A Control-based Approach Towards Adaptive Stream Processing

Student: Luo Mai
Advisors: Kai Zeng, Rahul Potharaju, Paolo Costa, Sriram Rao
Imperial College London, Microsoft
luo.mai11@imperial.ac.uk, {kaizeng, rapoth, pcosta, sriramra}@microsoft.com

CCS Concepts **• Software and its engineering → Software system structures**

Keywords Dataflow System, Stream Processing, Data Ingestion, Control System, Real-time Analytic

1. Motivation

Large-scale Internet-service providers such as Amazon, Google, Facebook, and Microsoft generate tens of millions of data events per second (Bailis et al. 2017). To handle such high throughput, they have traditionally resorted to offline batch systems, e.g., Spark SQL (Armbrust et al. 2015) and Hadoop MapReduce (Dean and Ghemawat 2008). More recently, however, there has been an increasing trend towards switching to online streaming systems to ensure timely processing and avoid the delays incurred by batching (Jindal et al. 2017; Meehan et al. 2017; Abraham et al. 2013).

Fully achieving the benefits promised by these online systems, however, proved particularly challenging. To start with, event-based workloads exhibit high *temporal* and *spatial* variability, up to an order of magnitude compared to the average load (Kulkarni et al. 2015; NetFlix 2016). Further, due to the large number of servers involved, failures and hardware heterogeneity makes it hard to ensure stable and predictable performance. Together these issues significantly complicate resource provisioning, forcing system administrators to over-provision resources, with the obvious negative consequences on cost and complexity.

An alternative and more efficient approach would be to dynamically modify the system reconfiguration (e.g., by adding/removing resources or by redistributing operators) whenever the workload or the environment changes. This would entail extending the streaming platform with a set of control policies and mechanisms that can reconfigure the setup based on the new conditions. Such approach, however, requires addressing four main requirements.

Programmability The control plane should be able to support a heterogeneous set of policies, providing sufficient hooks to monitor the current performance and supporting flexible triggers to react accordingly;

Easy-to-understand abstraction The control plane should make it easier for developers to create new custom policies, using a simple and intuitive interface;

Zero downtime The control plane should able to reconfigure systems without downtime. Taking system offline incurs data loss and delays that can negatively affect downstream computation (Goel et al. 2014);

Scalability and low-overhead The control plane should be able to support an arbitrarily large number of policies and should add negligible runtime overhead.

To the best of our knowledge, none of the existing streaming systems fully addresses the requirements above. Heron (Kulkarni et al. 2015) and Flink (Carbone et al. 2015) have monolithic control planes, lacking a well-defined control-plane API, which makes it hard to implement new control policies. Further, they rely on a centralized controller architecture, which suffers from scalability limits. Therefore, Heron and Flink support only a small set of predefined control policies (dynamic scaling and back pressure). Moreover, during reconfigurations, these systems have to freeze data processing and save / recover intermediate state, resulting in noticeable system downtime. Spark Streaming (Zaharia et al. 2013), instead, adopts a micro-batching-style architecture in which a set of events is buffered and processed as a batch. While this model allows to modify a dataflow across batches, it has limited flexibility due to the hard batch boundaries and incur high runtime overhead due to the synchronization and scheduling operations required.

2. Our Approach

To address the shortcomings of today's systems, we propose a novel design that integrates the data- and control-plane. The key idea is to treat control events as data events, adopting a shared programming and management style. Control events are interposed in between data events and are propagated as part of the data event stream. This yields several key advantages. First, by leveraging the existing data pipeline, control events can be streamed at high rate and in a scalable fashion, without requiring any ad hoc mechanism. This enables running multiple control policies concurrently (e.g., in a multi-tenant scenario or for A/B testing of new policies), without impacting the overall system performance. Second, adding control events right after data events provides an intuitive way to logically define the boundaries of data events to which the control events are applied. Thanks to the reliable and in-order delivery provided by the underlying data channel, this greatly simplifies control management because no coordination is required to ensure that no events have been missed by the control operations. Finally, along with *local-blocking* events, our design also supports *non-blocking* control events, which are processed and transferred asynchronously, thus avoiding expensive global synchronization and high runtime overhead. This makes it easy to efficiently implement global operations (e.g., snapshotting), which typically entail costly synchronization steps.

Control actions are triggered asynchronously on the different dataflow nodes depending on spatial and temporal conditions. Control events are defined using an extensible interface inspired by the reactive programming paradigm. This significantly reduces the burden on developers when implementing new control policies as they do not have to learn a separate API or rely on different management tools. Our simple API can encode a wide range of control policies, including stateful dynamic scaling, hot query refinement, non-blocking checkpoints, and asynchronous replays. These events require local synchronization for multiple channel inputs so that they can preserve the order of concurrent control policies (i.e., local-blocking). In addition, we also developed the control events for batch completion, watermark advancement, heartbeat, and statistics collection. These events are tolerant to processing order and thus do not require local synchronization (i.e., non-blocking).

To validate our claims and evaluate the runtime performance of our new control-plane design, we implemented it on top of *Flare*, a new distributed streaming dataflow system that is incubated within Microsoft. Flare exposes traditional stream analytic operators including windowing, group-by aggregates, filters, stream joins and user-defined functions. These operators can be combined to express complex event processing logic and then translated into a streaming dataflow. Multiple data flows are managed by parallel controllers. They are independently monitoring performance and executing control policies, thus establishing concurrent feedback-loop controls. To achieve performance, Flare is built on top of Orleans (Bernstein et al. 2014), a scalable distributed actor framework, and uses Trill (Chandramouli et al. 2014) as the underlying stream processing engine.

While we chose to showcase our approach on top of Flare for ease of implementation and deployment on our internal clusters, our design is not tied to a specific platform and, with modest additional engineering effort, it can be ported to existing systems such as Flink, which supports checkpoint messages that can serve as the starting point for implementing the control events proposed by this paper.

3. Current Status

We are finalizing the implementation of Flare and in the process of deploying it onto a production ingestion cluster, comprising hundreds of servers. In the following, we outline the control policies that we have developed thus far and report on the results of our preliminary experiments on an Azure VM cluster.

Statistic collection. We implemented a control policy to collect statistics generated by each data flow node and propagate them with the data stream to the controller, which then merges all of them to generate aggregate statistics. In our Azure testbed, transferring control events incurs microsecond-level latency within the local server and millisecond-level latency across servers.

Batch control event We developed a control policy that dynamically adjusts the data batch size to meet a latency threshold. A large batch amortizes per-event runtime overhead but incurs additional buffering latency. We investigate the performance of a data flow using different batch sizes. We then collect all aggregates and merge them using a map-reduce parallel execution plan. We run the Yahoo Streaming Benchmark (Chintapalli et al. 2016) using 16 servers with 4 vCPU and 28 GB RAM that continuously receive events from the network. With a small batch size (100 events) this cluster processes 3.2 millions events/s with a processing latency of 232 ms. Increasing the batch size to 5,000 achieves a throughput of 25.1 millions events/s but at the cost of a latency of 489 ms.

Topology control policy Flare uses topology control events not only for expanding or shrinking a single dataflow processing stage to cope with variable load, but also for adjusting a query plan by inserting or deleting a stage during uptime. Unlike existing systems such as Heron, Flink or Spark Streaming, which require to suspend the system when reconfiguring the query plan, Flare relies on non-intrusive topology control events to enforce distributed asynchronous modification, making negligible impacts towards processing throughput. Since control events are embedded in the data stream, it is easy to provide global snapshotting. This enables the system to identify the data events processed by the old and new topology. This further allows the controller to correctly handle data dependencies and perform source replays in case of failures.

References

L. Abraham, J. Allen, O. Barykin, V. Borkar, B. Chopra, C. Gerea, D. Merl, J. Metzler, D. Reiss, S. Subramanian, et al. Scuba: diving into data at Facebook. *Proceedings of the VLDB Endowment*, 6(11):1057–1067, 2013.

M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark SQL: Relational data processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 541–556. ACM, 2017.

P. Bernstein, S. Bykov, A. Geller, G. Kliot, and J. Thelin. Orleans: Distributed virtual actors for programmability and scalability. Technical report, MSR, March 2014.

P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. C. Platt, J. F. Terwilliger, and J. Wernsing. Trill: A high-performance incremental query processor for diverse analytics. *Proceedings of the VLDB Endowment*, 8(4):401–412, 2014.

S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1789–1792. IEEE, 2016.

J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

A. Goel, B. Chopra, C. Gerea, D. Mátáni, J. Metzler, F. Ul Haq, and J. Wiener. Fast database restarts at facebook. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 541–549. ACM, 2014.

A. Jindal, J. Quiané-Ruiz, and S. Madden. INGESTBASE: A declarative data ingestion system. *CoRR*, abs/1701.06093, 2017. URL `http://arxiv.org/abs/1701.06093`.

S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *SIGMOD Conference*, 2015.

J. Meehan, C. Aslantas, S. B. Zdonik, N. Tatbul, and J. Du. Data ingestion for the connected world. In *CIDR*, 2017.

NetFlix. Stream-processing with Mantis, 2016. `https://medium.com/netflix-techblog/stream-processing-with-mantis-78af913f51a6`.

M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM, 2013.