Concerto: Cooperative Network-Wide Telemetry with Controllable Error Rate

Yiran Li
Institute for Interdisciplinary Information Sciences
Tsinghua University
liyr14@mails.tsinghua.edu.cn

Kevin Gao
Institute for Interdisciplinary Information Sciences
Tsinghua University
kevingao96@gmail.com

Xin Jin
Department of Computer Science
Johns Hopkins University
xinjin@cs.jhu.edu

Wei Xu
Institute for Interdisciplinary Information Sciences
Tsinghua University
weixu@tsinghua.edu.cn

ABSTRACT

Network-wide telemetry requires real-time analysis of a large amount of traffic. Telemetry systems use stream processors to support various applications, and Protocol Independent Switching Architecture switches to reduce the workload on stream processors. Due to the inefficient use of switch resources, existing systems cannot fully reduce the workload on the stream processors. Unlike the existing systems that treat switches independently when assigning tasks, Concerto lets switches work together. The use of cooperating switches means more resources and a further reduction in the stream processor's workload. Furthermore, Concerto can also adhere to a more stringent error rate requirement. Our evaluation shows that Concerto reduces the stream processor's workload by as much as 19×, and under the same workload on the stream processor, Concerto achieves an error rate of 10^4× lower than existing systems.

CCS CONCEPTS

• **Networks → Network monitoring**; **Programmable networks**.

KEYWORDS

Network-Wide Monitoring; Programmable Switches; Stream Processing

APSys '20, August 24–25, 2020, Tsukuba, Japan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8069-0/20/08…$15.00
https://doi.org/10.1145/3409963.3410499

1 INTRODUCTION

Network telemetry systems have powerful detection capabilities. They generate useful status knowledge by collecting and analyzing network information, providing a basis for many monitoring, diagnosis, and operational tasks. To obtain network status, a network-wide telemetry system should 1) provide expressive and high fidelity operators, and 2) operate packets of the whole network in real-time.

Currently, telemetry tasks are run using the Protocol Independent Switch Architecture (PISA) switch [5, 27] and the stream processor [10, 11]. However, neither of these can meet both of the conditions on their own. On the one hand, PISA switches provide stages for customized packet processing in real-time, but the limited resources and operations restrict query fidelity and expressiveness. On the other hand, the stream processor is capable of general-purpose processing. Still, even state-of-the-art stream processing systems only achieve the line rate of the server, failing to process packets across the entire network [17, 35]. In order to execute expressive operators on the entire network in real-time, existing systems reduce the stream processor's workload using PISA switches [12]. The key to the design of a telemetry system is to reduce the workload on the stream processor.

The network-wide telemetry system faces similar problems as traditional systems, but its unique constraints call for a different solution. Database systems reduce operation sizes by choosing the evaluation plan. Dataflow computing frameworks utilize multiple servers by decomposing jobs into tasks [1, 10]. The telemetry system also targets to reduce the number of tuples to the stream processor and needs to split queries among devices. However, telemetry is different as 1) the switch resources determine the number of operators on the switch, it is impossible to trade tuple processing speed for more operators. And 2) the network configuration fixes the tuple processing workflow, resulting in various switch sets for different operations. Due to the strict processing constraints, telemetry systems are hard to benefit from various

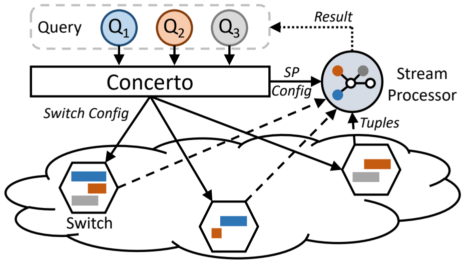


Figure 1: Concerto architecture.

evaluation plans or dynamic task assignments as database systems and dataflow computing frameworks.

Existing systems cannot sufficiently reduce the stream processor's workload on network-wide telemetry. 1) Applying basic filtering on switches only reduces the stream processor's workload by a small portion. 2) UnivMon [21] and Marple [26] explore the opportunity to place queries on multiple switches individually. However, the limited resources of a single PISA switch still restrict the number of operators offloaded to switches. 3) Sonata [12] observes the unity between the PISA switch and the stream processor. But it is limited to the single switch setting due to the complexity of splitting queries among multiple switches dynamically. Besides, its zoom-in technique does not provide a fidelity guarantee about the result with dynamic flows.

As a step towards reducing the workload on the stream processor with all switches' resources, we present Concerto, a network-wide telemetry system that lets switches execute queries cooperatively. In Sonata, a stream processor and an edge switch complement each other. Concerto further orchestrates switches in the path to accompany the stream processor, performing queries on a much larger scale. Just as switches route packets in a distributive manner, Concerto runs queries distributively, resulting in various operator sets on different switches (Section 3.2). Since Concerto handles arbitrary topology, it also applies to data center networks.

The main challenge of Concerto comes from the difficulty of splitting queries among switches while meeting the constraints. Concerto solves this challenge by analyzing query placement requirements and formulating the placement problem as a mixed-integer program (Section 3.3). By letting the switches cooperate, Concerto avoids the restrictions caused by the limited resources of a single switch and provides high fidelity results in real-time. Figure 1 shows the Concerto architecture. Our contribution includes:

- We propose a cooperative query execution model that effectively reduces the stream processor's workload;
- We provide a method to automatically analyze the query and optimize its placement on PISA switches;
- We show that Concerto reduces the stream processor's workload by as much as 19×, and achieves an error rate of 10^4× lower than state-of-the-art systems.

Table 1: Summary of Concerto language constructs.

Construct	Description
`packetStream`	Stream of packets in the network.
`map(expr)`	Applying `expr` to each tuple.
`filter(expr)`	Filtering out tuples that violate `expr`.
`distinct(expr)`	Leaving one tuple for each `expr` group.
`scan(expr, op)`	Applying `op` on tuples with the same `expr`.
`zip(expr, S)`	Stream matching according to `expr`.
`reduce(expr, op)`	Same as `scan`, but only output the final result.
`join(expr, S)`	Matching with stream `S` according to `expr`.

2 RELATED WORK

Reducing workload and improving efficiency are widely discussed in network telemetry systems. Many systems use the end-host and optimize for real-time processing on the CPU [20, 24, 28, 33, 38]. Although state-of-the-art stream processing systems achieve the line rate of the server [17, 35], it is hard for them to process packets across the entire network. The limited resources on switches restrict operations and scalability. Sampling can largely reduce the resource consumption on switches [7, 21, 29, 31, 36], but does not fit most quantitative tasks. The zoom-in technique is another widely-used method [12, 14, 22, 23, 37]. However, it does not provide any theoretical guarantee about the fidelity with dynamic flows. Besides, some other systems focus on algorithmic aspects to improve sketch data structure performance [15, 16, 32, 34]. Concerto presents a different approach to utilize switch resources by letting switches cooperate. Note that Concerto is independent of other techniques, but can work together with them.

There are various attempts to network-wide query placement. Many systems combine switches and software, especially the stream processor, to perform telemetry tasks [12, 19, 26, 36]. Sonata [12] is limited to a single switch and does not provide any method for scaling to multiple switches. To support network-wide telemetry, UnivMon [21] utilizes switches along the path, while Marple [26] further considers operator semantics and places operators at specific switches. As UnivMon and Marple do not let switches cooperate, PathQuery [25] and SNAP [2] distribute packet processing logics among switches. However, they focus on providing functions and do not take resource consumption and workload into consideration. Concerto reduces the stream processor's workload by formulating switch cooperation with the network-wide query placement problem.

3 CONCERTO DESIGN

Table 1 shows the Concerto language constructs, which is a widely adopted interface [12, 26, 38]. Concerto provides one-big-switch abstraction by abstracting the traffic as a single packet stream (`packetStream`). Concerto supports stateless operators (`map`, `filter`), stateful operators (`distinct`, `scan`),

```
1  packetStream
2  .map(p => (p.ip.sip,p.ip.dip))
3  .distinct((sip,dip) => (sip,dip))
4  .map((_,dip) => (dip,1))
5  .scan((dip,_) => dip, sum)
6  .filter((dip,count) => count==T)
7  .map((dip,count) => dip)
```

Figure 2: DDoS detection query.

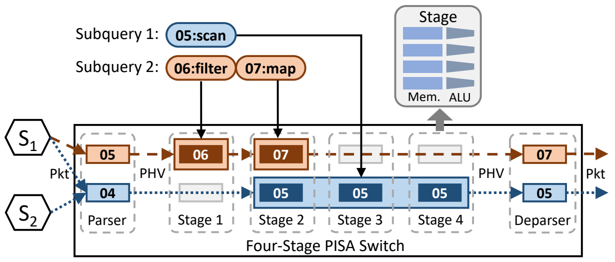


Figure 3: Concerto switch-based query execution.

as well as the `zip` on switches. As it is difficult to determine the last tuple or to generate all the possible combinations in a stream, Concerto also provides operators that are always done on the stream processor (`reduce`, `join`). We use the line number for phase, which is the last executed operator of the tuple. Figure 2 shows an example query of DDoS detection. The full list of evaluated queries is shown in Table 5.

The difficulty of network-wide telemetry comes from compiling the query to multiple switches, which takes query characteristics, network configurations, and switch resources into account. In this section, we start by providing basics for PISA-switch-based tuple processing. Next, we describe the cooperative query execution model of Concerto. Then, we introduce Concerto's automatic query planning method. At last, we use a case study to see the improvement of Concerto.

3.1 Tuple Reduction Using Switches

3.1.1 Protocol Independent Switch Architecture. The PISA switch provides a fixed number of stages for customized processing, together with a configurable parser and deparser [5, 27]. Figure 3 illustrates a four-stage PISA switch. We omit buffers for simplicity. For each incoming packet, the parser generates a packet header vector (PHV). The PHV is then used for customized packet processing and converted to the header of the outgoing packet at the deparser. Within each stage, when the PHV matches a customized rule in the match table, the specified operation is performed at the action ALU. Action ALUs support stateful operations with the help of register memories. Stages and registers are the key resources of the PISA switch. A typical PISA switch has 1-32 physical stages and 0.5-32 Mb of registers for each stage [6].

3.1.2 Offloading dataflow operators to switches. Concerto offloads query to PISA switches by translating operations to match-actions. Stateless operators and `zip` require a single match-action table to transform or remove the tuple in PHV. With the limited register in a single stage, stateful operators use consecutive hash tables like Bloom filter [3] or count-min sketch [8] to meet the required error rate. For k keys, when requiring the error rate less than p, the maximum number of distinct items n supported by d levels of hash tables is,

$$n = \left\lfloor -k \cdot \ln\left(1 - p^{\frac{1}{d}}\right) \right\rfloor .$$

We can precompute the number of hash tables given the trace and the error rate. In Figure 3, the stateful operator `05:scan` takes three stages, while others take one stage.

3.1.3 Best-effort tuple processing. In Concerto, switches send intermediate result tuples to the stream processor only when the switch resources are not enough or the `expr` is not executable on the switch. It also distinguishes `scan` and `zip` from `reduce` and `join` to offload restricted operations to switches and leave general operations to the stream processor.

3.2 Cooperative Query Execution

Due to the strict hardware restrictions and the complex routing, network-wide telemetry systems are hard to benefit from classical solutions. Different from identical workers in dataflow computing frameworks, each PISA switch in the network plays a unique role. Moreover, since network telemetry is more about filtering tuples than joining tables, it is more important to utilize switch resources efficiently.

Concerto proposes a cooperative query execution model. The model assigns various parts of the query to different switches and chains their stages to perform the query. The chaining finishes automatically as packets transfer. Concerto focuses on deterministic but potentially multi-path routing. That is, Concerto knows all switches a packet may traverse. This is a reasonable and common assumption [14, 21] as networks usually select among available paths. Note that Concerto is independent of the underlying routing method, as long as it knows the possible paths.

Each switch processes the tuple based on local information only (i.e., tuple's phase). The switch does not need to keep the sizable routing or operation information on other switches to process tuples.

Concerto adds a custom header before the payload to carry tuples and phases among switches. As Concerto does not modify original headers, other network behaviors are preserved. Table 5 (column Header) shows the maximum tuple size of different queries. We can see the maximum header overhead of evaluated queries is less than an IP header.

Figure 3 also shows the model. For example, the switch receives tuples at phase 4 from both S_1 and S_2. The switch processes all tuples of phase 4 to phase 5 ignoring its source.

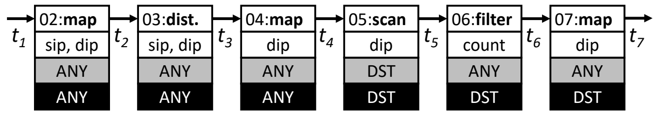


Figure 4: Query analysis of DDoS detection.

Table 2: Restriction solving of zip.

	Same Direction			Opposite Direction		
	ANY	SRC	DST	ANY	SRC	DST
ANY	ANY	SRC	DST	LST	DST	SRC
SRC	SRC	SRC	SP	SRC	SP	SRC
DST	DST	SP	DST	DST	DST	SP

3.3 Automatic Query Planning

Concerto plans the query under the cooperative execution model in two steps: 1) analyzing the placement requirement of the query, and 2) compiling the query to switches. The analysis deals with query semantics by extracting operators' placement restrictions, which is independent of network configurations. The compilation minimizes the stream processor's workload while meeting various constraints.

3.3.1 Query placement analysis. Figure 4 shows the analysis result of DDoS detection. Concerto analyzes the query placement in three steps. 1) Concerto parses the query into an abstract syntax tree. Each box in the figure represents an operator, and this step fills the first two lines of the box. Concerto builds separate DAGs for each query. 2) Concerto then extracts each operator's local restriction from `expr` (third line). The restriction has four types: *SRC*, *DST*, *ANY*, and *SP*, representing source edge switch, destination edge switch, any switch, and the stream processor, respectively. 3) Finally, Concerto propagates the local restrictions to final restrictions (fourth line). For an SRC-type (DST-type) operator, Concerto marks its preceding (succeeding) operators to be SRC (DST). Placing `zip` is more complicated as it involves two streams. Concerto tries to reverse the second stream when manipulating streams of the opposite directions (e.g., matching source IP with destination IP). The detailed rules for `zip` are shown in Table 2, in which LST stands for where the second stream places the previous stateful operator. When requirements conflict with each other, the operator will be marked SP.

3.3.2 Query compilation. Concerto formulates and solves the placement problem as a mixed-integer program (MIP) like previous works [2, 12, 21]. The formulation uses the trace to generate the number of tuples at different stages and paths, and the required number of stages for stateful operators. Table 3 summaries the variables in the network-wide query planning problem. We use subscripts t, u, v for phases, s for stages, w, x for switches, and p for paths. For simplicity, we focus on the single query setting, and only present the core formulation for the cooperative query execution model. We can easily extend it to multi-query by duplicating variables.

Table 3: Symbol list.

Input from the Network Configuration	
S	The number of stages in PISA switches.
o_p	Virtual origin node of path p before the source edge switch.
$e_{w,v}$	Edge from w to v.
Input from Queries and the Trace	
Z_t	Indicates whether phase t performs a stateful operation.
$N_{p,t}$	The number of tuples generated after phase t on path p.
Tuple Processing Variable	
$C_{w,t,u}$	Indicates whether w processes tuples at phase t to u.
$F_{p,w,t,u}$	Indicates whether w processes tuples in path p from t to u.
$H_{p,w,t}$	Indicates whether tuples in path p are at phase t after w.
$L_{w,t}$	Processing load of the stateful operator of phase t at w.
$T_{t,n}$	The maximum number of tuples handled by n consecutive hash tables for the stateful operator of phase t.
Output	
$X_{w,t,s}$	Indicates whether phase t executes at stage s of w.

$$\forall w,\, t: \quad \sum_{u>t} C_{w,t,u} \le 1 \tag{1}$$

$$\forall w,\, t < v \le u: \quad C_{w,t,u} = 1 \rightarrow \sum_{s \le S} X_{w,v,s} > 0 \tag{2}$$

$$\forall w,\, n,\, Z_t = 1: \quad L_{w,t} > T_{w,t,n} \rightarrow \sum_{s \le S} X_{w,t,s} > n \tag{3}$$

Figure 5: Tuple processing constraints.

$$\forall p,\, t: \quad H_{p,o_p,t} = \delta_{t,0} \tag{4}$$

$$\forall p,\, 0 < t < u,\, e_{w,x} \in p: \quad F_{p,x,t,u} = H_{p,w,t} \wedge C_{x,t,u} \tag{5}$$

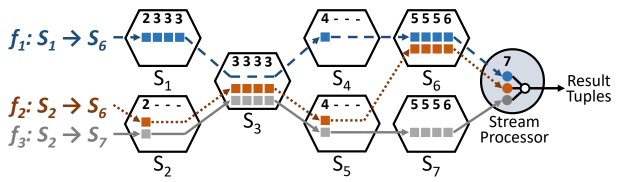
$$\forall p,\, u,\, e_{w,x} \in p: \quad H_{p,w,u} = 0 \rightarrow H_{p,x,u} = \sum_{t<u} F_{p,x,t,u} \tag{6}$$

$$\forall p,\, u,\, e_{w,x} \in p: \quad H_{p,w,u} = 1 \rightarrow H_{p,x,u} = 1 - \sum_{v \ne u} H_{p,x,v} \tag{7}$$

$$\forall w,\, Z_t = 1: \quad L_{w,t} = \sum_{p} N_{p,t} \cdot \bigvee_{u<t\le v} F_{p,w,u,v} \tag{8}$$

Figure 6: Path constraints.

The formulation divides into five parts: 1) *The target.* The target balances the average and the maximum number of tuples sent from edge switches to the stream processor by minimizing their sum, which is similar to that of UnivMon [21] and Sonata [12]; 2) *Switch hardware constraints.* Concerto guarantees the register usage is within the limit, and the order of operators is preserved among stages. Sonata also has similar constraints; 3) *Tuple processing.* For any input phase t, Concerto ensures that each switch only has one result phase u, operators from phase $t+1$ to u are installed at some stage, and there are enough tables for stateful operators. Figure 5 shows the constraints; 4) *Path.* For each path, a switch processes tuples from phase t to u, if and only if it has corresponding rules, and the previous switch in the path sends out tuples at phase t. Figure 6 shows the constraints. We use δ to stand for Kronecker delta, which is one if and only if the two subscripts are equal; 5) *Tuple reporting.* A



Flow	t_1, t_2	t_3, t_4	t_5	t_6, t_7	d_3	d_5
f_1	442628	50034	1033	25	3	3
f_2	1383594	113584	1739	36	4	3
f_3	307941	8874	2194	25	3	3
$f_1 + f_2$	1826222	163618	2772	61	5	3
$f_2 + f_3$	1691535	122458	3933	61	4	4
$f_1 + f_2 + f_3$	2134163	172492	4966	86	5	4

Figure 7: DDoS detection query planning on Concerto.

switch cannot report tuples at multiple phases except for those from parallel subqueries (e.g., `zip`). Compared to previous works using MIP, Concerto handles a more elaborate problem of letting switches cooperate.

3.4 Case Study: DDoS Detection

We now provide some intuition for the difference between Concerto and other systems through DDoS detection (Figure 2) query planning. In Figure 7, there are seven switches and three flows. We set each switch with four stages, and 0.5 Mb of registers at each stage, which is more strict than Barefoot Tofino [27]. We use the error rate of 1%, and generate the number of tuples at each phase (t_i) and the number of stages to perform stateful operators (d_j) from the trace. Note Concerto does not distinguish different flows for processing.

Concerto plans query according to the network configurations and the query analysis result (Figure 4). In Figure 7, numbers on the top of each switch represent operators installed at stages. We use squares to indicate that the switch processes tuples in a particular flow. As a result, with the cooperative query execution method, Concerto utilizes switch resources and only sends 86 tuples to the stream processor.

Existing systems cannot use resources on switches efficiently. For example, limited by the number of stages on edge switches, Sonata can only process tuples of f_1 to phase 3, sending over 1.7 million tuples to the stream processor. Although UnivMon can use all the switches from a global perspective, the resource of a single switch still restricts its operator placement. Thus, UnivMon can process tuples of f_1 and f_3 to phase 3 at S_1 and S_7, respectively, sending about 1.4 million tuples. By splitting the query to multiple switches, Concerto executes more operators on switches and reduces the stream processor's load by four orders of magnitude.

4 PRELIMINARY EVALUATION

We now use experiments to show that by splitting queries to multiple switches, Concerto reduces the workload on the stream processor. We use the number of tuples sent to the stream processor as the workload metric that we try to minimize, which is the same as Sonata [12].

Table 4: Evaluated topologies.

Topology	Number of Sites	Number of Links
Claranet	15	18
ATT North America	25	56
Cesnet-10	52	63
OTEGlobe	93	103

Table 5: Evaluated Concerto queries.

ID	Query	# Lines	Worst-Case Bits	
			PHV (Processing)	*Header (Transfer)*
1	Superspreader	6	123	72
2	Newly opened TCP	6	91	48
3	Port scan	7	107	56
4	DDoS	7	123	72
5	TCP incomplete flows	12	191	88
6	SSH brute force	6	139	88
7	Slowloris attacks	14	240	136

4.1 Evaluation Setup

Topology. We use representative real-world ISP topologies from the Topology Zoo dataset [18]. We set 20% of the ISPs to be edge switches. Table 4 shows the evaluated topologies. By default, we evaluate on the ATT North America topology,

Trace. We use CAIDA's anonymized packet trace [9], which was captured from an ISP's backbone link between New York and Sao Paulo. We randomly assign IP addresses and split the trace accordingly. As Sonata, we also replay the trace at 20× speed to emulate the edge switch's workload (20 Mpps on average) and use a time window of three seconds.

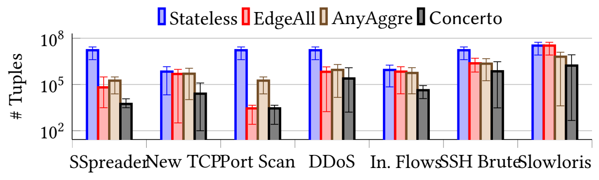
Application. Table 5 shows the evaluated queries from existing applications [4, 12, 30, 36–38]. Column *PHV* shows the bits required to process the whole query in one switch, while *Header* shows the maximum size of the tuple (e.g., phase 3 of DDoS detection). With only a small portion of packets carry a tuple, and the maximum size of a tuple is less than an IP header, the overall overhead is negligible.

Query compilation. We use simulated PISA switch to evaluate various settings. By default, we use an error rate of 1%, and a typical switch setting with 16 stages, 8 Mb of register memory per stage, and each operator can use up to half of the memory [12]. The core query planning contains 1,000 lines of Python code and uses Gurobi optimizer [13]. With only a few cases (a significant amount of concurrent queries) take more than 10 minutes, we set the time limit to 10 minutes.

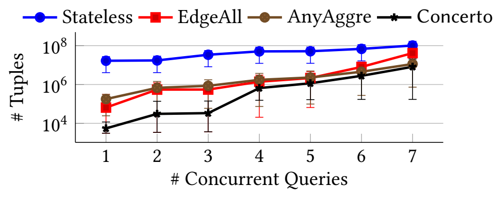
Compared systems. We divide six state-of-the-art systems into three categories (Table 6). As we are interested in the net effect of using different methods, we add constraints on MIP to simulate different systems. Note our evaluation result for

Table 6: Evaluated telemetry systems.

Method	Description	System
Stateless	Apply only stateless operators on switches.	Everflow [39], DREAM [22]
EdgeAll	Apply as many as possible operators on edge switches.	Sonata [12]
AnyAggre	Apply stateful operators at any switches and aggregate to the stream processor.	OpenSketch [36], UnivMon [21], Marple [26]

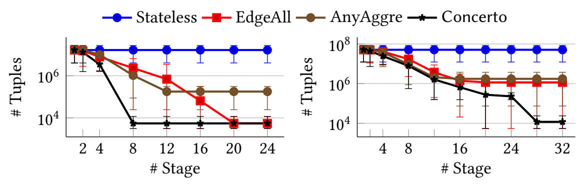


(a) Single-query performance on various queries.



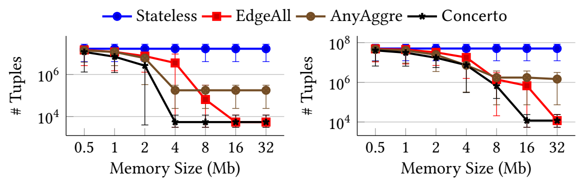
(b) Different numbers of queries.

Figure 8: Stream processor's workload on single queries and various numbers of concurrent queries.



(a) Single-query. (b) Multi-query.

Figure 9: Number of stages of PISA switch.



(a) Single-query. (b) Multi-query.

Figure 10: The register size of the PISA switch.

each method is superior to that of the represented systems as MIP removes redundant resource consumptions.

4.2 Performance of Different Queries

We now quantify how much Concerto reduces the stream processor's workload. The data point marks the average number of tuples sent to the stream processor by switches, and the error bar indicates the maximum and minimum values. Note as Concerto reduces the workload by a large portion, we use *logarithmic scales* on the y-axes.

Single query. As Figure 8(a) shows, comparing with the best previous methods for each query, Concerto reduces the stream processor's workload by as much as 19×, and 7.8× on average. Although AnyAggre places queries network-wide, it does not analyze queries carefully, forcing stateful operators to use the stream processor. EdgeAll can use edge switches to meet most of the resource requirements of a single query and supports stateful operators on the switch. Hence, it performs better than other existing methods. Note Sonata behaves worse than EdgeAll since it is limited to a single switch. The query analysis helps Concerto to perform stateful operators in network-wide telemetry, and the cooperative model utilizes the resources of multiple switches.

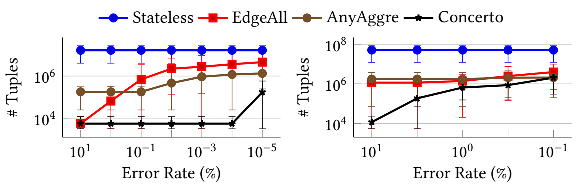
Multiple queries. As Figure 8(b) shows, Concerto is able to outperform existing methods by ~15× when there are a small number of concurrent queries. When there are seven concurrent queries, Concerto still reduces the workload by 28%. Different from single queries, AnyAggre performs better when there are many concurrent queries. The reason is AnyAggre distributes operators of different queries to multiple switches to utilize their resources. Concerto improves performance less on many concurrent queries since they use up critical resources such as stages in edge switches.

4.3 Robustness on Parameters

We now evaluate the robustness of Concerto under different switch hardware constraints and error rate requirements. We use the superspreader for single-query evaluation and four concurrent queries for multi-query evaluation.

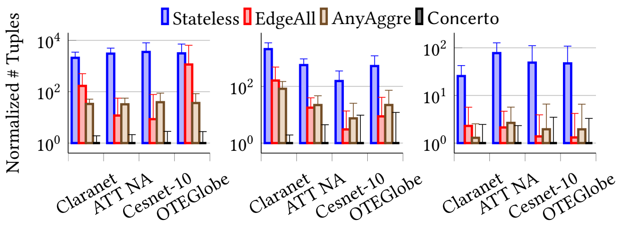
PISA switch constraints. The number of stages directly affects the number of operators on the switch. In the extreme case where the edge switch has unlimited stages, whether using non-edge switches does not matter. As Figure 9(a) shows, when there are few stages, Concerto performs well by utilizing all switches' resources. And when given a large number of stages, EdgeAll performs on par with Concerto. In the case of multi-query (Figure 9(b)), Concerto requires more stages to reduce the workload. Increasing the register size reduces the number of stages required to perform stateful operators. In both single-query (Figure 10(a)) and multi-query (Figure 9(b)) cases, we can observe similar results as changing the number of stages. Concerto provides performance gain under various resource constraints.

Error rate requirement. As Figure 11(a) shows, Concerto achieves a much lower error rate alongside a lower workload on the stream processor. For the single query, given the same workload, Concerto can achieve an error rate that is 10^4×



(a) Single-query. (b) Multi-query.

Figure 11: Error rate requirement.



(a) One query. (b) Two queries. (c) Four queries.

Figure 12: Normalized workload on various topologies with different numbers of concurrent queries.

lower. As for the multi-query case, Concerto reduces the workload by a relatively smaller amount, but can still reach a 10× lower error rate with the same workload. Requiring a lower error rate has a similar effect of reducing the register size. Thus it is unsurprising to find that the two subfigures of Figure 11 are mirror images of Figure 10(a) and Figure 10(b). Cooperatively utilizing resources from multiple switches enable Concerto to achieve far lower error rates.

4.4 Scalability

Topology size influences performance because larger networks tend to share the same switch with more paths. We use real-world WAN topologies of different sizes from the topology zoo [18]. We evaluate different numbers of concurrent queries to eliminate the influence of query load and observe the net effect of changing topology. For better comparison, we normalize the tuple number to that of Concerto.

We can observe from Figure 12 that the improvement of Concerto is stable among different topologies. Within each subfigure, the relative number of tuples to the stream processor does not change much with the topology, which suggests the topology has a limited impact on performance. Comparing different subfigures, we can find the improvement changes as the number of concurrent queries changes, meaning that the query has a more significant impact on performance. This proves the scalability of Concerto.

5 CONCLUSION AND FUTURE WORK

To analyze large amounts of traffic in real-time, network-wide telemetry systems need to reduce the workload on the stream processor. Concerto proposes a cooperative query execution model that reduces the stream processor's workload under the constraints of query requirements, switch hardware, and routing. Results on real-world trace show that by splitting queries to multiple switches, Concerto reduces the stream processor's workload by as much as 19×, and achieves an error rate of 10^4× lower than existing systems.

For future work, we plan to add timestamp support, which will enable Concerto to perform queries such as high latency detection [26]. Adding timestamp support requires redesigning the interface. Besides, we are also trying to make the query analysis more powerful by finding common parts of subqueries to reduce resource consumption.

ACKNOWLEDGEMENT

We would like to show great appreciation to our shepherd Qun Huang and anonymous reviewers for their feedback. This work is supported in part by the National Natural Science Foundation of China (NSFC) Grant 61532001 and gift funds from Huawei, Ant Financial, Nanjing Turing AI Institute, and the Zhongguancun Haihua Institute for Frontier Information Technology.

REFERENCES

[1] 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, Dandelion 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. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. 2016. SNAP: Stateful Network-Wide Abstractions for Packet Processing. In *Proceedings of the 2016 ACM SIGCOMM Conference.* Association for Computing Machinery, 29–43.

[3] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[4] Kevin Borders, Jonathan Springer, and Matthew Burnside. 2012. Chimera: A Declarative Language for Streaming Network Traffic Analysis. In *Proceedings of the 21st USENIX Conference on Security Symposium.* USENIX Association, 19.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[6] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.

[7] Cisco. 2020. NetFlow. https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[8] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[9] Center for Applied Internet Data Analysis (CAIDA). 2019. The CAIDA UCSD Anonymized Internet Traces - 201901. https://www.caida.org/data/passive/passive_dataset.xml.

[10] The Apache Software Foundation. 2020. Apache Flink. https://flink.apache.org/.

[11] The Apache Software Foundation. 2020. Apache Spark. https://spark.apache.org/.

[12] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, Association for Computing Machinery, 357–371.

[13] LLC Gurobi Optimization. 2020. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[14] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-Wide Heavy Hitter Detection with Commodity Switches. In *Proceedings of the Symposium on SDN Research*. Association for Computing Machinery, Article 8, 7 pages.

[15] Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 576–590.

[16] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. 2019. QPipe: Quantiles Sketch Fully in the Data Plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. Association for Computing Machinery, 285–291.

[17] Anurag Khandelwal, Rachit Agarwal, and Ion Stoica. 2019. Confluo: Distributed Monitoring and Diagnosis Stack for High-speed Networks. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 421–436.

[18] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The internet topology zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[19] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. USENIX Association, 311–324.

[20] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 334–350.

[21] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. Association for Computing Machinery, 101–114.

[22] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2014. DREAM: Dynamic Resource Allocation for Software-Defined Measurement. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. Association for Computing Machinery, 419–430.

[23] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2015. SCREAM: Sketch Resource Allocation for Software-Defined Measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. Association for Computing Machinery, Article 14, 13 pages.

[24] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and Precise Triggers in Data Centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*. Association for Computing Machinery, 129–143.

[25] Srinivas Narayana, Mina Tashmasbi Arashloo, Jennifer Rexford, and David Walker. 2016. Compiling Path Queries. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. USENIX Association, 207–222.

[26] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 85–98.

[27] Barefoot Networks. 2020. Tofino. https://www.barefootnetworks.com/products/brief-tofino.

[28] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[29] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. 2014. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 407–418.

[30] Vyas Sekar, Michael K Reiter, and Hui Zhang. 2010. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 328–341.

[31] sFlow.org. 2020. sFlow. https://sflow.org/.

[32] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research*. Association for Computing Machinery, 164–176.

[33] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying Datacenter Network Debugging with Pathdump. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, 233–248.

[34] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 561–575.

[35] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. 2019. DShark: A General, Easy to Program and Scalable Framework for Analyzing in-Network Packet Traces. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 207–220.

[36] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 29–42.

[37] Lihua Yuan, Chen-Nee Chuah, and Prasant Mohapatra. 2011. ProgME: towards programmable network measurement. *IEEE/ACM Transactions on Networking* 19, 1 (2011), 115–128.

[38] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. 2017. Quantitative Network Monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 99–112.

[39] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 479–491.