Correctness in Stream Processing: Challenges and Opportunities

Caleb Stanford
castan@cis.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Konstantinos Kallas
kallas@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Rajeev Alur
alur@cis.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Today's real-time data analytics applications are built using a range of software platforms for distributed stream processing. Popular stream processing platforms include Apache Storm [1], Spark [2, 3], and Flink [4]; Google Cloud Dataflow [5]; Microsoft Trill [6]; and emerging frameworks such as Timely [7, 8] and Differential Dataflow [9]. However, engineering and performance advances over the last two decades have not been met by adequate attention to software correctness. Correctness is especially important in this context because the amount of data, distributed deployment, and real-time nature of these applications makes them difficult to understand and to debug [10–12]. Moreover, errors are catastrophic: whereas an error in an offline application might go unnoticed if it is diagnosed and fixed in a timely manner, an error in a streaming application immediately results in either wrong results, delays, or service outages for downstream consumers. To ensure the highest level of safety for present and future applications, we advocate for formal methods work in the rigorous formalization and verification of stream processing programs and systems.

Challenges. Before we can achieve verified applications, researchers and practitioners must agree on what it means for stream processing programs to be correct. Unfortunately, this remains an outstanding challenge: there is no unifying language standard, specification, or semantics that is understood across systems. For example, a stream processing program is typically taken to be a dataflow graph of operators, but systems disagree on whether edges in the graph can be ordered streams, or whether all data may be out-of-order. The details of how streams are partitioned between graph operators is also system-dependent. To further complicate matters, modern stream processing applications may support a number of complex features, including user-defined stateful operators [1, 13, 14], communication across partitions [15], querying or interfacing with external services [16, 17], and iterative computation [8].

Broader context. In contrast to today's stream processing applications, database query engines and batch processing applications often benefit from formal semantics built on relational algebra that is well-understood and agreed upon, leading to fruitful research on semantically predictable query languages, optimization, and distributed evaluation. In distributed systems, formal specifications can be exploited in order to prove systems correct under faults, to prove safety through model-checking or to test correctness at runtime using traces. Formalization in these areas has enabled verification, testing, optimization, and synthesis.

Opportunities. We identify four *correctness dimensions* which are common to all stream processing platforms, regardless of specific system choices and features, and represent important opportunities in this space. First, stream processing applications process both out-of-order and in-order data. Data cannot be treated naively as an unordered relation, because some time-series constructs (windowing, streaming aggregation, and interpolation) are implemented in an order-dependent way, but it is also not solely ordered, because distribution and network delays often cause out-of-order arrivals. This raises the need to encode precisely what *ordering requirements* are made on physical events [8, 18–22]. Second, stream processing systems perform program transformations to achieve distribution: the (generally sequential or declarative) user query is parallelized and distributed across nodes, which requires making choices about how streams are partitioned and how operators are replicated. This raises the need to ensure *safe distribution*: the distributed code should be semantically equivalent to the original program in some sense [20, 21, 23]. Third, we observe that the performance of operators in a stream processing program is actually critical for correctness, and not just a matter of efficiency. This is because if a program receives more input items than it can handle, it will crash and the fault is likely unrecoverable. This raises the need to infer *performance guarantees* on operators to ensure predictable execution, ideally at compile time [24–29]. Finally, due to distributed deployment, stream processing applications should be fault-tolerant. This dimension is well-studied by existing work on ensuring fault tolerance for distributed streaming applications [30–34].

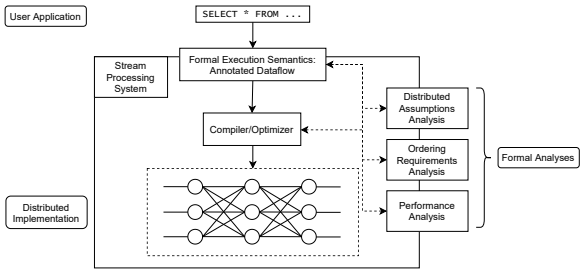

Figure 1: Envisioned Stream Processing Architecture: System exposes formal execution semantics and supports pluggable formal analyses.

Outlook. If successul, formalization could shape design and tool support for the future of stream processing systems. Figure 1 shows how formal models could fit in a unified architecture for stream processing applications. The system interface offers well-defined formal semantics and supports formal analyses (checking whether certain assumptions are met), which inform the compiler in generating an efficient and correct implementation. *Ordering requirements* are encoded in the formal execution semantics, and can be exploited by formal analyses and tools. *Safe distribution* semantics are exploited by the distributed implementation and compiler/optimizer. *Performance guarantees* are provided by a formal analysis of the user query, and preserved by the compiler/optimizer.

COPYRIGHT

REFERENCES

[1] Apache. Apache storm. http://storm.apache.org/, 2019. [Online; accessed March 31, 2019].

[2] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[3] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 423–438, New York, NY, USA, 2013. ACM.

[4] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38:28–38, 2015.

[5] Google Cloud. https://cloud.google.com/dataflow, 2021. [Online; accessed August 27, 2021].

[6] Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, Danyel Fisher, John C Platt, James F Terwilliger, and John Wernsing. Trill: A high-performance incremental query processor for diverse analytics. *Proceedings of the VLDB Endowment*, 8(4):401–412, 2014.

[7] Frank McSherry. Timely dataflow (rust). https://github.com/TimelyDataflow/timely-dataflow/, 2020. [Online; accessed September 30, 2020].

[8] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 439–455, New York, NY, USA, 2013. ACM.

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

[10] Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, and Miryung Kim. Bigdebug: Debugging primitives for interactive big data processing in spark. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 784–795. IEEE, 2016.

[11] Alexandre Vianna, Waldemar Ferreira, and Kiev Gama. An exploratory study of how specialists deal with testing in data stream processing applications. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019.

[12] Buğra Gedik, Henrique Andrade, Andy Frenkiel, Wim De Pauw, Michael Pfeifer, Paul Allen, Norman Cohen, and Kun-Lung Wu. Tools and strategies for debugging distributed stream processing applications. *Software: Practice and Experience*, 39(16):1347–1376, 2009.

[13] Apache. Apache flink. https://flink.apache.org/, 2019. [Online; accessed March 31, 2019].

[14] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in apache flink: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, August 2017.

[15] Apache flink 1.10 documentation: The broadcast state pattern. https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/state/broadcast_state.html.

[16] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. Samza: Stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, August 2017.

[17] Lorenzo Affetti, Alessandro Margara, and Gianpaolo Cugola. Flowdb: Integrating stream processing and consistent state management. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 134–145, 2017.

[18] Matthias Brun, Sára Decova, Andrea Lattuada, and Dmitriy Traytel. Verified progress tracking for timely dataflow. In *12th International Conference on Interactive Theorem Proving (ITP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[19] Rajeev Alur, Konstantinos Mamouras, Caleb Stanford, and Val Tannen. Interfaces for stream processing systems. In *Principles of Modeling*, pages 38–60. Springer, 2018.

[20] Konstantinos Mamouras, Caleb Stanford, Rajeev Alur, Zachary G. Ives, and Val Tannen. Data-trace types for distributed stream processing systems. pages 670–685, 2019.

[21] Rajeev Alur, Phillip Hilliard, Zachary G Ives, Konstantinos Kallas, Konstantinos Mamouras, Filip Niksic, Caleb Stanford, Val Tannen, and Anton Xue. Synchronization schemas. In *Invited contribution to Principles of Database Systems (PODS, invited contribution)*, pages 1–18, 2021.

[22] Konstantinos Kallas, Filip Niksic, Caleb Stanford, and Rajeev Alur. Diffstream: Differential output testing for stream processing programs. *Proceedings of the ACM on Programming Languages*, (OOPSLA), 2020.

[23] Scott Schneider, Martin Hirzel, Buğra Gedik, and Kun-Lung Wu. Safe data parallelism for general streaming. *IEEE transactions on computers*, 64(2):504–517, 2013.

[24] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[25] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[26] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[27] Konstantinos Mamouras, Mukund Raghothaman, Rajeev Alur, Zachary G Ives, and Sanjeev Khanna. Streamqre: Modular specification and efficient evaluation of quantitative queries over streaming data. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 693–708, 2017.

[28] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. Regular programming for quantitative properties of data streams. In *Proceedings of the 25th European Symposium on Programming (ESOP '16)*, pages 15–40, 2016.

[29] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Modular quantitative monitoring. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–31, 2019.

[30] Pedro F Silvestre, Marios Fragkoulis, Diomidis Spinellis, and Asterios Katsifodimos. Clonos: Consistent causal recovery for highly-available streaming dataflows. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1637–1650, 2021.

[31] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, pages 601–613, 2018.

[32] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Michael Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 13–24, 2005.

[33] Bonaventura Del Monte, Steffen Zeuch, Tilmann Rabl, and Volker Markl. Rhino: Efficient management of very large distributed state for stream processing engines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2471–2486, 2020.

[34] Mehul A Shah, Joseph M Hellerstein, and Eric Brewer. Highly available, fault-tolerant, parallel dataflows. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 827–838, 2004.