In-Network Splitter Function Enabling Highly-Parallel Event Stream Processing

Bochra Boughzala
University of Groningen
Groningen, The Netherlands

Boris Koldehofe
Technische Universität Ilmenau
Ilmenau, Germany

***Abstract*—Cloud data center workloads are comprised with a substantial volume of data-analytics tasks which rely heavily on data parallel stream processing. Splitting the data streams into independent data partitions and their distribution across multiple operators is a heavy operation than can become a performance bottleneck. Emerging in-network hardware accelerators in cloud infrastructures can enhance the performance of the splitter function and improve the parallelism degree of data-intensive cloud applications. Building on Programming Protocol-independent Packet Processors (P4), we devised a P4-based splitter fully in the data plane to benefit from line-rate performance of P4 packet processors, e.g., Tofino Switch. We present a summary of the splitter switch capabilities and complement our proposed design with new discussion points for future consideration.**

***Index Terms*—Data parallelism, P4 Programmable switches, Hardware acceleration, Event streaming systems.**

I. Introduction

Nowadays, data-analytics tasks represent major contributors in cloud data center workloads. The increasing volume of data coming from an ever growing number of data producers, e.g., IoT and sensing devices, is creating a snowball effect on data-analytics cloud applications. Parallel stream processing [1] is a powerful paradigm for coping with increased data workloads and speeding up the execution of data-analytics tasks by deploying them concurrently over replicated operator instances. The key component of the data parallelization framework utilized in streaming systems is the *splitter* (Fig. 1). Its role consists of partitioning infinite event streams into finite sub-sets of events and dispatching the event partitions, called *windows*, across a dynamic set of operator instances.

Existing streaming systems, e.g., Apache Flink, Apache Kafka, rely exclusively on traditional Central Processing Unit (CPU) and Graphics Processing Unit (GPU) programming models to enable data parallel operator execution using host-based software implementation of the splitter. However, these approaches have limited performance especially for cloud deployments involving multiple large-scale data-analytics applications running simultaneously and dealing with a huge number of real-time data streams. On one hand, the increased number of data sources requires higher splitter ingestion rate in processing the growing workloads of incoming data streams. On the other hand, increasing the degree of parallelism, i.e., the number of operator instances, to keep up with the higher demands does not guarantee an improved throughput if the serving rate of the splitter is the bottleneck. To scale the system performance, conventional methods dedicate more CPU cores for the splitter execution. However, there is a hard limit to how many cores are available on a single machine leading to bounded parallelism degree in CPU-based solutions, e.g., one splitter serving 64 operators [2].

Emerging network hardware accelerators are becoming commonly available in modern cloud infrastructures in addition to the existing CPUs and GPUs. In-Network Computing (INC) emerged as a new paradigm closely aligned with network softwarization and enabled by the advancements made in Software-Defined Networking (SDN) and Network-Function Virtualization (NFV). It enables custom network functions to be executed along the communication path by building on available network resources. An important motivation for in-network computing is to reduce the latency with shorter paths, save network bandwidth, and benefit from specialized network hardware accelerators, e.g., packet processors.

P4 [3] is a programming model that supports in-network computing by bringing the flexibility for defining new packet headers and describing the dependencies in packet processing. Particularly, P4 programmable switches, e.g., Tofino Switch [5], enable the network softwarization of performance-critical functions, e.g., the splitter, and their execution right in the data plane at terabit speed.

Motivated by the performance benefits of network-centric processing for data analytics [7], we devised a P4-based splitter switch (initial version P4SS [8] and later version S4 [9]) to speed up the splitter performance using P4 programmable switches. In this paper, we present a summary of our approach and complement our design with new discussion points.

II. In-Network Splitter Function

We present briefly the key ideas of our approach. The details of the proposed in-network splitter function is described in [9].

A. System Architecture

Data parallel processing is important for increasing the scalability and throughput in distributed event streaming systems. A data parallelization framework often builds on the *splitter-merger* architecture (Fig. 1) to enable the parallel operator execution over different subsets of the data. The splitter receives the incoming data streams and applies a particular splitting strategy involving one of the various *windowing semantics* in streaming systems, e.g., count-based

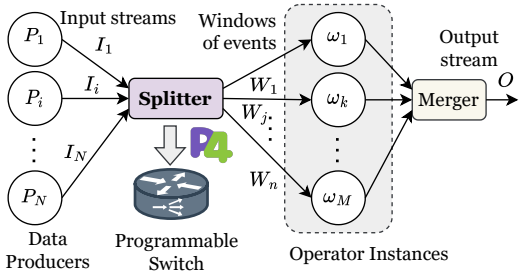

Fig. 1. Offloading the splitter function to a P4 programmable switch in a splitter-merger data parallelization framework.

sliding windows. The splitter divides the input streams into independent units called *windows*. The splitter also assigns the computed windows to the operator instances to ensure the load-balancing using a scheduling mechanism such as Round Robin (RR). The operator instances are a replication of the same computation logic, e.g., pattern detection, which they perform on the *windows* of events dispatched by the splitter. The merger aggregates the computation outcome from the different operators into an outgoing stream. In the described system, the parallelism degree is the maximum number of operator instances which can be served by the splitter and it determines the overall system throughput. Therefore, the splitter must be very efficient in processing the incoming event streams and performing the stream splitting and window scheduling logic.

B. Window-based Stream Splitting Logic

To offload the splitter function to a P4 programmable switch, we have to support the essential *windowing semantics* of event streaming systems, directly in the data plane. Since P4 does not have built-it support for window-based operations, we leverage stateful processing, i.e., stateful Arithmetic Logic Unit (ALU)s and registers to devise the data path algorithm of four essential windowing semantics which are the most common in stream processing systems. The proposed in-network splitter function design is depicted in Fig. 2.

The supported window types are :

- Count-based windows are defined by a size n and shift δ. If an event corresponds to a P4 packet and $\delta = 1$, then a new window starts for every incoming packet. An ongoing window closes after n packets.
- Time-based windows are defined by a duration Δ and a shift δ in time measures. The time reference can be anchored as a timestamp in the packet itself or from the internal switch processing time. The start of the first window is triggered by the arrival of the first packet. The window ends after Δ period and the next window starts after δ time.

Both window types can be defined according to one of the following windowing scheme :

- Tumbling (non-overlapping) windows result in windows with disjoint events mapped to different operators. Since there is no overlap, there is always a one-to-one mapping between an event packet and an operator instance. They are easier to implement than sliding windows as they require less state to maintain.
- Sliding (overlapping) windows result in multiple windows with shared events. Due to the overlap an event can be part of multiple windows at the same time. Since there is more than one active window at a given time, sliding windows are more challenging to implement and require more state variables.

The state management of ongoing windows of the different streams and their consistent mapping to the right operator instances is the most challenging part of the proposed splitter function. To ensure the consistency of the splitting logic, state variables are maintained using registers for window tracking of individual streams.

C. Window Scheduling Logic

The P4 splitter switch distributes the incoming workloads across multiple operators using a round-robin scheduling mechanism. The load-balancing of each individual stream is handled within its own parallelism degree and independent from the other streams.

III. Implementation

We developed the splitter function in the $P4_{16}$ programming language [4] targetting Tofino1 [5] and using the Open P4Studio [6], i.e., Barefoot Networks / Intel's Software Development Environment (SDE). We release the source code of our P4 splitter switch implementation and make it available in Github [1]. By providing the P4 implementation of the splitter, future research can expand on devising further more complex windowing semantics and advanced window scheduling techniques, e.g., Weighted Round Robin (WRR). Beside research, the released implementation can be used in academia to experiment with the tofino model, inspect its internal behavior and have a better understanding of stateful processing.

IV. Discussion

A. Handling Out-of-Order Events

The order of events is important in window-based operations, e.g., Complex Event Processing (CEP), since it has an influence over the detection of meaningful situation or the absence of it, i.e., false positives/negatives. For instance, in a count-based window of 3 events where a pattern recognition operator is searching for the sequence of events $\{A, B, C\}$, an out-of-order event may result in a false negative if the pattern occurred in the real world. Packets of a data stream may arrive out-of-order to the splitter due to taking different paths with varying network delays. When out-of-order events happen we discuss the following reordering mechanisms to be used along the splitter depending on the sought ordering guarantees.

[1] https://github.com/rug-ds-lab/splitter-switch

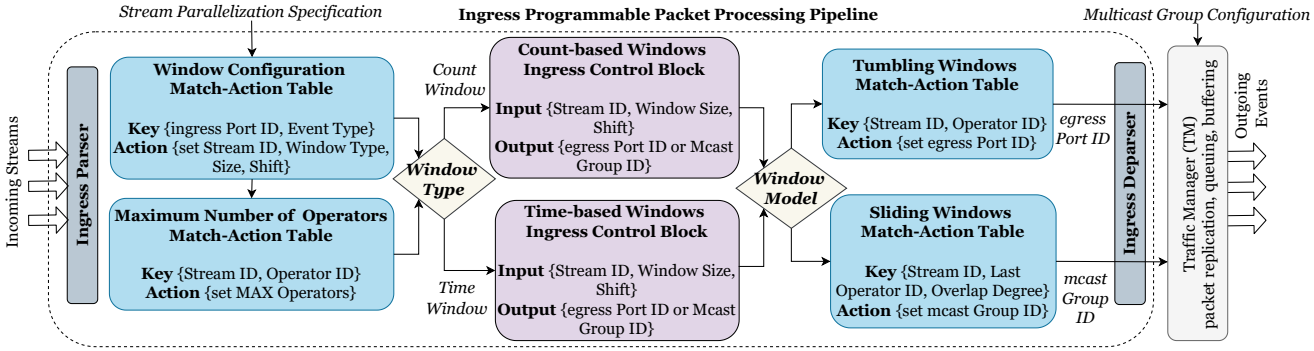

Fig. 2. Overview of the splitter function implementation over a P4 packet processing pipeline.

Causal Order. When an event $(e_1, t_1 = 1)$ *happened before* $(e_2, t_2 = 2)$ but the splitter receives the event e_2 before e_1, (and it has already delivered an event $(e_0, t_0 = 0)$), the splitter can detect a gap by comparing every pair of two consecutive events within each stream and verify that their sequence numbers are increased by 1 in each step. When a packet from the future arrives at the splitter $(t_2 - t_0 = 2)$, while an older packet is late, the splitter can purposefully delay e_2 by using the recirculation path (repeatedly) as a reordering mechanism and then checking the condition until e_1 is delivered. However, a fallback mechanism is needed in case the delayed event e_1 is lost, and all consecutive events are endlessly recirculated inside the switch. For instance, a counter can be added to limit the recirculation to $c = 3$ attempts after which the event e_2 will be delivered before the current window closes. These mechanisms require additional state variables to store and manage which will impact the splitter hardware resources consumption.

Total Order. When an operator receives an event e_1 before e_2, then all the operators receiving the events e_1 and e_2 will receive them in the same order. Total order guarantees are relevant for the case of sliding windows when events overlap across multiple operators. In such case, the P4 splitter switch can be used together with a total ordering service, such as P4mCast [10], a P4-based atomic multicast protocol to provide total order guarantees at line rate.

B. Handling Single Point-of-Failure

In the proposed approach, the splitter function is dependent on a single switch which introduces the risk of a Single Point-of-Failure (SPOF). In case of a failure, e.g., switch crash, the entire streaming system becomes unavailable. For the purpose of fault-tolerance, we can replicate the splitter function over multiple P4 switches using a replication protocol and a primary-secondary model, i.e., leader-follower roles. A single *active* switch takes the primary role and handles the splitting function over the ongoing data streams. The backup *passive* switches keep the state updates of the currently active windows and their mapping to the operator instances. If the primary splitter switch fail, one of the backup switches take over using consensus-based re-election protocol.

V. Conclusion and Future Work

With network softwarization becoming more widespread in modern cloud data centers, data-analytics applications can benefit from network hardware accelerators such as P4 programmable switches. In this work, we propose a P4-based splitter switch to address the performance bottleneck in data parallel stream processing systems. First, the P4-based splitter offers the flexibility to configure distinct data streams each with its own individual parallelization model. Seconds, it enables great performance and scalability with its highest achievable parallelism degree being up to $500k$ operators [9].

For future work, we would study the implications of the integration of in-network splitting function with existing stream processing systems such as Apache Flink. We would also extend the current evaluation with concrete data-analytics applications and real-world dataset.

References

[1] H. Röger and R. Mayer, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–37, Jun. 2019.

[2] M. Amann, "Efficient splitter for data parallel complex event processing," Bachelor's thesis, 2018.

[3] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[4] P4 Language Consortium, "$P4_{16}$ Language Specification," Feb. 2025. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html.

[5] Intel®, "Tofino1 6.5 Tbps," Feb. 2025. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html.

[6] P4 Language Consortium, "Tofino Open P4Studio," Feb. 2025. [Online]. Available: https://github.com/p4lang/open-p4studio.

[7] B. Boughzala and B. Koldehofe, "Accelerating the performance of data analytics using network-centric processing," in *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*, Jun. 2021, pp. 192–195.

[8] B. Boughzala, C. Gärtner, and B. Koldehofe, "Window-based parallel operator execution with in-network computing," in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*, Jun. 2022, pp. 91–96.

[9] B. Boughzala and B. Koldehofe, "In-network management of parallel data streams over programmable data planes," in *IFIP Networking 2024*, Apr. 2024.

[10] B. Boughzala and B. Koldehofe, "Poster: In-network total order guarantees supporting state machine replication with P4 programmable switches," in *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, Oct. 2024, pp. 1–3.