37s (-35%)	124s (-52%)
Project	85%	20s (-64%)	38s (-86%)
Shuffle	40%	14s (-76%)	31s (-88%)
Partial-Agg	90%	14s (-76%)	17s (-94%)

Table 1: Effect of in-network offload on traffic and Spark query time. Input data is TPC-DS `store_sales` as JSON.

	Byte stream	Record / Datagram
Ex.	TCP	UDP, DCCP, SCTP
Reqs.	Flow tracking.	Record-packet alignment.
Pro	Fewer changes to existing software.	Efficient per-packet processing.
Con	Must modify flow.	Must packetize records.

Table 2: Comparison of transport layers for in-network processing, showing known protocols, requirements, pros, cons.

records to the right node in the first place with a network operator. Based on traffic reduction, shuffle should improve query time more, but Spark's API lacked a way to declare that data is pre-shuffled and our hand-written aggregation is slower than Spark's native SQL operators [44]. If Spark's optimizer had such an API, we could use the native operation.

Partial Aggregation, inspired by [25, 57], aggregates within a storage limit (8MB) by evicting partially aggregated records and sending it to the end host. Overall, as we offload most operations to the network, **query time drops by 94% of the baseline (a 16x speedup).** Given partial-aggregation runs at just 4 GB/s (on 4 nodes), assuming data can be processed as fast as Spark reads seems reasonable because 40Gbps RMT switches operate at 5 GB/s and the latest software JSON parsers can operate at 2 GB/s *per core* [42, 48].

Prior work has already shown benefits of performing in-network operations. For in-network aggregation: Using RMT switches, Sonata [25] reduced network telemetry traffic to the end host by 3-7 orders, DAIET [72] reduces aggregation traffic by 86-89% (6-8x). Using middleboxes, NetAgg [50] increased search result aggregation throughput by 9.3x, and reduced total Hadoop execution time by 4.5x. TAG [49] aggregates sensor data and decreased traffic by 8x. Mellanox's SHArP [23, 54], a commercial in-network aggregator, shows a 2x performance improvement on an MPI programs [23].

More complex operations and compilation strategies are being explored: Netaccel [45] runs join-and-group-by on an RMT switch and accelerates a TPC-H query fragment by 2x. The Marple [57] compiler translates network telemetry queries to P4 code, and Sonata [25] partitions these queries between end-host machines and programmable switches. Floem [66] enables experiment with various NIC offloading strategies, and improves data analytics throughput up to 96%.

2 NPaaS Design Challenges

#1: NPaaS requires extensive co-design.

Given an analytics task, data source and destination endpoint, NPaaS must orchestrate the processing of data passing over the network by selecting and configuring available devices. This is challenging because the devices that can be used depend on the data format and transport protocols used to send data, which in turn depend on where data is stored. For example, Figure 1 shows on-path and off-path devices. On-path devices operate at line-rate, but can only read a small number of bytes of each packet, and have small amounts of state [11, 14, 45, 72]. Off-path devices are more flexible, but slower, and introduce latency. *Achieving even basic network processing requires co-designing storage systems, network transport, and data formats with the capabilities of the available network processors.*

Network Transports. Table 2 covers network transports NPaaS could use to send data. Our main constraint is many network devices (see §1.1) only operate on packets, and can't buffer data across packets in a network flow. To operate per-packet, a complete record must be in a single packet. In other words, each record must be *packetized*.

Stream protocols like TCP require fewer modifications to existing systems. But, operating on a flow requires buffering data, observing all packets, and modifying TCP state; tasks that are outside the capabilities of packet processors. Even if records are chunked into individual TCP packets, they cannot be reordered or dropped without tracking the behavior of the TCP state machine *per flow*. In practice, modifying TCP streams requires a proxy, stream processor or middlebox.

Prior work has thus sent records over UDP-based protocols (e.g., Netaccel [45]). In practice, we need a reliable protocol to ensure data has been processed. Fortunately, reliable datagram protocols, such as SCTP [76] and DCCP[4] [40], are available in the Linux kernel today. However, the downside to datagram protocols is that *the sender must packetize records.*

Data Formats. Table 3 covers data formats NPaaS might support. Our main concern is the ability to process popular data formats while guaranteeing the packet content is parseable by the network devices we wish to support. For instance, RMT switches can only process 200-500 bytes of fixed-length data from each packet [12, 72, 73].

Unstructured formats, like JSON, are prevalent but are difficult to parse quickly [42, 48, 61], although there are promising

[4] Assuming a reliable layer on top of the congestion control protocol.

Flat Columnar	Nested Columnar	Unstructured
Examples:		
ORC,Arrow,Albis [5, 59, 77]	Parquet,Dremel [6, 55]	CSV, JSON
Storage Pro/Cons:		
Low storage overhead, but must be converted.		No conversion, high overhead.
Packetizing complexity:		
Fixed/length-prefixed records	Reassembly [36]	Newline search

Table 3: Data formats categorized by their storage-side benefits, and challenges for storage-side record alignment.

hardware accelerators [18]; binary formats can be parsed more quickly. We argue NPaaS should support both unstructured data (JSON) for general applicability and binary formats for best performance and compatibility with hardware. Not all in-network processors need to support all formats, but NPaaS should allow for processing of different formats.

Data formats vary in how complex it is to find record boundaries to packetize data. JSON or CSV records are easily found by searching for newlines [8, 28] and flat binary formats just need offset calculations, but binary formats for nested schemata require complex algorithms [36].

Storage System Requirements. As said earlier, to send data via record transport, a storage system needs to packetize records. We must also be careful with distributed file systems that split data files into chunks, like HDFS [9] or Ceph [79]. Since chunking is done without awareness of records, records can span multiple chunks, and it is possible to see incomplete records at the flow level. If we want to use a record transport, one solution is to have a middlebox read all chunks that make up a file, transcoding data into a record transport on the fly.

Open Research Questions. There are many more research questions along the road to co-designing NPaaS. Here is a short selection. ***Q1****: How should we allocate and schedule processing with respect to the network topology? How long will processing pipelines need to last?* ***Q2****: Can middleboxes perform fast enough to make a difference to applications?* Even though hardware packet processors are faster, it is worthwhile to implement operators in software, even to just provide a performance baseline. ***Q3****: Are existing transport protocol sufficient for NPaaS or do we need custom protocols?* Customized protocols allow for domain-specific optimizations but require significant development effort.

#2: Integration with Analytics Frameworks.

Multiple data analytics systems need to communicate their operations to NPaaS. Is there a common format? What operations can be supported? Conveniently, most data processing systems model programs as dataflow between nodes in an abstract graph [2, 4, 30, 82] and have equivalent operators: many support SQL dialects and functional style operations (e.g., filter, project, map, reduce, group-by, shuffle) that can be directly mapped between frameworks [21].

When an analytics framework requests computation, the task for NPaaS is to map the desired operations to implementations on network processors. A simple way is to use pre-written or templated implementations for each device. To support framework-specific operators, or avoid hand-coding a multitude of operators, NPaaS could draw on cross-framework intermediate representations that enable compilation to different backend devices (e.g., Weld [63, 64], Dandelion [71]).

#3: Multi-tenancy and Isolation.

How can we run user code on hardware that lacks isolation? If NPaaS only uses client-allocated resources, such as VMs and containers, isolation and multi-tenancy is, arguably[5] addressed by existing isolation mechanisms. But, if NPaaS runs user-provided code on provider-managed devices that lack hardware isolation, such as programmable switches, a major challenge is to ensure user programs don't abuse access to the switch. A promising option is software isolation (SI), as proposed by Singularity [43], where user supplied programs are type and memory safe and restricted to use specific interfaces. Proposals to support this notion of isolation already exist for switches [51] and Netbricks [65] uses SI to eliminate hardware-isolation costs to improve performance. Similarly, Azure SmartNICs let users write policies for the Virtual Filtering Platform (VFP) [19], which implicitly restricts them to the user's network. NPaaS can provide operations which are guaranteed to only act on traffic belonging to the requester.

#4: Failure handling and debugging.

How should NPaaS recover from failure? NPaaS systems will be composed of heterogeneous devices with their own failure modes. Detecting failures will be an ongoing issue that NPaaS systems must address. Prior work handles failures by restarting jobs on new nodes [23, 45] or by routing around the failure [50]. The best option depends on the expected lifetime of the pipeline, likelihood of a failure, and any existing failure recovery mechanisms. For short-lived jobs, failure is unlikely and restarting is fast, so a lightweight mechanism like restarting is ideal [45]. For longer-lived jobs, partial recovery becomes ideal [50]. In either case, the calling framework may provide better failure recovery than NPaaS can provide (e.g., Spark's lineage graph [82]).

How can we debug in-network programs? Data processing errors are often data dependent and difficult to trace because errors are caused by a few malformed data records [29]. As with existing distributed computation systems, NPaaS should

[5] Putting aside recently discovered side-channel vulnerabilities.

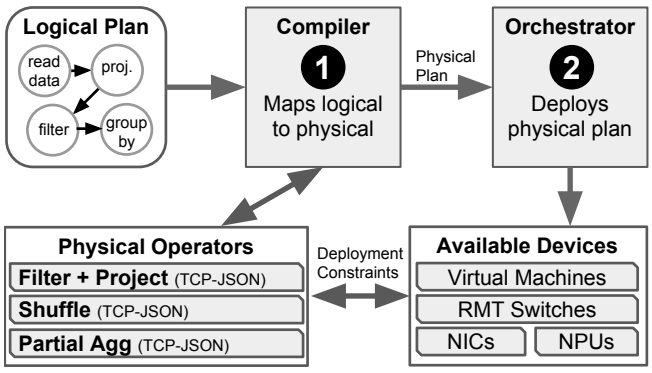

Figure 2: Overview of the how Jumpgate will interact with a data processing system to compile and orchestrate in-network processing.

return a summary of any problems encountered during execution to the framework, including enough context to help find the problematic record(s).

#5: Who should provide NPaaS?

Is it better if the cloud provider or the user operates NPaaS? The cloud provider has a privileged view of the network topology, lower level control over the hardware and network configuration, and often more money to pay experts to develop fast operators. On the other hand, users have a better understanding of their specific workload and could implement workload-specific operators. Users may also prefer to keep any needed encryption keys on infrastructure they own. One way to capture the best of both is for the user to operate the NPaaS system, while allocating proprietary in-network operators from the cloud provider.

3 Jumpgate: Our NPaaS implementation

We are developing Jumpgate, a compiler and orchestration system that can provide NPaaS. Jumpgate provides a client-facing API and maps client requests to relevant devices using an extensible architecture that simplifies adding new operators and devices without modifying the client.

Figure 2 shows an overview of a client's interaction with Jumpgate. **Step 1:** the compiler receives a logical plan of operations from data analytics frameworks (e.g., filter, project, shuffle, partial or full aggregation) and maps logical operations to stages of physical operators that can be deployed on available devices. **Step 2:** the orchestration layer coordinates runtime execution of the physical plan on devices. Devices are allocated and initialized to perform the physical operations[6]. After initialization, network addresses are known and can be propagated as needed (e.g., end-host addresses, or next-hop addresses in a processing pipeline).

Client API. Jumpgate's API allows a client to specify input data sources (files or network tuples), a DAG of logical operations to apply to the input data, and the destination network tuples for receiving processed data. We plan to expand the API as our evaluations determine a need for it (e.g., to communicate skew and cardinality of input data).

Mapping. The compiler maps logical operations to stages of pipelines of physical operators. For example, joins require a build stage to load data into the device and then a probe stage to output matched records. Since in-network operations execute concurrently in pipelines, we must make sure that other operators execute at the correct time to feed and receive data from the join. Jumpgate computes all needed stages and which stage(s) each operator executes in using a dependency-driven simulation. Logical operators are then mapped to physical operators, as determined by each physical operator's matching functions (below).

Extensibility. Jumpgate supports adding new logical and physical operators. For instance, a logical operator that translates unstructured to structured data, or a physical operator that runs filters on supporting storage systems. Logical operations are represented as typed nodes in the DAG with specified input/output connectivity and stage logic. Physical operators are comprised of: (1) matching functions and (2) a controller used to drive device allocation and execution. Matching functions determine which logical operators can be replaced based on type and properties, such as available resources and operator-specific parameters. We plan to evaluate heuristic and optimal approaches (e.g., SMT solvers) for exploring the space of potential logical to physical matches.

Limitations and Future Work A compiler and orchestration layer are bare necessities for providing NPaaS, but are not sufficient for a full NPaaS system. At the very least, NPaaS also needs failure handling and topology-aware scheduling. For now, we are focused on enabling analytics applications to execute in-network computations on software and programmable in-network devices, supporting common operations (e.g., TPC-DS [67], BigDataBench [78]), and measuring performance. Our eventual goal is to share Jumpgate with other researchers in order to answer broader challenges posed by this paper and lay fertile groundwork for research into making in-network processing available to more users.

4 Conclusion

In-network compute capability is growing, and the benefits to data analytics are clear. To allow all analytics systems to benefit, this paper advocated for providing in-network processing as a service (NPaaS) that abstracts the desired operations on data from their low-level implementation on network processors. We are developing Jumpgate to provide NPaaS and help drive research and adoption of in-network processing.

[6] If any physical operators need to generate and compile code, it could happen at this point.

5 Contributions to Workshop Discussion

Feedback desired and open issues: We are not cloud providers and have little knowledge about the day-to-day deployment and management challenges such a company might face if they offered NPaaS. While we have covered many open issues in §2, we are hoping to hear about additional challenges we have not addressed. Examples include the integration of NPaaS into cloud systems and potential security, fairness, reliability, and usability concerns. In particular, we are interested in discussing difficulties deploying applications to FPGAs and network accelerators currently used in data centers, so that we can work on addressing them in Jumpgate.
Discussion: More generally, we are hoping to spark a discussion of how best to expose in-network processing hardware to end users and allow them to accelerate their applications, ideally without writing low-level code. For example, are analytics operators the right abstractions to expose in-network processors to users? Are there other abstractions that could be more useful or more easily deployed?
Controversial points: Placing more functionality than just packet processing on networking devices is highly controversial. For instance, McCauley et al. [52] argue strongly against pushing complex operators to the network, on the basis that it is unnecessary for performance and too restrictive to applications semantics. Detractors might suggest we instead develop more efficient data analytics software. However, we believe that network processors present a unique hardware architecture that is distinct from classical out-of-order CPU designs, that can benefit data analytics so much (§1.2) that it is worthwhile exploring ways to make them available to end-users.
Circumstances in which the whole idea might fall apart: This project hinges on the assumption that network processors will become ubiquitous in datacenters. This requires that these devices are cost-effective and flexible enough to be widely used. Currently, this is at a stand-still: users can't benefit from in-network processing until cloud providers make programmable network processors available, but providers won't bother if there are no popular use-cases. By proposing NPaaS we hope to break this deadlock.

References

[1] AWS Lambda: Run code without thinking about servers. https://aws.amazon.com/lambda/.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Sandeep R Agrawal, Sam Idicula, Arun Raghavan, Evangelos Vlachos, Venkatraman Govindaraju, Venkatanathan Varadarajan, Cagri Balkesen, Georgios Giannikis, Charlie Roth, Nipun Agarwal, and Eric Sedlar. A many-core architecture for in-memory data processing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 245–258, New York, NY, USA, 2017. ACM.

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015.

[5] Apache Software Foundation. Apache Arrow. http://arrow.apache.org/. [Online; accessed 5-Dec-2018].

[6] Apache Software Foundation. Apache Parquet. http://parquet.apache.org/. [Online; accessed 5-May-2018].

[7] Apache Software Foundation. Apache Spark. http://spark.apache.org/. [Online; accessed 10-April-2017].

[8] Apache Software Foundation. Apache Spark: JSON Files. https://spark.apache.org/docs/latest/sql-data-sources-json.html. [Online; accessed 18-Dec-2018].

[9] Apache Software Foundation. HDFS Architecture Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Online; accessed 10-April-2017].

[10] Cagri Balkesen, Nitin Kunal, Georgios Giannikis, Pit Fender, Seema Sundara, Felix Schmidt, Jarod Wen, Sandeep Agrawal, Arun Raghavan, Venkatanathan Varadarajan, Anand Viswanathan, Balakrishnan Chandrasekaran, Sam Idicula, Nipun Agarwal, and Eric Sedlar. Rapid: In-memory analytical query processing engine with extreme performance per watt. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1407–1419, New York, NY, USA, 2018. ACM.

[11] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[12] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[13] Cavium. Liquidio ii network appliance smart nics. https://www.cavium.com/liquidio-II-network-appliance-adapters.html.

[14] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. Fboss: building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 342–356. ACM, 2018.

[15] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. Query processing on smart ssds: Opportunities and challenges. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1221–1230, New York, NY, USA, 2013. ACM.

[16] Facebook Inc. Bryce Canyon Storage Specification. https://www.opencompute.org/contributions?query=bryce%20canyon%20spec. [Online; accessed 20-Dec-2018].

[17] Facebook Inc. Mono Lake Server Specification. https://www.opencompute.org/contributions?query=mono%20lake. [Online; accessed 20-Dec-2018].

[18] Yuanwei Fang, Chen Zou, Aaron J. Elmore, and Andrew A. Chien. Udp: A programmable accelerator for extract-transform-load workloads and more. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 55–68, New York, NY, USA, 2017. ACM.

[19] Daniel Firestone. Vfp: A virtual switch platform for host sdn in the public cloud. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 315–328, Berkeley, CA, USA, 2017. USENIX Association.

[20] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, 2018. USENIX Association.

[21] Ionel Gog, Malte Schwarzkopf, Natacha Crooks, Matthew P Grosvenor, Allen Clement, and Steven Hand. Musketeer: all for one, one for all in data processing systems. In *Proceedings of the Tenth European Conference on Computer Systems*, page 2. ACM, 2015.

[22] V. Govindaraju, S. Idicula, S. Agrawal, V. Vardarajan, A. Raghavan, J. Wen, C. Balkesen, G. Giannikis, N. Agarwal, and E. Sedlar. Big data processing: Scalability with extreme single-node performance. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 129–136, June 2017.

[23] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *Proceedings of the First Workshop on Optimization of Communication in HPC*, COM-HPC '16, pages 1–10, Piscataway, NJ, USA, 2016. IEEE Press.

[24] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 153–165, Piscataway, NJ, USA, 2016. IEEE Press.

[25] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 357–371, New York, NY, USA, 2018. ACM.

[26] Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey

Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back, 2018.

[27] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, pages 54–66, New York, NY, USA, 2018. ACM.

[28] Ian Ward. Documentation for the JSON Lines text file format. http://jsonlines.org/. [Online; accessed 18-Dec-2018].

[29] Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, and Tyson Condie. Titian: Data provenance support in spark. *Proc. VLDB Endow.*, 9(3):216–227, November 2015.

[30] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

[31] Zsolt István, David Sidler, and Gustavo Alonso. Caribou: Intelligent distributed storage. *Proc. VLDB Endow.*, 10(11):1202–1213, August 2017.

[32] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, 2018. USENIX Association.

[33] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 121–136, New York, NY, USA, 2017. ACM.

[34] Insoon Jo, Duck-Ho Bae, Andre S. Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel D. G. Lee, and Jaeheon Jeong. Yoursql: A high-performance database system leveraging in-storage computing. *Proc. VLDB Endow.*, 9(12):924–935, August 2016.

[35] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 445–451, New York, NY, USA, 2017. ACM.

[36] Julien Le Dem. The striping and assembly algorithms from the Dremel paper. https://github.com/julienledem/redelm/wiki/The-striping-and-assembly-algorithms-from-the-Dremel-paper. [Online; accessed 18-Dec-2018].

[37] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 107–127, Carlsbad, CA, 2018. USENIX Association.

[38] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 29:1–29:15, New York, NY, USA, 2016. ACM.

[39] Ana Klimovic, Yawen Wang, Christos Kozyrakis, Patrick Stuedi, Jonas Pfefferle, and Animesh Trivedi. Understanding ephemeral storage for serverless analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 789–794, Boston, MA, 2018. USENIX Association.

[40] Eddie Kohler, Mark Handley, and Sally Floyd. Designing dccp: Congestion control without reliability. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 27–38, New York, NY, USA, 2006. ACM.

[41] Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. Summarizer: Trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 219–231, New York, NY, USA, 2017. ACM.

[42] Geoff Langdale and Daniel Lemire. simdjson : Parsing gigabytes of JSON per second. https://github.com/lemire/simdjson.

[43] James Larus and Galen Hunt. The singularity system. *Commun. ACM*, 53(8):72–79, August 2010.

[44] Jacek Laskowski. Spark's Whole-Stage Java Code Generation. https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-whole-stage-codegen.html.

[45] Alberto Lerner, Rana Hussein, and Philippe Cudre-Mauroux. The Case for Network-Accelerated Query Processing. CIDR 2019, 2019.

[46] Jialin Li, Ellis Michael, and Dan R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 104–120, New York, NY, USA, 2017. ACM.

[47] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. Just say no to paxos overhead: Replacing consensus with network ordering. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 467–483, Berkeley, CA, USA, 2016. USENIX Association.

[48] Yinan Li, Nikos R. Katsipoulakis, Badrish Chandramouli, Jonathan Goldstein, and Donald Kossmann. Mison: A fast json parser for data analytics. *Proc. VLDB Endow.*, 10(10):1118–1129, June 2017.

[49] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, December 2002.

[50] Luo Mai, Lukas Rupprecht, Abdul Alim, Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander L. Wolf. Netagg: Using middleboxes for application-specific on-path aggregation in data centres. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 249–262, New York, NY, USA, 2014. ACM.

[51] Mario Baldi. Exposing Data Plane Programmability - Mario Baldi. https://www.inf.usi.ch/faculty/soule/baldi.pdf. [Online; accessed 16-Jan-2019].

[52] James McCauley, Aurojit Panda, Arvind Krishnamurthy, and Scott Shenker. Thoughts on load distribution and the role of programmable switches. *ACM SIGCOMM Computer Communication Review*, 49(1):18–23, 2019.

[53] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? In *HotOS*, 2015.

[54] Mellanox Technologies. Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)™. http://www.mellanox.com/page/products_dyn?product_family=261&mtag=sharp. [Online; accessed 7-Jan-2019].

[55] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. *Proc. VLDB Endow.*, 3(1-2):330–339, September 2010.

[56] Mihir Nanavati, Jake Wires, and Andrew Warfield. Decibel: Isolation and sharing in disaggregated rack-scale storage. In *NSDI*, pages 17–33, 2017.

[57] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 85–98, New York, NY, USA, 2017. ACM.

[58] Netronome. Smartnics overview. https://www.netronome.com/products/smartnic/overview/.

[59] Apache ORC. Orc specification v1. https://orc.apache.org/specification/ORCv1/.

[60] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 293–307, Berkeley, CA, USA, 2015. USENIX Association.

[61] Shoumik Palkar, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Filter before you parse: Faster analytics on raw data with sparser. *Proceedings of the VLDB Endowment*, 11(11), 2018.

[62] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[63] Shoumik Palkar, James Thomas, Deepak Narayanan, Pratiksha Thaker, Rahul Palamuttam, Parimajan Negi, Anil Shanbhag, Malte Schwarzkopf, Holger Pirk, Saman Amarasinghe, Samuel Madden, and Matei Zaharia. Evaluating end-to-end optimization for data analytics applications in weld. *Proc. VLDB Endow.*, 11(9):1002–1015, May 2018.

[64] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*, 2017.

[65] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking

the v out of nfv. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, Berkeley, CA, USA, 2016. USENIX Association.

[66] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. Floem: A programming system for nic-accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 663–679, Carlsbad, CA, 2018. USENIX Association.

[67] Meikel Poess, Bryan Smith, Lubor Kollar, and Paul Larson. Tpc-ds, taking decision support benchmarking to the next level. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 582–587, New York, NY, USA, 2002. ACM.

[68] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Siracusano. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 531–548, Boston, MA, 2019. USENIX Association.

[69] DPDK Project. Data plane development kit (dpdk). https://www.dpdk.org/.

[70] Christopher J Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. Ptask: operating system abstractions to manage gpus as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 233–248. ACM, 2011.

[71] Christopher J. Rossbach, Yuan Yu, Jon Currey, Jean-Philippe Martin, and Dennis Fetterly. Dandelion: A compiler and runtime for heterogeneous systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 49–68, New York, NY, USA, 2013. ACM.

[72] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 150–156, New York, NY, USA, 2017. ACM.

[73] Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson, Changhoon Kim, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. Evaluating the power of flexible packet processing for network resource allocation. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 67–82, Berkeley, CA, USA, 2017. USENIX Association.

[74] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 15–28, New York, NY, USA, 2016. ACM.

[75] Brent Stephens, Aditya Akella, and Michael M Swift. Your programmable nic should be a programmable switch. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 36–42. ACM, 2018.

[76] Randall Stewart. Stream control transmission protocol, rfc 4960. Technical report, IETF, 2007.

[77] Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Adrian Schuepbach, and Bernard Metzler. Albis: High-performance file format for big data systems. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 615–630, Boston, MA, 2018. USENIX Association.

[78] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu. Bigdatabench: A big data benchmark suite from internet services. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–499, Feb 2014.

[79] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[80] Wikipedia. Tilera. https://en.wikipedia.org/wiki/Tilera.

[81] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. Q100: The architecture and design of a database processing unit. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 255–268, New York, NY, USA, 2014. ACM.

[82] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J.

Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[83] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 26–31. ACM, 2016.