

Big Stream Processing Systems

Edited by

Tilman Rabl¹, Sherif Sakr², and Martin Hirzel³

1 TU Berlin, DE, rabl@tu-berlin.de

2 KSAU-HS, Riyadh, SA, sakrs@ksau-hs.edu.sa

3 IBM TJ Watson Research Center – Yorktown Heights, US, hirzel@us.ibm.com

Abstract

This report summarizes the Dagstuhl Seminar 17441 on “Big Stream Processing Systems” and documents its talks and discussions. The seminar brought together 29 researchers in various areas related to stream processing including systems, query languages, applications, semantic processing and benchmarking. The seminar program included four tutorials that have been delivered by experts in the various topics in addition to 29 lightening talks by the participants of the seminar. In this report, the abstracts of these talks are documented. Two working groups has been formed during the seminar. A report about the discussion outcomes of each group is presented in this report.

Seminar October 29–3, 2017 – <http://www.dagstuhl.de/17441>

1998 ACM Subject Classification C.2.4 Distributed Systems, C.2.4 Distributed Databases, H.2.4 Query Processing, D.3.2 Data-Flow Languages

Keywords and phrases Big Data, Big Streams, Stream Processing Systems, Benchmarking, Declarative Programming

Digital Object Identifier 10.4230/DagRep.7.10.111

1 Executive Summary

Martin Hirzel

Tilman Rabl

Sherif Sakr

License © Creative Commons BY 3.0 Unported license
© Martin Hirzel, Tilman Rabl, and Sherif Sakr

As the world gets more instrumented and connected, we are witnessing a flood of digital data that is getting generated, in a high velocity, from different hardware (e.g., sensors) or software in the format of streams of data. Examples of this phenomena are crucial for several applications and domains including financial markets, surveillance systems, manufacturing, smart cities and scalable monitoring infrastructure. In these applications and domains, there is a crucial requirement to collect, process, and analyze big streams of data in order to extract valuable information, discover new insights in real-time and to detect emerging patterns and outliers. Recently, several systems (e.g., Apache Apex, Apache Flink, Apache Storm, Heron, Spark Streaming,) have been introduced to tackle the real-time processing of big streaming data. However, there are several challenges and open problems that need to be addressed in order improve the state-of-the-art in this domain and push big stream processing systems to make them widely used by large number of users and enterprises. The aim of this seminar was to bring together active and prominent researchers, developers and practitioners actively working in the domain of big stream processing to discuss very relevant open challenges and



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Big Stream Processing Systems, *Dagstuhl Reports*, Vol. 7, Issue 10, pp. 111–138

Editors: Tilman Rabl, Sherif Sakr, and Martin Hirzel



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

research directions. The plan was to work on specific challenges including the trade-offs of the various design decisions of big stream processing systems, the declarative stream querying and processing languages, and the benchmarking challenges of big stream processing systems.

On Monday morning, the workshop officially kicked off with a round of introductions about the participants where adhoc clusters for the interests of the participants have been defined. The clusters have been revolving around the topics of systems, query languages, benchmarking, stream mining and semantic stream processing. The program of the seminar included 4 tutorials, one per day. On Monday, Martin Strohbach from AGT International presented different case studies and scenarios for large scale stream processing in different application domains. On Tuesday, we enjoyed the systems tutorial which has been presented by Paris Carbone from KTH Royal Institute of Technology, Thomas Weise from Data Torrent Inc. and Matthias J. Sax from Confluent Inc. Paris presented an interesting overview of the journey of stream processing systems, Thomas presented the recent updates about the Apache Apex system while Matthias presented an overview about the Apache Kafka and Kafka Streams projects. On Wednesday, Martin Hirzel from IBM TJ Watson Research Center presented a tutorial about the taxonomy and classifications of stream processing languages. On Thursday, Tilmann Rabl from TU Berlin presented a tutorial about the challenges of benchmarking big data systems in general in addition to the specific challenges for benchmarking big stream processing systems. All tutorials have been very informative, interactive and involved very deep technical discussions. On Thursday evening, we had a lively demo session where various participants demonstrated their systems to the audience on parallel round-table interactive discussions. On Wednesday, the participants split into two groups based on common interest in selected subset of the open challengers and problems. The selected 2 topics of the groups were systems and query languages. Thursday schedule was dedicated to working group efforts. Summary about the outcomes of these 2 groups is included in this report. It is expected that work from at least one of the groups to be submitted for publication, and we expect further research publications to result directly from the seminar.

We believe that the most interesting aspect of the seminar was providing the opportunity to freely engage in direct and interactive discussions with solid experts and researchers in various topics of the field with common focused passion and interest. We believe that this is a unique feature for Dagstuhl seminars. We received very positive feedback from the participants and we believe that most of the participants were excited with the scientific atmosphere at the seminar and reported that the program of the seminar was useful for them. In summary, we consider the organization of this seminar as a success. We are grateful for the Dagstuhl team for providing the opportunity and full support to organize it. The success of this seminar motivated us to plan for future follow-up seminars to continue the discussions on the rapid advancements on the domain and plan for narrower and more focused discussion with concrete outputs for the community.

2 Table of Contents

Executive Summary

<i>Martin Hirzel, Tilman Rabl, and Sherif Sakr</i>	111
--	-----

Overview of Talks

Approximate data analytics systems <i>Pramod Bhatotia</i>	115
Data Stream Mining <i>Albert Bifet</i>	115
Analysis of Queries on Big Graphs <i>Angela Bonifati</i>	116
Privacy Preserving, Peta-scale Stream Analytics for Domain-Experts <i>Michael H. Böhlen</i>	117
Interconnecting the Web of Data Streams <i>Jean-Paul Calbimonte</i>	117
Consistent Large-Scale Data Stream Processing <i>Paris Carbone</i>	118
Tutorial: Data Stream Processing Systems <i>Paris Carbone</i>	119
Cyber-Physical Social Systems for City-wide Infrastructures <i>Javier David Fernández-García</i>	120
Scaling SPADE to “Big Streams” <i>Ashish Gehani</i>	121
Benchmarking Enterprise Stream Processing Architectures <i>Günter Hesse</i>	123
Sliding-Window Aggregation in Worst-Case Constant Time <i>Martin Hirzel</i>	123
Tutorial: Stream Processing Languages <i>Martin Hirzel</i>	124
Benchmarking Semantic Stream Processing Platforms for IoT Applications <i>Ali Intizar</i>	124
Efficient evaluation of streaming queries comprising user-defined functions <i>Asterios Katsifodimos</i>	125
Towards an effort for Adaptable Stream Processing Engines <i>Nikos Katsipoulakis</i>	126
Druid as an Example of a Federated Streaming Data System <i>Henning Kropp</i>	126
Autonomous Semantic Stream Processing <i>Danh Le Phuoc</i>	127
Collaborative Distributed Processing Pipelines (CDPPs) <i>Manfred Hauswirth</i>	127

FlowDB: Integrating Stream Processing and Consistent State Management <i>Alessandro Margara</i>	128
Mining Big Data Streams – Better Algorithms or Faster Systems? <i>Gianmarco Morales</i>	128
Data Streams in IoT Applications: Data Quality and Data Integration <i>Christoph Quix</i>	129
Benchmarking Modern Streaming Systems <i>Tilmann Rabl</i>	130
Tutorial: Benchmarking <i>Tilmann Rabl</i>	130
Collaborative Benchmarking of Computer Systems <i>Sherif Sakr</i>	131
Low Latency Processing of Transactional Data Streams <i>Kai-Uwe Sattler</i>	131
Streams and Tables in Apache Kafka’s Streams API <i>Matthias J. Sax</i>	132
IoT Stream Processing Tutorial <i>Martin Strohbach, Alexander Wiesmaier, and Arno Mittelbach</i>	132
Quantifying and Detecting Incidents in IoT Big Data Analytics <i>Hong-Linh Truong</i>	133
Data Management of Big Spatio-temporal Data from Streaming and Archival Sources <i>Akrivi Vlachou</i>	133
Stream Processing with Apache Apex <i>Thomas Weise</i>	134
Lifetime-Based Memory Management in Distributed Data Processing Systems <i>Yongluan Zhou</i>	134
Working groups	
Working Group: Languages and Abstractions <i>Martin Hirzel</i>	135
Working Group: Systems and Applications <i>Tilmann Rabl</i>	136
Participants	138

3 Overview of Talks

3.1 Approximate data analytics systems

Pramod Bhatotia (University of Edinburgh, GB)

License © Creative Commons BY 3.0 Unported license
© Pramod Bhatotia

Joint work of Pramod Bhatotia, Le Quoc Do, Martin Beck, Dhanya Krishnan, Ruichuan Chen, Christof Fetzer, Thorsten Strufe, Volker Hilt, Istemi Akkus, Rodrigo Rodrigues, Spyros Blanas

We present approximate data analytics systems. Approximate computing aims for efficient execution of workflows where an approximate output is sufficient instead of the exact output. The idea behind approximate computing is to compute over a representative sample instead of the entire input dataset. Thus, approximate computing — based on the chosen sample size — can make a systematic trade-off between the output accuracy and computation efficiency.

In this talk, I presented data analytics system for approximate computing. Our work aims for an efficient mechanism based on approximation for large-scale data analytics. In particular, I presented four systems for approximate computing: (1) StreamApprox, a stream analytics systems for approximate computing, (2) PrivApprox, a privacy-preserving stream analytics system using approximate computing, (3) IncApprox, a data analytics system that combines incremental and approximate computing, and lastly, (4) ApproxJoin, a data analytics to support approximate distributed joins.

We have built our systems based on prominent distributed data analytics platforms, such as Apache Spark Streaming, and Apache Flink, which allow to transparently target a large class of existing applications. The source code of our approximate data analytics systems along with the full experimental evaluation setup is publicly available for the research community.

- Approx [1]: <https://privapprox.github.io/>
- StreamApprox [3]: <https://streamapprox.github.io/>
- IncApprox [2]: <https://gitlab.com/tudinfse/incapprox>

References

- 1 Do Le Quoc, Martin Beck, Pramod Bhatotia, Ruichuan Chen, Christof Fetzer, Thorsten Strufe. *PrivApprox: Privacy-Preserving Stream Analytics*. USENIX ATC, 2017.
- 2 Dhanya R. Krishnan, Do Le Quoc, Pramod Bhatotia, Christof Fetzer, Rodrigo Rodrigues. *IncApprox: A Data Analytics System for Incremental Approximate Computing*. WWW, 2016.
- 3 Do Le Quoc, Ruichuan Chen, Pramod Bhatotia, Christof Fetzer, Volker Hilt, Thorsten Strufe. *StreamApprox: approximate computing for stream analytics*. Middleware, 2017.

3.2 Data Stream Mining

Albert Bifet (Telecom ParisTech, FR)

License © Creative Commons BY 3.0 Unported license
© Albert Bifet

Big Data and the Internet of Things (IoT) have the potential to fundamentally shift the way we interact with our surroundings. The challenge of deriving insights from the Internet of Things (IoT) has been recognized as one of the most exciting and key opportunities for both academia and industry. Advanced analysis of big data streams from sensors and devices is

bound to become a key area of data mining research as the number of applications requiring such processing increases. Dealing with the evolution over time of such data streams, i.e., with concepts that drift or change completely, is one of the core issues in stream mining. I will present an overview of data stream mining, and I will introduce two popular open source tools for data stream mining.

3.3 Analysis of Queries on Big Graphs

Angela Bonifati (University Claude Bernard – Lyon, FR)

License © Creative Commons BY 3.0 Unported license

© Angela Bonifati

Joint work of Angela Bonifati, Nicky Advokaat, Guillaume Bagan, Radu Ciucanu, George Fletcher, Benoit Groz, Aurelien Lemay, Wim Martens, Thomas Timm

Main reference Angela Bonifati, Wim Martens, Thomas Timm: “An Analytical Study of Large SPARQL Query Logs”, *PVLDB* 11(2): 149–161, 2017.

URL <http://www.vldb.org/pvldb/vol11/p149-bonifati.pdf>

My research revolves around graph data management by focusing in particular on graph query processing, graph benchmarking and graph query log analysis. Several modern graph query languages are capable of expressing sophisticated graph queries, which return nodes connected by arbitrarily complex labeled paths. Such paths can be synthesized by means of regular expressions and often involve recursion. Such graph queries are known as Regular Path Queries (RPQ) and correspond to Property Paths in Sparql 1.1 to make an example of a concrete graph query language. Graph queries arbitrarily combine RPQ with conjunctions and unions to constitute a comprehensive query language for graph databases. Recently, with my colleagues I have been investigating graph queries and their different fragments by studying the synthetic generation problem of graph instances and graph query workloads [1, 2], along with the complexity of graph query evaluation [3] and the analysis of a large corpus of real-world graph queries [4]. In the latter work, we have examined streaks of queries that are queries that originate from gradual modifications of a seed query. If we switch from batch processing to online processing, these streaks can be considered as streams of queries collected from SPARQL endpoints. In summary, graph benchmarking and graph log analysis are essential to shape the future capabilities of graph query engines. Hence, they have a strong potential to influence the work of our community on graph query processing and optimization.

References

- 1 Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George Fletcher, Aurelien Lemay, and Nicky Advokaat. *Generating Flexible Workloads for Graph Databases*. *PVLDB*, 9(13):1447–1460, 2016.
- 2 Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George Fletcher, Aurelien Lemay, and Nicky Advokaat. *gMark: Schema-Driven Generation of Graphs and Queries*. *IEEE Trans. on Knowl. Data Eng.*, 29(4): 856–869, 2017
- 3 Guillaume Bagan, Angela Bonifati, and Benoit Groz. *A trichotomy for regular simple path queries on graphs*. In *Proceedings of PODS*, pages 261–272, 2013.
- 4 Angela Bonifati, Wim Martens, Thomas Timm: An Analytical Study of Large SPARQL Query Logs. *PVLDB* 11(2): 149–161 (2017)

3.4 Privacy Preserving, Peta-scale Stream Analytics for Domain-Experts

Michael H. Böhlen (Universität Zürich, CH)

License © Creative Commons BY 3.0 Unported license
© Michael H. Böhlen

Joint work of Michael H. Böhlen, Abraham Bernstein, Daniele dell'Aglio, Muhammad Saad, Pengcheng Duan

Production of big data will soon outpace the availability of both storage and computer science experts who know how to handle such data. Moreover, society is increasingly concerned about data protection. Addressing these issues requires so-called stream-processing systems that continuously analyse incoming data (rather than store it) and allow non-computer scientists to specify its analysis in a privacy-preserving manner. We will develop a petabyte-scale stream analytics system that enables non-computer scientists to analyse high-performance data streams on commodity hardware. First, we provide a declarative language based on traditional querying but with extensions for statistical operations and capabilities for real-time operations. Second, the language permits users to specify the desired level of privacy. Third, the system translates the statistical functions and privacy specifications into executable computations. Finally, the runtime environment selects the best approach for optimising execution using existing systems (e.g. Apache Flink, Spark Streaming or Storm). To evaluate the robustness and functionality of our system, we will replicate the processing pipeline for the Australian Square Kilometre Array Pathfinder radio telescope. This will generate up to 2.5 gigabytes per second of raw data. To evaluate privacy preservation, we will analyse the TV viewing habits of around 3 million individuals.

3.5 Interconnecting the Web of Data Streams

Jean-Paul Calbimonte (HES-SO Valais – Sierre, CH)

License © Creative Commons BY 3.0 Unported license
© Jean-Paul Calbimonte

Main reference Daniele Dell'Aglio, Danh Le Phuoc, Anh Lê Tuấn, Muhammad Intizar Ali, Jean-Paul Calbimonte: “On a Web of Data Streams”, in Proc. of the Workshop on Decentralizing the Semantic Web 2017 co-located with 16th Int'l Semantic Web Conf. (ISWC), 2017.

URL <http://ceur-ws.org/Vol-1934/contribution-11.pdf>

Main reference Jean-Paul Calbimonte: “Linked Data Notifications for RDF Streams”, in Joint Proc. of the Web Stream Processing workshop (WSP 2017) and the 2nd Int'l Workshop on Ontology Modularity, Contextuality, and Evolution (WOMoCoE 2017) co-located with 16th Int'l Semantic Web Conference (ISWC 2017), pp. 66–73, Vienna, Austria, October 22nd, 2017.

URL <http://ceur-ws.org/Vol-1936/paper-06.pdf>

The Web evolves toward a vast network of data, making it possible to publish, discover, access, process, and consume information through standard protocols. This Web of Data increasingly includes data streams, whose velocity and variety challenge current methods and techniques for processing/consumption/publishing. Although semantic data models such as RDF have shown to be successful for addressing these issues for stored data, it remains an open problem for data streams on the Web. We propose using standard protocols such as Linked Data Notifications (LDN) as the backbone for sending, receiving and consuming stream elements on the Web. This will allow a wider reuse of stream, going beyond existing silos and technical and administrative barriers. To achieve this, we envision the use of semantically rich metadata that describes these streams, regardless of their format and structure, providing the means to enhance findability, accessibility, linkability of these streams.

3.6 Consistent Large-Scale Data Stream Processing

Paris Carbone (*KTH Royal Institute of Technology, SE*)

License © Creative Commons BY 3.0 Unported license
© Paris Carbone

Joint work of Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, Kostas Tzoumas
Main reference Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, Kostas Tzoumas: “State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing”, PVLDB, Vol. 10(12), pp. 1718–1729, 2017.

URL <http://www.vldb.org/pvldb/vol10/p1718-carbone.pdf>

An early dogma on data stream processing technology labeled that tech as a fast, yet approximate, method for data analysis [5]. This argument has its roots on early design choices that considered limited in-memory data structures to maintain state, the lack of scale-out approaches seen in Map-Reduce and most importantly the conception that offering strong state consistency guarantees was non-trivial in the context of streams. Today, we are witnessing a ‘big stream processing’ revolution where stream processors such as Apache Flink, Kafka-Streams, Apex, Millwheel [4] (Beam/Dataflow Cloud [2] runner) etc. are being adopted as building blocks for scalable, continuous processing applications and pipelines. Modern data stream processors offer built-in state management with exactly-once end-to-end guarantees, eliminating the possibility of data loss or state inconsistencies as well as scaling-out in a data-parallel manner. The two dominant architectures to state management are 1) Externally persisted state in a transactional data store [4] and 2) Local state to compute nodes that is committed and replicated using consistent distributed snapshots flink [3, 1], ibmstreams, apex. Throughout this lightning talk and discussion we analyse the reasons behind locally maintained state, primarily in the context of Apache Flink [1] and its core snapshotting algorithm. We show that snapshots can be used for all operational needs of a long-running application such as live reconfiguration, fault tolerance, software patches and versioning. Finally, we address the costs of employing such an architecture both in terms of runtime overhead and reconfiguration time.

References

- 1 P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, K. Tzoumas, “State management in Apache Flink®: consistent stateful distributed stream processing” *Proceedings of the VLDB Endowment*, 2017.
- 2 Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., et al.: The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. VLDB (2015)
- 3 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*, p. 28, 2015.
- 4 Akidau T, Balikov A, Bekiroglu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) MillWheel: Fault-tolerant stream processing at internet scale. In: VLDB
- 5 The Lambda Architecture. <http://lambda-architecture.net/>

3.7 Tutorial: Data Stream Processing Systems

Paris Carbone (KTH Royal Institute of Technology, SE)

License  Creative Commons BY 3.0 Unported license
© Paris Carbone

Conventional data management considers data, in its traditional definition, as facts and statistics organised and collected together for future reference or analysis. A wide family of database management systems designed around the principle of having data as a static component, yet allowing complex and flexible retrospective analysis on that data such as declarative adhoc sql queries. Large-scale data processing architectures aimed to scale out those technologies, examples are Map-Reduce and the Apache Spark stack. Yet, regardless of the undeniable scale-out we effectively got the same old perception of data processing in a “new outfit”.

Data stream processing revolutionizes the way we define data in the first place, lifting its context from retrospective data set analysis to continuous unbounded processing coupled with persistent application state. Parts of that technology have been available in different primary forms and domains, such as network-centric processing on byte streams, functional programming (e.g., monads), actor programming and materialized views. However, we now see how all of these ideas have been put together to form a system architecture that is built and therefore keeps evolving around the notion of data as an unbounded collective stream of facts.

This new “Scalable Stream Processing” architecture has its own stack. Starting from the storage layer, dedicated partitioned logs such as Apache Kafka and Pravega replace distributed file systems (e.g., HDFS). Partitioned logs have unique characteristics that build on the notion of streams such as event stream producers, consumers, delivery guarantees as well as stream reconfiguration capabilities. At the middle, we have data compute systems such as Flink, Kafka-Streams, Apex, Timely-Dataflow and Spark-Streaming that offer continuous stateful stream processing with, most often, end-to-end exactly-once processing guarantees. Furthermore, the semantics of all these systems have been lifted from primitive per-event processing to declarative high-level abstractions such as stream windowing (triggers, event-time domain integration, sessions etc.). Finally, at the top-most layer we can see new forms of libraries and user-facing programming models covering relational event streams (stream sql), complex event processing (cep) as well as newly formed domain-specific languages and models for streams.

The ultimate aim of the tutorial is to offer a top-down view of all these concepts and pose potential insights of the upcoming needs of stream processing technology. On that end, we discuss the prospects and benefits of standardization, both in terms of runtime characteristics (e.g., snapshots), core programming models (e.g., Beam) and finally higher-level APIs (Stream SQL, Calcite, complex event processing APIs).

3.8 Cyber-Physical Social Systems for City-wide Infrastructures

Javier David Fernández-García (Wirtschaftsuniversität Wien, AT)

License © Creative Commons BY 3.0 Unported license
© Javier David Fernández-García

Main reference Aljbin Ahmeti, Saimir Bala, Fajar J. Ekaputra, Javier D. Fernández, Elmar Kiesling, Andreas Koller, Jan Mendling, Angelika Musil, Axel Polleres, Peb R. Aryan, Marta Sabou, Andreas Solti, Juergen Musil: “CitySPIN: Cyber-Physical Social Systems for City-wide Infrastructures”, in Proc. of the 13th Int’l Conf. on Semantic Systems, Posters and Demos track, 2017.

URL <http://ceur-ws.org/Vol-2044/paper21>

The potential of Big Semantic Data is under-exploited when data management is based on traditional, human-readable RDF representations, which add unnecessary overheads when storing, exchanging and consuming RDF in the context of a large-scale and machine-understandable Semantic Web. In the first part of the talk, we briefly present our HDT [1] proposal, a compact data structure and binary serialization format that keeps big RDF datasets compressed while maintaining search and browse operations without prior decompression. As a practical use case, we show the recent project LOD-a-lot [4], which uses HDT to represent and query more than 28 billion triples of the current Linked Open Data network. Then, we inspect the challenges of using a write-once-read-multiple HDT format in a streaming scenario, providing details on two real-time solutions: i) SOLID [2], a lambda-based compact triplestore to manage evolving Big Semantic Data, and ii) ERI [3], a compact serialization format for RDF streams. Finally, we present CitySPIN [5], a project using such Big Semantic Data technologies to integrate and manage cyber-physical social systems in order to facilitate innovative Smart City infrastructure services.

References

- 1 J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange. *Journal of Web Semantics*, 19:22–41, 2013.
- 2 M. A. Martínez-Prieto, C. E. Cuesta, M. Arias, and J. D. Fernández. The solid architecture for real-time management of big semantic data. *Future Generation Computer Systems*, 47, 62–79, 2015.
- 3 J. D. Fernández, A. Llaves, and O. Corcho. Efficient RDF interchange (ERI) format for RDF data streams. In *Proceedings of the International Semantic Web Conference*, pp. 244–259, 2014.
- 4 J. D. Fernández, W. Beek, M. A. Martínez-Prieto, and M. Arias. LOD-a-lot: A Queryable Dump of the LOD cloud. In *Proceedings of the International Semantic Web Conference*, pp. 75–83, 2017.
- 5 A. Ahmeti, S. Bala, F. J. Ekaputra, J. D. Fernández, E. Kiesling, A. Koller, J. Mendling, A. Musil, A. Polleres, P. R. Aryan, M. Sabou, A. Solti, and J. Musil. CitySPIN: Cyber-Physical Social Systems for City-wide Infrastructures. In *13th International Conference on Semantic Systems, Posters and Demos track*, 2017.

3.9 Scaling SPADE to “Big Streams”

Ashish Gehani (SRI – Menlo Park, US)

License © Creative Commons BY 3.0 Unported license
© Ashish Gehani

Joint work of Ashish Gehani, Hasanat Kazmi, Hassaan Irshad

Main reference Ashish Gehani, Hasanat Kazmi, Hassaan Irshad: “Scaling SPADE to “Big Provenance””, in Proc. of the 8th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2016, Washington, D.C., USA, June 8-9, 2016., USENIX Association, 2016.

URL <https://www.usenix.org/conference/tapp16/workshop-program/presentation/gehani>

Knowledge of the provenance of data has many uses, but also imposes novel challenges. Video provenance facilitates fine-grained attribution [1]. Operating system provenance enables principled forensic analysis [3], identification of the source of Grid infections [6], and authenticity claims that span trust domains [5]. Since provenance metadata can become voluminous, organizing it [4] and securing it [2] can be challenging. This has motivated policy-based [7] and flexible middleware-supported [8] approaches.

SPADE [13] is SRI’s open source system for managing provenance metadata. It has served as the research framework for exploring a range of ideas. These include automating the capture of application-level provenance through compiler instrumentation [15], accelerating distributed provenance queries via the use of sketches [12], optimizing the re-execution of workflows [11], diagnosing problems in mobile applications [10], vetting application for sensitive data flows [16], and studying tradeoffs in byte-, function-, and system-call level provenance tracking [14].

Most systems that track data provenance in distributed environments opt to collect it centrally giving rise to “*big streams*”. SPADE stores provenance at the host that it originated from, thereby decomposing a single big stream into multiple, concurrent ones. It avoids reconstruction of the entire global stream by introducing *coordination points* between independent provenance streams. In SPADE’s terminology, these are called *network artifacts* since they can be computed independently on each host using attributes of incoming and outgoing network flows. At query time, SPADE extracts relevant subsets from each host’s provenance stream. Using the network artifacts, these are stitched together into a single response that corresponds to the appropriate subset of the global provenance stream.

Since each host gives rise to a big stream of provenance, scaling continues to pose a challenge. Three strategies are adopted to ameliorate the issues that arise [9]. (1) Though SPADE allows a provenance stream to be abstracted online through the use of *filters*, the approach cannot be employed when details must be retained – in the case of forensic analysis, for example. Instead responses to big stream queries are rewritten with *transformers* to provide more comprehensible answers. (2) Merging auxiliary information into a big stream is problematic when large integration windows are needed. If the stream schema allow, content-based integration can be adopted to address this. (3) Deduplication of elements in a stream requires memory linear in the history’s size. For big streams, this cost becomes prohibitive. Leveraging persistent storage can address this at the cost of performance. A hybrid approach that combines caching and Bloom filters is developed to screen out duplicates with high probability.

References

- 1 Ashish Gehani and Ulf Lindqvist, **VEIL: A System for Certifying Video Provenance**, *9th IEEE International Symposium on Multimedia (ISM)*, 2007.

- 2 Ashish Gehani and Ulf Lindqvist, **Bonsai: Balanced Lineage Authentication**, *23rd Annual Computer Security Applications Conference (ACSAC)*, IEEE Computer Society, 2007.
- 3 Ashish Gehani, Florent Kirchner, and Natarajan Shankar, **System Support for Forensic Inference**, *5th IFIP International Conference on Digital Forensics*, 2009.
- 4 Ashish Gehani, Minyoung Kim, and Jian Zhang, **Steps Toward Managing Lineage Metadata in Grid Clusters**, *1st Workshop on the Theory and Practice of Provenance (TaPP)* affiliated with the *7th USENIX Conference on File and Storage Technologies (FAST)*, 2009.
- 5 Ashish Gehani and Minyoung Kim, **Mendel: Efficiently Verifying the Lineage of Data Modified in Multiple Trust Domains**, *19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.
- 6 Ashish Gehani, Basim Baig, Salman Mahmood, Dawood Tariq, and Fareed Zaffar, **Fine-Grained Tracking of Grid Infections**, *11th ACM/IEEE International Conference on Grid Computing (GRID)*, 2010.
- 7 Ashish Gehani, Dawood Tariq, Basim Baig, and Tanu Malik, **Policy-Based Integration of Provenance Metadata**, *12th IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, 2011.
- 8 Ashish Gehani and Dawood Tariq, **SPADE: Support for Provenance Auditing in Distributed Environments**, *13th ACM/IFIP/USENIX International Conference on Middleware*, 2012.
- 9 Ashish Gehani, Hasanat Kazmi, and Hassaan Irshad, **Scaling SPADE to “Big Provenance”**, *8th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2016.
- 10 Nathaniel Husted, Sharjeel Qureshi, Dawood Tariq, and Ashish Gehani, **Android Provenance: Diagnosing Device Disorders**, *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP)* affiliated with the *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
- 11 Hasnain Lakhani, Rashid Tahir, Azeem Aqil, Fareed Zaffar, Dawood Tariq, and Ashish Gehani, **Optimized Rollback and Re-computation**, *46th IEEE Hawaii International Conference on Systems Science (HICSS)*, IEEE Computer Society, 2013.
- 12 Tanu Malik, Ashish Gehani, Dawood Tariq, and Fareed Zaffar, **Sketching Distributed Data Provenance**, *Data Provenance and Data Management for eScience, Studies in Computational Intelligence*, Vol. 426, Springer, 2013.
- 13 Support for Provenance Auditing in Distributed Environments, <http://spade.csl.sri.com>
- 14 Manolis Stamatogiannakis, Hasanat Kazmi, Hashim Sharif, Remco Vermeulen, Ashish Gehani, Herbert Bos, and Paul Groth, **Tradeoffs in Automatic Provenance Capture**, *Provenance and Annotation of Data and Processes, Lecture Notes in Computer Science*, Vol. 9672, Springer, 2016.
- 15 Dawood Tariq, Maisem Ali, and Ashish Gehani, **Towards Automated Collection of Application-Level Data Provenance**, *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2012.
- 16 Chao Yang, Guangliang Yang, Ashish Gehani, Vinod Yegneswaran, Dawood Tariq, and Guofei Gu, **Using Provenance Patterns to Vet Sensitive Behaviors in Android Apps**, *11th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2015.

3.10 Benchmarking Enterprise Stream Processing Architectures

Günter Hesse (Hasso-Plattner-Institut – Potsdam, DE)

License © Creative Commons BY 3.0 Unported license
© Günter Hesse

Main reference Günter Hesse, Christoph Matthies, Benjamin Reissaus, Matthias Uflacker: “A New Application Benchmark for Data Stream Processing Architectures in an Enterprise Context: Doctoral Symposium”, in Proc. of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017, pp. 359–362, ACM, 2017.

URL <http://dx.doi.org/10.1145/3093742.3093902>

Data stream processing systems have become increasingly popular tools for analyzing large amounts of data that are generated in short periods of time. This is the case, for example in Industry 4.0 and Internet of Things scenarios, where masses of sensor and log data are continuously produced. This information can be leveraged in order to develop advanced applications, e.g., in the area of predictive maintenance.

Analysis of such data streams can become even more valuable when it takes advantage of traditional enterprise data, i.e., data from business systems such as an ERP system. Combining this transactional data with streaming data can unveil new insights with respect to processes and causal relations.

In recent years, many new data stream processing systems have been developed. Although a broad variety of systems allows for more choice, choosing the system that best suits a given use case becomes more difficult. Benchmarks are a common way of tackling this issue and allow a comprehensive comparison of different systems and setups. Currently, no suitable benchmark is available for comparing data stream processing architectures, especially when non-streaming enterprise data needs to be integrated.

With Senska, we will develop a new application benchmark for data stream processing architectures in an enterprise context that fills this gap.

3.11 Sliding-Window Aggregation in Worst-Case Constant Time

Martin Hirzel (IBM TJ Watson Research Center – Yorktown Heights, US)

License © Creative Commons BY 3.0 Unported license
© Martin Hirzel

Joint work of Kanat Tangwongsan, Martin Hirzel, Scott Schneider


Main reference Kanat Tangwongsan, Martin Hirzel, Scott Schneider: “Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time”, in Proc. of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017, pp. 66–77, ACM, 2017.

URL <http://dx.doi.org/10.1145/3093742.3093925>

This talk briefly summarizes a paper with the same title that appeared at DEBS 2017 (International Conference on Distributed and Event-based Systems), where it won a best-paper award. Sliding-window aggregation is a widely-used approach for extracting insights from the most recent portion of a data stream. The aggregations of interest can usually be cast as binary operators that are associative, but they are not necessarily commutative nor invertible. Non-invertible operators, however, are difficult to support efficiently. The best published algorithms require $O(\log n)$ aggregation steps per window operation, where n is the sliding-window size at that point. This paper presents DABA, a novel algorithm for aggregating FIFO sliding windows using only $O(1)$ aggregation steps per operation in the worst case (not just on average).

3.12 Tutorial: Stream Processing Languages


Martin Hirzel (IBM TJ Watson Research Center – Yorktown Heights, US)

License  Creative Commons BY 3.0 Unported license
© Martin Hirzel

This tutorial gives an overview of several styles of stream processing languages. The tutorial illustrates each style (relational, synchronous, explicit graph, etc.) with a representative example language. Of course, for each style, there is an entire family of languages, and this tutorial does not aim to be exhaustive. Overall, the field is diverse. Efforts to consolidate and standardize should be informed by an overview of the state of the art, which this tutorial provides.

3.13 Benchmarking Semantic Stream Processing Platforms for IoT Applications

Ali Intizar (National University of Ireland – Galway, IE)

License  Creative Commons BY 3.0 Unported license
© Ali Intizar

With the growing popularity of Internet of Things (IoT) technologies and sensors deployment, more and more IoT-enabled applications are designed that can leverage the rich source of streaming data to gather knowledge, support data analytics and provide useful applications for end users such as smart city applications. Semantic Web and its underlying technologies are an ideal fit to support distributed heterogeneous applications designed over deployed sensors based infrastructure within smart cities. A merger of IoT and semantic Web has lead to the inception of RDF stream processing (RSP) and several RSP based streaming engines have been proposed. However, RSP technologies are still in their infancy and yet to be tested for their performance and scalability. Particularly for smart city applications IoT-enabled semantic solutions should be tested and benchmarked with the real deployments and infrastructure accessible within smart cities.

The Citybench Benchmark is a comprehensive benchmarking suite to evaluate RSP engines within smart city applications using real streaming data generated by smart cities. CityBench includes real-time IoT data streams generated from various sensors deployed within the city of Aarhus, Denmark. We provide a configurable testing infrastructure and a set of continuous queries covering a variety of data and application dependent characteristics and performance metrics, to be executed over RSP engines using CityBench datasets. This work can be used as a baseline to identify capabilities and limitations of existing RSP engines for smart city applications and provide support to smart city application developers to benchmark their applications.

3.14 Efficient evaluation of streaming queries comprising user-defined functions

Asterios Katsifodimos (TU Delft, NL)

License © Creative Commons BY 3.0 Unported license
© Asterios Katsifodimos

Joint work of Asterios Katsifodimos, Paris Carbone, Jonas Traub, Volker Markl, Tilman Rabl, Seif Haridi, Sebastian Bress

Main reference Paris Carbone, Jonas Traub, Asterios Katsifodimos, Seif Haridi, Volker Markl: “Cutty: Aggregate Sharing for User-Defined Windows”, in Proc. of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016, pp. 1201–1210, ACM, 2016.

URL <http://dx.doi.org/10.1145/2983323.2983807>

Aggregation queries on data streams are evaluated over evolving and often overlapping logical views called windows. While the aggregation of periodic windows were extensively studied in the past through the use of aggregate sharing techniques such as Panes and Pairs, little to no work has been put in optimizing the aggregation of very common, non-periodic windows. Typical examples of non-periodic windows are punctuations and sessions which can implement complex business logic and are often expressed as user-defined operators on platforms such as Google Dataflow or Apache Storm. The aggregation of such non-periodic or user-defined windows either falls back to expensive, best-effort aggregate sharing methods, or is not optimized at all. In my talk I presented a technique to perform efficient aggregate sharing for data stream windows, which are declared as user-defined functions (UDFs) and can contain arbitrary business logic. I introduced the concept of User-Defined Windows (UDWs), a simple, UDF-based programming abstraction that allows users to programmatically define custom windows. I then defined semantics for UDWs, based on which we designed Cutty [1], a low-cost aggregate sharing technique. I believe that user-defined windows is a noteworthy programming abstraction to be included in future stream programming languages.


On the other side of the spectrum, real-time sensor data enables diverse applications such as smart metering, traffic monitoring, and sport analysis. In the Internet of Things, billions of sensor nodes form a sensor cloud and offer data streams to analysis systems. However, it is impossible to transfer all available data with maximal frequencies to all applications. Therefore, we need to tailor data streams to the demand of applications. In recent work we contributed a technique that optimizes communication costs while maintaining the desired accuracy. Our technique [2] schedules reads across huge amounts of sensors based on the data-demands of a huge amount of concurrent queries. In the same spirit as Cutty, we introduce user-defined sampling functions that define the data-demand of queries and facilitate various adaptive sampling techniques, which decrease the amount of transferred data.

References

- 1 Paris Carbone, Jonas Traub, Asterios Katsifodimos, Seif Haridi, and Volker Markl. *Cutty: Aggregate sharing for user-defined windows*. In the Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 1201–1210. ACM, 2016.
- 2 Jonas Traub, Sebastian Breß, Tilman Rabl, Asterios Katsifodimos, and Volker Markl. *Optimized on-demand data streaming from sensor nodes*. In the Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17). ACM, New York, NY, USA, 586–597.

3.15 Towards an effort for Adaptable Stream Processing Engines

Nikos Katsipoulakis (University of Pittsburgh, US)

License  Creative Commons BY 3.0 Unported license
© Nikos Katsipoulakis

Joint work of Nikos Katsipoulakis, Alexandros Labrinidis, Panos K. Chrysanthis

Online data flow processing requires that a Parallel Stream Processing Engine (pSPE) processes data by the time they become available and queries execute for a long period of time. Low operational cost and production of results in a timely fashion can be challenging goals for a pSPE, considering the volatile nature of streams. When the arrival rate of input data spikes, pSPEs need to be able to adjust their internal components, so that late delivery of results is avoided. This calls for adaptable pSPEs that can react to fluctuations in processing demands. This work aims to improve pSPEs' adaptability by revisiting internal components' design and load-balancing techniques. Adaptability is studied from the aspect of partitioning (Distribute), repartitioning (Divide), and load shedding (Drop). For each technique, the current state of the art is examined, both analytically and experimentally, and its shortcomings are exposed. In addition, new algorithms are proposed and this thesis' goal is (i) the investigation of decision-making algorithms, for selecting the best technique, (ii) the exploration of novel synergies among partitioning, repartitioning and load shedding.

References

- 1 N. R. Katsipoulakis, A. Labrinidis, and P. Chrysanthis, *A holistic view of stream partitioning costs*. PVLDB, pp. 1286–1297, 2017.
- 2 N. R. Katsipoulakis, C. Thoma, E. A. Gratta, et al., *Ce-storm: Confidential elastic processing of data streams*. ACM SIGMOD, pp. 859–864, 2015.
- 3 T. N. Pham, N. R. Katsipoulakis, P. K. Chrysanthis, and A. Labrinidis, *Uninterruptible migration of continuous queries without operator state migration*. SIGMOD Rec., vol. 46, no. 3, pp. 17–22, 2017.

3.16 Druid as an Example of a Federated Streaming Data System

Henning Kropp (Hortonworks – München, DE)

License  Creative Commons BY 3.0 Unported license
© Henning Kropp

Streaming systems are often looked at as long running queries executed on an unbounded stream of data. In such a scenario a Federated Streaming Data System (FSDS) would execute such queries on heterogeneous and autonomous streaming systems.

In modern large enterprises heterogeneous streaming systems inevitable arise from different requirements, diverse system landscapes, evolving technology, and geographic distribution. FSDDS enable such companies to fully leverage the full potential of their streamed data. The success of unifying programming models like Apache Beam and Apache Calcite are a testimony for the potential value federated streaming data systems hold.

Further Druid (<http://druid.io/>) could be seen as an example of a federated streaming system. In it's architecture Duid distinguishes between a group of realtime and a group of historic nodes which use different kind of data access patterns to collect data from streams and answer queries. Combining the two types of node types Druid uses a catalog and query federation service, common to federated database systems, to transparently serve data from two different kind of systems. Although Druid is more like a streaming database then a streaming data system, it's architectures gives an idea of how a FSDDS could look like.

3.17 Autonomous Semantic Stream Processing

Danh Le Phuoc (TU Berlin, DE)

License © Creative Commons BY 3.0 Unported license
© Danh Le Phuoc

Semantic Data Stream enables embedding computational semantics of contextual information and user/developer intentions into data stream generated from sensory data to structured (dynamic) knowledge base. By its intrinsic nature, semantic data stream sources are dynamically distributed in terms of spatial context, connectivity and distribution of the processing flow. To deal with this dynamicity, we propose the autonomous semantic stream processing model which sees semantic data streams as temporal RDF graphs (RDF streams). Via this model, distributed stream processing agents can autonomously coordinate their execution processes with their neighbour peers via exchanging control RDF streams among them. The control RDF streams enable a local RDF-based stream processing engine to continuously feed processing statuses from down stream processing agents such as data distribution, rate, available resources, processing capabilities and connectivity (network connections and link statuses, etc) to adaptively optimise its local processing pipelines. The adaptive optimisation of an processing agent can be done continuously in a collaboratively fashion with its neighbour peers which allow its to shift processing load and processing states to these peers. To realise the processing model, we built an autonomous processing kernel, called CQELS (Continuous Query Evaluation over Linked Stream) [1], which can run on small devices which collect sensor data as stream sources as well as on gateways with more processing power that can coordinate a small group of processing nodes. For dealing with big workload, the kernel can be also run as a cluster of powerful processing nodes on the Cloud. With our CQELS kernel, a distributed stream processing workflow is autonomously coordinated via its instances deployed in highly dynamic network topologies of heterogeneous processing nodes.

References

- 1 Phuoc, D.L., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In: ISWC. pp. 370–388 (2011)

3.18 Collaborative Distributed Processing Pipelines (CDPPs)

Manfred Hauswirth

License © Creative Commons BY 3.0 Unported license
© Manfred Hauswirth

Joint work of Anja Feldmann, Manfred Hauswirth, Volker Markl

Main reference Anja Feldmann, Manfred Hauswirth, Volker Markl: “Enabling Wide Area Data Analytics with Collaborative Distributed Processing Pipelines (CDPPs)”, in Proc. of the 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017, pp. 1915–1918, IEEE Computer Society, 2017.

URL <http://dx.doi.org/10.1109/ICDCS.2017.332>

Novel concepts to organize the distribution and processing of information over the Internet in a secure and safe way which is scalable are needed. An important building block for achieving this goal are mobile edge clouds. A mobile edge cloud provides micro data centers which can move in the network according to load or other parameters and requirements. This is another form of virtualization which is transforming infrastructure everywhere, including network components, end-user devices, and eventually even sensors to transparently use any kind

of resource. To work efficiently and scalably, mobile edge clouds need to be complemented by an appropriate world-wide network as well as sufficient backend computing power and storage. Combining all of this, we will see a seamless integration of network, storage, and computing in the future.

3.19 FlowDB: Integrating Stream Processing and Consistent State Management

Alessandro Margara (Polytechnic University of Milan, IT)

License © Creative Commons BY 3.0 Unported license

© Alessandro Margara

Joint work of Lorenzo Affetti, Alessandro Margara, Gianpaolo Cugola

Main reference Lorenzo Affetti, Alessandro Margara, Gianpaolo Cugola: “FlowDB: Integrating Stream Processing and Consistent State Management”, in Proc. of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017, pp. 134–145, ACM, 2017.

URL <http://dx.doi.org/10.1145/3093742.3093929>

Recent advances in stream processing technologies led to their adoption in many large companies, where they are becoming a core element in the data processing stack. In these settings, stream processors are often used in combination with various kinds of data management frameworks to build software architectures that combine data storage, processing, retrieval, and mining. However, the adoption of separate and heterogeneous subsystems makes these architectures overmuch complex, and this hinders the design, development, maintenance, and evolution of the overall system. In this talk, we propose a new model that integrates data management within a distributed stream processor. The model enables individual stream processing operators to persist data and make it visible and queryable from external components. It offers flexible mechanisms to control the consistency of data, including transactional updates plus ordering and integrity constraints. We implemented the model into the FlowDB prototype, and studied its overhead with respect to a pure stream processing system using real world case studies and synthetic workloads, proving the benefits of the proposed model by showing that FlowDB can outperform a state-of-the-art, in-memory distributed database in data management tasks.

3.20 Mining Big Data Streams – Better Algorithms or Faster Systems?

Gianmarco Morales (QCRI – Doha, QA)

License © Creative Commons BY 3.0 Unported license

© Gianmarco Morales

Main reference Gianmarco De Francisci Morales, Albert Bifet: “SAMOA: scalable advanced massive online analysis”, *Journal of Machine Learning Research*, Vol. 16, pp. 149–153, 2015.

URL <http://dl.acm.org/citation.cfm?id=2789277>

The rate at which the world produces data is growing steadily, thus creating ever larger streams of continuously evolving data. However, current (de-facto standard) solutions for big data analysis are not designed to mine evolving streams. So, should we find better algorithms to mine data streams, or should we focus on building faster systems?


In this talk, we debunk this false dichotomy between algorithms and systems, and we argue that the data mining and distributed systems community need to work together to bring about the next revolution in data analysis. In doing so, we introduce Apache SAMOA

(Scalable Advanced Massive Online Analysis), an open-source platform for mining big data streams (<http://samoa.incubator.apache.org>). Apache SAMOA provides a collection of distributed streaming algorithms for data mining tasks such as classification, regression, and clustering. It features a pluggable architecture that allows it to run on several distributed stream processing engines such as Storm, Flink, and Samza.

As a case study, we present one of SAMOA's main algorithms for classification, the Vertical Hoeffding Tree (VHT). Then, we analyze the algorithm from a distributed systems perspective, highlight the issue of load balancing, and describe a generalizable solution to it. Finally, we conclude by envisioning system-algorithm co-design as a promising direction for the future of big data analytics.

3.21 Data Streams in IoT Applications: Data Quality and Data Integration

Christoph Quix (Fraunhofer FIT – Sankt Augustin, DE)

License  Creative Commons BY 3.0 Unported license
© Christoph Quix

Data integration is an open challenge that has been addressed in static data management for decades [1]. In data streams, the need of efficient data integration techniques is even more significant as the data has to be integrated while the stream is being processed. This means that the data can be processed only once and there should be not much overhead caused by the integration operations. Data quality issues also often arise in the context of data integration, as inconsistencies and incorrect values are revealed when several data sources are combined.

In data stream processing for Internet-of-Things (IoT) applications, the challenges of data integration and data quality are also very important as a network of interoperable devices and systems can only be established, if there are appropriate tools and techniques to integrate data and to verify the quality of the data. For example, in traffic applications, various data sources have to be combined for traffic state estimation or queue-end detection [3]. However, the techniques applied in this context are approximate techniques, such as data stream mining or map matching; thus, a result should always include a confidence value that indicates the quality. In industrial applications, the semantic heterogeneity of data, e.g., sensor data or data from ERP systems, requires a sophisticated semantic modeling of the data. In the context of Industry 4.0 applications, it is becoming less important in which factory a particular step of a manufacturing process is executed as the information of the production process needs to be efficiently exchanged along the value chain [2].

In this talk, we present our recent results on data integration and data quality management in data lakes [4, 5], and data streams [3].

References

- 1 Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, Magdalena Balazinska, Philip A. Bernstein, Michael J. Carey, Surajit Chaudhuri, Jeffrey Dean, AnHai Doan, Michael J. Franklin, Johannes Gehrke, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, H. V. Jagadish, Donald Kossmann, Samuel Madden, Sharad Mehrotra, Tova Milo, Jeffrey F. Naughton, Raghu Ramakrishnan, Volker Markl, Christopher Olston, Beng Chin Ooi, Christopher Ré, Dan Suci, Michael Stonebraker, Todd Walter, and Jennifer Widom. The beckman report on database research. *Commun. ACM*, 59(2):92–99, 2016.

- 2 Malte Brettel, Niklas Friederichsen, Michael Keller, and Marius Rosenberg. How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(1):37–44, 2014.
- 3 Sandra Geisler, Christoph Quix, Stefan Schiffer, and Matthias Jarke. An evaluation framework for traffic information systems based on data streams. *Transportation Research Part C*, 23:29–55, August 2012.
- 4 Rihan Hai, Sandra Geisler, and Christoph Quix. Constance: An intelligent data lake system. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proc. Intl. Conf. on Management of Data (SIGMOD)*, pages 2097–2100, San Francisco, CA, USA, 2016. ACM.
- 5 Matthias Jarke and Christoph Quix. On warehouses, lakes, and spaces: The changing role of conceptual modeling for data integration. In Jordi Cabot, Cristina Gómez, Oscar Pastor, Maria-Ribera Sancho, and Ernest Teniente, editors, *Conceptual Modeling Perspectives.*, pages 231–245. Springer, 2017.

3.22 Benchmarking Modern Streaming Systems

Tilman Rabl (TU Berlin, DE)

License  Creative Commons BY 3.0 Unported license
© Tilman Rabl

Due to the recent trends of increasingly fast analysis of data, an increasing number of stream processing systems is built. Many of these include advanced features, such as a distributed architecture, different notions of time, and fault tolerance, with varying performance characteristics. These characteristics as well as basic stream processing operations pose specific challenges in benchmarking. In this talk, we identify several of these challenges, most notably the open world setup. In contrast to the closed world setup, where the streaming system under test has full control of the rate of incoming data, in an open world setup, the system has no influence on the data rate. While this is the typical setup of real deployments, there is no benchmark that properly tests this configuration. Our experiments demonstrate that current benchmarks fail to report correct latency measurements and overestimate throughput measures for real world setups.

3.23 Tutorial: Benchmarking

Tilman Rabl (TU Berlin, DE)

License  Creative Commons BY 3.0 Unported license
© Tilman Rabl

In this tutorial, we cover why, when, and how to benchmark. Before introducing different types of benchmarks and standardized instantiations of those, we give a brief overview of performance estimation. After this, we give an overview of existing stream benchmarks and specific challenges when benchmarking streaming systems. Then, we exemplify the development process of modern industry standard benchmarks through TPCx-HS.

3.24 Collaborative Benchmarking of Computer Systems

Sherif Sakr (KSAU-HS – Riyadh, SA)

License © Creative Commons BY 3.0 Unported license
© Sherif Sakr

Joint work of Sherif Sakr, Amin Shafaat, Fuad Bajaber, Ahmed Barnawi, Omar Batarfi, Abdulrahman Altalhi

Main reference Sherif Sakr, Amin Shafaat, Fuad Bajaber, Ahmed Barnawi, Omar Batarfi, Abdulrahman H. Altalhi: “Liquid Benchmarking: A Platform for Democratizing the Performance Evaluation Process”, in Proc. of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015., pp. 537–540, OpenProceedings.org, 2015.

URL <http://dx.doi.org/10.5441/002/edbt.2015.52>

Performances evaluation, reproducibility and benchmarking represent crucial aspects for assessing the practical impact of research results in the computer science

field. In spite of all the benefits (e.g., increasing impact, increasing visibility, improving the research quality) that can be gained from performing extensive experimental evaluation or providing reproducible software artifacts and detailed description of experimental setup, the required effort for achieving these goals remains prohibitive. In practice, conducting an independent, consistent and comprehensive performance evaluation and benchmarking is a very time and resource consuming process. As a result, the quality of published experimental results is usually limited and constrained by several factors such as: limited human power, limited time, or shortage of computing resources.

Liquid Benchmarking has been designed as an online and cloud-based platform for democratizing the performance evaluation and benchmarking processes. In particular, the platform facilitates the process of sharing the experimental artifacts (software implementations, datasets, computing resources, benchmarking tasks) as services where the end user can easily create, mashup, run the experiments and visualize the experimental results with zero installation or configuration efforts. In addition, the collaborative features of the platform enables the user to share and comment on the results of the conducted experiments so that it can guarantee a transparent scientific crediting process.

3.25 Low Latency Processing of Transactional Data Streams

Kai-Uwe Sattler (TU Ilmenau, DE)

License © Creative Commons BY 3.0 Unported license
© Kai-Uwe Sattler

Transactional database systems and data stream management systems have been thoroughly investigated over the past decades. While both system approaches follow completely different data processing models, i.e., push and pull based data forwarding, transactional stream processing combines both models. This means that stream queries writing to tables represent sequences of transactions and at the same time stream or batch queries on such tables get transaction isolation. In this talk, we present the PipeFabric framework – a lightweight publish-subscribe framework optimized for low-latency processing which comprises a library of operators for data stream processing including windows, aggregates, grouping, joins, CEP, matrix operations and a basic DSL for specifying dataflows. In addition to vectorized and data-parallel processing, PipeFabric provides support for tables as sinks and sources for streams as well as for maintaining persistent states of operators such as windows, aggregates, CEP or mining models. In this talk, we discuss challenges of persistent state management in low latency stream processing and sketch ideas for addressing these challenges.

3.26 Streams and Tables in Apache Kafka’s Streams API

Matthias J. Sax (Confluent Inc – Palo Alto, US)

License © Creative Commons BY 3.0 Unported license
© Matthias J. Sax

Apache Kafka introduces a novel approach for data stream processing compared to existing systems. Most state-of-the-art stream processing system use the abstraction of record- or fact-streams, that are append only sequences of immutable data items as first class citizen. Apache Kafka introduces a second type of data stream, called a changelog stream. A changelog stream is an append only sequence of updates and thus it models an evolving collection of data items. This evolving collection of data items can also be described as a table or materialized view and the individual records in the changelog stream are inserts/updates/deletes into this table. In contrast to a “static” database table, a Kafka table can be described as a continuously updating materialized view using the changelog topic as a kind of database redo log. This model opens a new processing paradigm for data stream processing and bridges the gap between static database tables and dynamic data streams. However, the current understanding of the duality is streams and tables is limited and we lack a semantically sound model that allows to define operator semantics that allows to reason about a computation or to do relational-style query optimization like operator reordering.

3.27 IoT Stream Processing Tutorial

Martin Strohbach (AGT International – Darmstadt, DE), Alexander Wiesmaier, and Arno Mittelbach

License © Creative Commons BY 3.0 Unported license
© Martin Strohbach, Alexander Wiesmaier, and Arno Mittelbach

Main reference Vincenzo Gulisano, Zbigniew Jerzak, Roman Katerinenko, Martin Strohbach, Holger Ziekow: “The DEBS 2017 Grand Challenge”, in Proc. of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017, pp. 271–273, ACM, 2017.

URL <http://dx.doi.org/10.1145/3093742.3096342>

This tutorial covers the class of Internet of Things (IoT) streaming applications, i.e. applications that are concerned with interpreting and conceptualizing sensor data in real-time. It focuses on applications from two distinct domains. The first domain relates to Sports and Entertainment in which quantifiable insights about sports and entertainment events are created from sensors deployed at a venue. The second domain relates to Industry 4.0 in which sensor data from production machines is used to reduce energy costs and operations and maintenance costs.

The application examples are based on commercial deployments that run on top of AGT International’s Internet of Things Analytics (IoTA) platform. IoTA is an IoT-based AI platform that provides cognitive and emotional computing skills to understand complex physical environments in real-time.

With the applications described for sports and entertainment we illustrate the specific characteristics of IoT streaming applications and the associated challenge of choosing an appropriate streaming infrastructure. This choice is influenced by the lack of standardized stream processing query languages, the multitude of available distributed streaming processing systems, required flexibility for a wide range of programming languages in which IoT analytics

are being implemented, the focus on low latencies and a large number of shortlived processing pipelines, and the need to map processing results to a semantic data model.

We use the applications for Industry 4.0 to illustrate how stream processing applications can be benchmarked using the HOBBIT benchmarking platform. We describe applications in which we implemented solutions for reducing electricity costs for industrial customers by predicting energy peaks and applications in which we detect anomalies in machine states. We describe how we use the HOBBIT benchmarking platform in order to find anomalies from production machines in data streamed as RDF. The platform was used as part of this year's DEBS Grand Challenge.

3.28 Quantifying and Detecting Incidents in IoT Big Data Analytics

Hong-Linh Truong (TU Wien, AT)

License © Creative Commons BY 3.0 Unported license
© Hong-Linh Truong

Systems for IoT Big data analytics are extremely complex. Different software components at different software stacks from different infrastructures and providers are involved in handling different types of data. Various types of incidents may occur during execution of such a big data analytics due to problems occurring in software stacks, the data itself, and processing algorithms. Here incidents reflect unexpected situations that might happen within data themselves, machine learning algorithms, data pipelines, and underlying big data services and computing platforms. It is important to address any incident that prevents the pipeline running correctly or producing the expected quality of analytics. In this presentation, we show the motivation for quantifying, monitoring and analytics of incidents in IoT big data analytics systems and discuss our plan to tackle this important research.

3.29 Data Management of Big Spatio-temporal Data from Streaming and Archival Sources

Akrivi Vlachou (University of Thessaly – Lamia, GR)

License © Creative Commons BY 3.0 Unported license
© Akrivi Vlachou

An ever-increasing number of critical applications generate, collect, manage and process spatio-temporal data related to the mobility of entities in different domains. In this talk, an overview of the EU-funded project datAcron (<http://www.datacron-project.eu/>) is presented, which addresses time-critical mobility forecasting in maritime and aviation domains using Big Data analytics, focusing mainly on data management aspects. We describe a framework for semantic integration of big mobility data with other data sources, which is necessary for facilitating data analysis tasks, towards a unified representation of such data. First, data transformation from a wide variety of heterogeneous streaming and archival sources to a common representation (RDF) is performed. This is a typical situation in the analysis of mobility data, such as maritime and aviation, where streaming position data of moving objects need to be associated with static information (such as crossing sectors, protected geographical areas, weather forecasts, etc.) in order to provide semantically enriched trajectories. Next, spatio-temporal link discovery between spatio-temporal entities from diverse data sources is

performed. Finally, our framework supports RDF queries that combine historical, static and streaming data. In this talk, we present an overview of the functionality of our framework as well as the technical challenges that are posed.

3.30 Stream Processing with Apache Apex

Thomas Weise (Mountain View, US)

License © Creative Commons BY 3.0 Unported license
© Thomas Weise

Our environment is generating increasing volumes of data from a rapidly growing number of sources like mobile devices, sensors, industrial machines, web logs and more. Organizations are looking to convert these continuous streams of data into insights and competitive advantage, which requires systems that can process data at scale, with minimum delay and with accuracy.

The stream processing space has been evolving rapidly and adoption for real-world, business critical use cases is increasing. A new generation of systems supports large-scale, high-throughput, low-latency processing with correctness guarantees. Apache Apex, presented here, is one of these stream processing systems. The project started in 2012 with the vision to provide an alternative to MapReduce on Apache Hadoop YARN for low-latency processing. Originally a proprietary product, the project was open sourced as Apache Software Foundation project in 2015. Apex has been one of the innovators in the stream processing space with features such as distributed checkpointing, dynamic resource allocation/scaling and dynamic modification of the processing graph.

3.31 Lifetime-Based Memory Management in Distributed Data Processing Systems

Yongluan Zhou

License © Creative Commons BY 3.0 Unported license
© Yongluan Zhou

Joint work of Lu Lu, Xuanhua Shi, Yongluan Zhou, Xiong Zhang, Hai Jin, Cheng Pei, Ligang He, Yuanzhen Geng
Main reference Lu Lu, Xuanhua Shi, Yongluan Zhou, Xiong Zhang, Hai Jin, Cheng Pei, Ligang He, Yuanzhen Geng: “Lifetime-Based Memory Management for Distributed Data Processing Systems”, PVLDB 9(12): 936–947, 2016.

URL <http://www.vldb.org/pvldb/vol9/p936-lu.pdf>

In-memory caching of intermediate data and eager combining of data in shuffle buffers have been shown to be very effective in minimizing the re-computation and I/O cost in distributed data processing systems like Spark and Flink. However, it has also been widely reported that these techniques would create a large amount of long-living data objects in the heap, which may quickly saturate the garbage collector, especially when handling a large dataset, and hence would limit the scalability of the system. To eliminate this problem, we propose a lifetime-based memory management framework, which, by automatically analyzing the user-defined functions and data types, obtains the expected lifetime of the data objects, and then allocates and releases memory space accordingly to minimize the garbage collection overhead. In particular, we present Deca, a concrete implementation of our proposal on top of Spark, which transparently decomposes and groups objects with similar lifetimes into byte arrays and releases their space altogether when their lifetimes come to an end. An extensive

experimental study using both synthetic and real datasets shows that, in comparing to Spark, Deca is able to 1) reduce the garbage collection time by up to 99.9%, 2) to achieve up to 22.7x speed up in terms of execution time in cases without data spilling and 41.6x speedup in cases with data spilling, and 3) to consume up to 46.6% less memory.

4 Working groups

4.1 Working Group: Languages and Abstractions

Martin Hirzel (IBM TJ Watson Research Center – Yorktown Heights, US)

License © Creative Commons BY 3.0 Unported license
© Martin Hirzel

Based on the definitions and survey from the corresponding tutorial, this working group identified and described three challenges faced by stream processing languages.

Variety of data models is a challenge for stream processing languages. A data model organizes elements of data with respect to their semantics, their logical composition into data structures, and their physical representation. Producers and consumers of streams to and from a streaming application dictate data models it must handle, and the application's own conversion and processing needs drive additional data-model variety. There is no consensus on what a stream data item is. At one extreme, in StreamIt, each data item is a simple number, while at the other extreme, C-SPARQL streams entire self-describing graphs. Streaming languages have so far failed to consolidate on a data model because data-model variety is a difficult challenge.

Data-model variety causes streaming-specific issues, since the data model affects the speed of serialization, transmission, compression, and dynamic checks for the presence or absence of certain fields, and because the online setting leaves no time for separate batch data integration. Some stream processing languages are designed around their data model, e.g., CQL on tuples or path expressions on XML trees. Furthermore, the data model enables streaming-language compilers to provide helpful error messages and optimizations.

The goal is for streaming languages to let the programmer use the logical data model they find most convenient while letting the compiler choose the best physical representation. Metrics of success are the expressive power of the language along with its throughput, latency, and resource consumption.

Veracity with simplicity is a challenge for stream processing languages. Veracity means producing accurate and factual results, and simplicity means avoiding unnecessary language complexity. There are several reasons why streaming veracity is hard. Sensors producing input data have limited precision, energy, and memory. In long-running and loosely-coupled streaming applications, sources come and go. And approximate stream algorithms and stream mining introduce additional uncertainty. This is compounded by the lack of ground truth in an online setting, and by the difficulty of anticipating and testing every eventuality.

Veracity causes streaming-specific issues, since it requires accurate real-time responses without having seen all the data, and because the online setting leaves no time for separate batch data cleansing. Also, streaming is often used in a distributed setting, where there can be no global clock. Some streaming languages are explorations in handling uncertainty on top of stream-relational algebra, but restricting stream operators to support retraction or uncertainty propagation limits expressiveness and raises complexity. A more general solution might use probabilistic programming to handle uncertainty in a principled way.

The goal is for streaming languages to help minimize compounding uncertainty by being quality-aware and adaptive while remaining simple, expressive, and fast. This inherently leads to multiple metrics (e.g., precision, recall, throughput, latency) and harder-to-quantify objectives (simplicity, expressiveness). One can maximize one set of metrics while satisficing a threshold on the others, or one can seek Pareto-optimal solutions.

Adoption is a challenge for stream processing languages: while there are many languages, none have reached broad acceptance and use. The community should care about adoption of streaming languages because it would drive adoption of streaming technologies in general. A widely-adopted language is more attractive for students to learn, leading to a bigger pool of skilled people to hire for companies. Furthermore, a widely-adopted language would drive more mature libraries, tools, benchmarks, and optimizations. The lack of a dominant language indicates that adoption is a difficult goal.

Streaming as a domain is young, fast-moving, and prone to vendor lock-in. At the same time, not only is there no consensus on a streaming language, there is not even a consensus on which language features are most important and which can be omitted to reduce complexity. Furthermore, several recent streaming systems have a DSEL (domain-specific embedded language), which tends to have less well-isolated semantics and more host-language dependencies than a stand-alone DSL (domain-specific language).

The goal is for the community to agree upon one or a few languages that get widely adopted. Metrics for language adoption include lines of code, as well as mentions in resumes, job posting, courses, and support forums. Adoption can also be measured by the number of systems that support a language, open-source and open-governance implementations, and ultimately, an industry standard.

We hope this summary of our working group discussion helps guide future streaming-language research in novel and impactful directions.

4.2 Working Group: Systems and Applications

Tilman Rabl (TU Berlin, DE)

License  Creative Commons BY 3.0 Unported license
© Tilman Rabl

In this working group, participants discussed characteristics and open challenges of stream processing systems. The discussions mainly focused on the topics state management, transactions, and pushing computation to the edge.

State management – Modern streaming systems are stateful, which means they can remember the state of the stream to some extent. A simple example is a counting operator that counts the number of elements seen so far. While even simple state like this poses several challenges in streaming setups (such as fault tolerance and consistency), many use cases call for more advanced state management capabilities. An example is the combination of streaming and batch data. This is for example required when combining the history of a user with her current activity or when finding matching advertisement campaigns with current activity a popular example of such a setup is modeled in the Yahoo! Streaming Benchmark [2]. Today, most setups deal with such challenges by combining different systems (e.g., a key value store for state and a streaming system for processing), however, it is desirable to have both in a single system for consistency and manageability reasons.

State can be considered the equivalent of a table in a database system. As a result,

besides the combination of stream and state, several high level operations can be identified: conversion of streams to tables (e.g., storing a stream), conversion of tables to streams (e.g., scanning a table), as well operations only on tables or streams (joins, filters, etc.). The management of state opens the design space in between existing stream processing systems and database systems, which has only been partially explored by current systems. In contrast to database systems, stream systems typically operate in a reactive manner, i.e., they have no control over the incoming data stream, specifically, they do not control and define the consistency and order semantics in the stream. This requires advanced notions of time and order as for example specified for streams in the dataflow model [1], for state and stream this remains an open field of research.

Transactions

A further discussion topic where transactions in stream processing systems. The main difference between traditional database transactions and stream processing transactions is that in databases the computation moves and data stays (in the system), in stream processing systems the computation stays and the data moves to the computation (and out again).

Considering state management, the form of transactions as applied in databases can also be used in a stream processing system, if the state is managed in a transactional way. However, the operations on streams themselves can be transactional and then we can differentiate between single tuple transactions and multiple tuple transactions. For multiple tuple transactions, the transaction can only commit when all tuples are consumed. The tuples then have to pass the whole operator graph or at least the transactional subgraph.

The semantics of transactions on streams is currently still an open field of research.

By *pushing computation to the edge* of a network, stream processing can be highly distributed and decentralized. This is very useful when preprocessing or filtering can be done without a centralized view of the data. Especially, in setups with high communication cost or slow connections (e.g., mobile connections), it makes sense to not send all data to a central server, but distribute the computation. A logical first step is filtering, but aggregations and even more complex operations can be pushed to the edge, if possible. Many modern scenarios prohibit centralized data storage, which further encourages distributed setups with early aggregations. Primary points of research are the declarativity for specifying highly distributed data processing programs and the architecture of systems to support these use cases.

Other topics discussed were ad hoc queries and graph stream processing. Most current systems only discuss long running queries, but in many use cases (e.g., sports, automotive) streams can be short lived as can be stream queries.

References

- 1 T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8:1792–1803, 2015.
- 2 S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1789–1792, 2016.

Participants

- Pramod Bhatotia
University of Edinburgh, GB
- Albert Bifet
Telecom ParisTech, FR
- Michael H. Böhlen
Universität Zürich, CH
- Angela Bonifati
University Claude Bernard –
Lyon, FR
- Jean-Paul Calbimonte
HES-SO Valais – Sierre, CH
- Paris Carbone
KTH Royal Institute of
Technology, SE
- Emanuele Della Valle
Polytechnic University of
Milan, IT
- Javier D. Fernández-García
Wirtschaftsuniversität Wien, AT
- Ashish Gehani
SRI – Menlo Park, US
- Manfred Hauswirth
TU Berlin, DE
- Günter Hesse
Hasso-Plattner-Institut –
Potsdam, DE
- Martin Hirzel
IBM TJ Watson Research Center
– Yorktown Heights, US
- Ali Intizar
National University of Ireland –
Galway, IE
- Asterios Katsifodimos
TU Delft, NL
- Nikos Katsipoulakis
University of Pittsburgh, US
- Henning Kropp
Hortonworks – München, DE
- Danh Le Phuoc
TU Berlin, DE
- Alessandro Margara
Polytechnic University of
Milan, IT
- Gianmarco Morales
QCRI – Doha, QA
- Christoph Quix
Fraunhofer FIT –
Sankt Augustin, DE
- Tilmann Rabl
TU Berlin, DE
- Sherif Sakr
KSAU-HS – Riyadh, SA
- Kai-Uwe Sattler
TU Ilmenau, DE
- Matthias J. Sax
Confluent Inc – Palo Alto, US
- Martin Strohbach
AGT International –
Darmstadt, DE
- Hong-Linh Truong
TU Wien, AT
- Akrivi Vlachou
University of Thessaly –
Lamia, GR
- Thomas Weise
Mountain View, US
- Yongluan Zhou
University of Copenhagen, DK

