Asynchronous Stream Data Processing using a Light-Weight and High-Performance Dataflow Engine

Vinu E. Venugopal
Faculty of Science, Technology and Medicine
University of Luxembourg
Belval, Luxembourg
vinu.venugopal@uni.lu

Martin Theobald
Faculty of Science, Technology and Medicine
University of Luxembourg
Belval, Luxembourg
martin.theobald@uni.lu

In the last decade, various *distributed stream processing engines* (DSPEs) were developed in order to process data streams in a flexible, scalable, fast and resilient manner. Coping with the increasing high-throughput and low-latency requirements of modern applications led to a careful investigation and re-design of new tools for stream processing. The first generation of tools, such as Apache Hadoop [15], Spark [16], Storm [14] and Kafka [10], were designed to split an incoming data stream into batches and to then synchronously execute their analytical workflows over these data batches. To overcome the limitations—primarily, the high latency—of this iterative form of *bulk-synchronous processing* (BSP), *asynchronous stream-processing* (ASP) engines such as Apache Flink [13] and Samza [11] have also recently emerged.

To guarantee a high *sustainable throughput* (ST), systems that rely on direct worker-worker-based communication protocols [6], [9] were proposed in contrast to the prevalent driver-worker architectures. One upper hand of these systems, compared to all of the aforementioned ASP and BSP engines, is the complete avoidance of hidden synchronization barriers and the constant need of state exchange (and hence communication overhead) between a dedicated driver and its worker nodes. However, most of these efforts are still limited to scale-up oriented architectures, and no serious advancement in terms of the supported programming abstractions has been developed in the last years for these engines. For example, in ST-oriented systems such as FastFlow [1], GrPPi [3], Streambox [7] and PiCo [8], only very basic parallel programming patterns for stream data processing were proposed. However, these patterns are general and not specifically tailored for modern streaming analytics. All of these systems remain prototypes, providing only basic support for defining more complex DAGs of dataflow operators from the traditional streaming algebra.

In this talk, we will give an overview of our new DSPE architecture, called *"AIR"*, which can readily be deployed for ST-oriented applications in an efficient and scalable manner. AIR is based on a novel communication protocol among the worker nodes, which we refer to *"Asynchronous Iterative Routing"*, to process one or more incoming data streams in a parallel and asynchronous manner. AIR has been developed from scratch in C++ and is purely based on the Message Passing Interface (MPI) in order to facilitate a low-level and highly efficient communication protocol among the worker nodes.

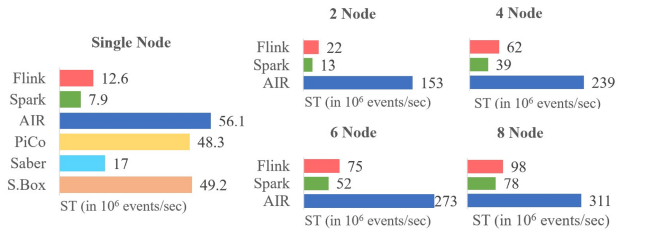

Fig. 1. Comparison of the sustainable throughput (ST) of various SPEs and DSPEs on a single-node setup (Intel Xeon Platinum 8260 CPU with 2.40GHz, 24 physical cores, 32 KB L1 cache and 1024K KB L2 cache), and on multi-node setup (where each node is equipped with two 2.6 GHz Intel Xeon Gold 6132 CPUs, 28 cores per CPU and 128GB RAM), for the Yahoo! Streaming Benchmark [2].

Our experiments based on various streaming benchmarks confirm that AIR scales out much better than existing distributed SPEs to clusters consisting of up to 8 nodes and 224 cores. In the YSB setting (as shown in Fig 1), AIR performs up to 15 times better than Spark and up to 5.8 times better than Flink in terms of ST. On a single node, AIR exhibits an improvement of a factor of 5–6 in performance over the Java-based SPEs (Saber, Spark and Flink) which is typically due to their less efficient Java (versus C++) implementations, the increased overhead of various API layers, and less efficient CPU utilization. However, the HPC-optimized C++-based SPEs, such as PiCo and StreamBox, show a performance close to the ST obtained using our AIR dataflows on a 1 node setup.

AIR fills an important gap among DSPEs on an HPC infrastructure by providing the ability to process streams asynchronously and by utilizing the underlying resources more efficiently via higher task-level parallelism and multi-threading. We believe that, with the design of AIR, we found a good compromise for a light-weight, reduced design of a DSPE that exhibits good performance and scales well also to larger cluster deployments. As for future research topics, we also

plan to investigate new mechanisms to include more explicit forms of workload balancing and fault tolerance directly into the driver-less architecture of AIR.

AIR is available as open-source release at our GitHub repository [12]. More details about the architecture of AIR and detailed experimental results (including a comparison with the modern scale-up SPEs) can be found at [5]. Our initial efforts that paved the way to developing this framework is detailed in [4].

REFERENCES

[1] Marco Aldinucci, Sonia Campa, Marco Danelutto, Peter Kilpatrick, and Massimo Torquati. Targeting distributed systems in fastflow. In *Euro-Par 2012: Parallel Processing Workshops*, pages 47–56, 2012.

[2] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *IPDPSW*, pages 1789–1792, 2016.

[3] David del Rio Astorga, Manuel F. Dolz, Javier Fernández, and José Daniel García. A generic parallel pattern interface for stream and data processing. *Concurr. Comput. Pract. Exp.*, 29(24), 2017.

[4] Vinu E. Venugopal and Martin Theobald. Benchmarking synchronous and asynchronous stream processing systems. In *CoDS-COMAD 2020: 7th ACM IKDD CoDS and 25th COMAD, Hyderabad India, January 5-7, 2020*, pages 322–323. ACM, 2020.

[5] Vinu E. Venugopal, Martin Theobald, Samira Chaychi, and Amal Tawakuli. AIR: A light-weight yet high-performance dataflow engine based on asynchronous iterative routing. In *SBAC-PAD 2020, Portugal, September 8-11, 2020*, pages 51–58, 01 2020.

[6] Frank McSherry, Derek Gordon Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow. In *CIDR*, 2013.

[7] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S. McKinley, and Felix Xiaozhu Lin. StreamBox: Modern Stream Processing on a Multicore Machine. In *USENIX*, pages 617–629, 2017.

[8] Claudia Misale, Maurizio Drocco, Guy Tremblay, Alberto R. Martinelli, and Marco Aldinucci. Pico: High-performance data analytics pipelines in modern C++. *Future Generation Comp. Syst.*, 87:392–403, 2018.

[9] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *SOSP*, pages 439–455, 2013.

[10] Neha Narkhede, Gwen Shapira, and Todd Palino. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*. O'Reilly Media, Inc., 2017.

[11] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. Samza: Stateful scalable stream processing at linkedin. *PVLDB*, 10(12):1634–1645, 2017.

[12] AIR team. Github link: https://github.com/bda-uni-lu/air, Last Accessed 29/08/2020.

[13] Flink team. Download link: https://flink.apache.org/, Last Accessed 21/08/2020.

[14] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In *SIGMOD*, pages 147–156, 2014.

[15] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[16] 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 *NSDI*, pages 2–2, 2012.