Seamless State Management for Distributed Stream Processing Framework

Pritish Mishra

1 Motivation

Stream processing frameworks were being traditionally built for Big-data use-cases. A common assumption for these use-cases was centralised processing of all data, i.e., in the cloud datacenters. However, with the proliferation of IoT devices, most of the data produced today are at the edge of the network. Also, these applications demand very low latency. Hence, stream processing frameworks have to support such distributed systems by bringing computation closer to the data sources [1].

However, the solution isn't that simple. Devices at edge are constrained in resources. The application requirements also change from time to time. A new data source could appear or an existing data source could move from one location to another. Such scenarios demand a reconfiguration of the system and the application modules (henceforth referred to as *operators*) need to moved from one location to another to accommodate the changing needs.

For stateful applications, the problem is even more complicated. Reconfiguration for a stateful operator includes moving the accompanying state from one location to another. Since the data is continuously being processed and that cannot be stopped or reduced, this movement of state must be done while ensuring data consistency.

Current state-of-the-art stream processing frameworks don't have an optimum strategy for handling such reconfigurations seamlessly. While many of the stream processing frameworks like Apache Flink [2], Apache Storm [3], Spark Streaming [4] and Turbine [5] don't have a seamless reconfiguration strategy at all and rely on stop-restart, there are some others like StreamCloud [6], SEEP [7], IBM Streams [8] , FUGU [9, 10], Megaphone [11] and Rhino [12] that pause the processing of data for a small period of time during the migration (Pause-Resume). While better than Stop-Restart strategy, Pause-Resume strategy still causes a spike in the latency of data processing to accommodate state migration. Hence, there is a need of Live Migration strategy. To best of our knowledge, only one solution, Chronostream [13] proposes a live migration strategy. But, there are many issues with the strategy like data consistency,

2 Idea

The main idea of the project revolves around creating a stream processing framework for managing stateful stream applications. Inheriting the programming model from the research literature [14], the framework's novel contribution will be seamless reconfiguration and state migration. Then, the plan is to find a small benchmark application to first test if the system actually works. Finally, an actual IoT benchmark like [15] can be used to test the performance of the system.

The aims of the project are:

- First, to build a state management system that provides basic support to stateful applications.
- Second, design a basic partitioning strategy for indexing the state. This can be used later to move / distribute some amount of state amongst the operators.
- Third, design a storage mechanism. The state cannot be stored in memory. So, it must be periodically backed up in a persistent storage. This state's backup can then be moved from one location to another during reconfiguration.
- Fourth, design the actual reconfiguration mechanism. This will involve designing components for both backup of state on original location and restore of state on new location. However, the restore mechanism should be accompanied with a strategy that continues the processing of data all throughout this process.
- Fifth, conduct experimental evaluations using a IoT benchmark to show the performance of the system. Ideally, the system should produce exactly same throughput and latency before, during and after a state reconfiguration. This will prove that our system is seamless. Experiments should also ensure that the data remains consistent before and after the reconfiguration.

References

[1] Sajjad, Hooman Peiro, Ken Danniswara, Ahmad Al-Shishtawy, and Vladimir Vlassov. “Spanedge: Towards unifying stream processing over central and near-the-edge data centers.” In 2016 IEEE/ACM Symposium on Edge Computing (SEC), pp. 168-178. IEEE, 2016.

[2] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas. “Apache Flink: Stream and batch processing in a single engine.” IEEE Data Engineering Bulletin, 38, 2015

[3] Apache Storm. http://storm.apache.org/

[4] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. “Structured streaming: A declarative API for real-time applications in Apache Spark.” In SIGMOD, 2018.

[5] Y. Mei, L. Cheng, V. Talwar, M. Y. Levin, G. Jacques-Silva, N. Simha, A. Banerjee, B. Smith, T. Williamson, S. Yilmaz, et al. “Turbine: Facebook’s service management platform for stream processing.” Traffic, 40:80.

[6] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente, and P. Valduriez. “StreamCloud: An elastic and scalable data streaming system.” IEEE Transactions on Parallel and Distributed Systems, 23(12):15, 2012.

[7] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. “Integrating scale out and fault tolerance in stream processing using operator state management.” In SIGMOD, 2013.

[8] B. Gedik, S. Schneider, M. Hirzel, and K. L. Wu. “Elastic scaling for data stream processing.” IEEE Transactions on Parallel and Distributed Systems, 25(6):17, 2014.

[9] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer. “Latency aware elastic scaling for distributed data stream processing systems.” In DEBS, 2014.

[10] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. “Auto-scaling techniques for elastic data stream processing.” In ICDE Workshops, 2014.

[11] M. Hoffmann, A. Lattuada, F. McSherry, V. Kalavri, J. Liagouris, and T. Roscoe. “Megaphone: Latency-conscious state migration for distributed streaming dataflows.” In VLDB, 2019.

[12] B. Del Monte, S. Zeuch, T. Rabl, and V. Markl. “Rhino: Efficient management of very large distributed state for stream processing engines.” In SIGMOD, 2020.

[13] Y. Wu and K.-L. Tan. “ChronoStream: Elastic stateful stream computation in the cloud.” In ICDE, 2015.

[14] Alexandrov, Alexander, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao et al. “The stratosphere platform for big data analytics.” The VLDB Journal 23, no. 6 (2014): 939-964.

[15] RIoTBench: A Real-time IoT Benchmark for Distributed Stream Processing Platforms, Anshu Shukla, Shilpa Chaturvedi and Yogesh Simmhan, “Concurrency and Computation: Practice and Experience”, Volume 29, Issue 21, 2017, doi:10.1002/cpe.4257