LifeStream: A High-performance Stream Processing Engine for Periodic Streams

Extended Abstract

Anand Jayarajan[1,4], Kimberly Hau[1], Andrew Goodwin[2,3], Gennady Pekhimenko[1,4]
[1]University of Toronto, [2]The Hospital for Sick Children, [3]University of Sydney, [4]Vector Institute

1. Motivation

Hospitals collect large amount of physiological data from intensive care patients every day. Recently, researchers has been making considerable progress in using statistical and machine learning algorithms on physiological data to gain more insights and provide improved medical diagnoses [13, 5]. Availability of efficient data processing systems is crucial in enabling this upward trend in the healthcare industry. Even though big data processing has been a well-studied area, from our experience closely working with the clinicians, data analysts, and machine learning researchers at The Hospital for Sick Children[1] [6] at Toronto, Canada, we recognize several new requirements that make physiological data processing both unique and challenging.

Simple and flexible programming interface with temporal logic support. Raw physiological data collected from the patients contains high degree of *noise* and *discontinuities*. Therefore, the data should to go through a series of data cleaning operations and transformations before it can be used for meaningful analyses. In certain cases, researchers also need to compute derived variables from the raw data. The choice of operations performed on physiological data are highly use-case specific and most of operations follow a strong notion of the temporal ordering of the data. This necessitates the data processing system to have a simple and flexible programming interface with a rich temporal query language support.

Efficient hardware resource utilization. Common physiological data processing use cases involve highly compute intensive operations and need to process terabytes of data [7]. However, most hospitals only have limited computing infrastructure in-place as maintaining large machine clusters require specific expertise that is hard to come by. Using cloud resources is also usually not a viable option because the data collected and stored at the hospitals are fully-identified, which limits the movement of datasets outside the hospital facilities in order to avoid the risk of leaking personal information about the patients [8]. Therefore, the physiological data processing system has to provide *high performance* by efficiently utilizing the available *limited hardware resources*.

We observe that the currently available data processing solutions [14, 1, 3] fail to match either of these two key requirements. Our main **goal** in this work is to build a physiological data processing system that is both *easy to program* and provides *high performance* even under hardware resource constraints.

[1]Canada's largest pediatric hospital

2. Limitations of the State of the Art

Since physiological data is produced in a streaming manner, stream processing engines are a natural choice for building the physiological data processing pipelines. Most of the contemporary stream processing engines (e.g., Spark streaming [2], Apache Beam [1], and Microsoft Trill [4]) provide a simple and flexible declarative programming interface with rich set of temporal operators. However, our experiments reveal that even the state of the art streaming engines fall short in performance compared to hand-optimized implementations available in the numerical libraries (e.g., SciPy [10], NumPy [9], and Scikit-learn [11]) on commonly used physiological data transformation operations by $1-2$ orders of magnitude because of their sub-optimal hardware utilization (see Section 3 in the main paper for more details).

Numerical libraries, on the other hand, does provide large ecosystem of highly efficient data processing operations with hand-tuned implementations, however, they fail to provide simple and flexible programming interfaces that are suitable for physiological data processing due to lack of implicit notion of temporal ordering of the data, rigid API specifications, and lack of a common data abstraction across different libraries. This makes using numerical libraries an undesirable choice for building physiological data processing pipelines in terms of both ease of programming and ease of maintenance.

3. Key Insights

In this paper, we introduce LifeStream, a new high performance stream processing engine for physiological data. We make a key observation that data streams like physiological data are produced at regular intervals (periodic) which can be exploited to perform end-to-end optimizations on the stream processing pipelines. LifeStream hits the sweet spot in both *ease of programming* by providing a rich SQL-like temporal query language support (Listing 1) and *performance* by employing three key optimizations utilizing the periodic nature of the data.

LifeStream provides a data abstraction consisting of a stream of events in chronological order. An *event* is a single unit of data with three fields: (i) a user-defined *payload*, (ii) a *sync time* dictating the time from which the event is active, and (iii) a *duration* defining the active lifetime of the event. Even though we are primarily targeting physiological data, any streaming data that can be represented in this format can take advantage of the benefits of LifeStream.

Next, we derive the following two key mathematical prop-

```
var left = sig500
    .Multicast(s => s
      .Select(e => e.val) // select signal value
      // compute mean and subtract from values
      .Join(s.TumblingWindow(100).Mean(),
        (val, mean) => val - mean));
var right = sig200
    .Select(e => e.val); // select signal value
var output = left
    // join with sig200 values
    .Join(right, (l, r) => new {l, r});
```

Listing 1: An example query written in LifeStream

erties of the temporal operations on periodic streams.

Linearity property: *The sync time of events in the output stream of a temporal operator is a linear transformation of that of the events in the input stream.*

This property allows LifeStream to map the events in the output stream of an operator to the corresponding parent events in the input stream(s). This property also ensures that the output stream generated by the temporal operators are also periodic. Therefore, the linearity property is preserved throughout the entire pipeline and can be used to track the entire lineage of every event produced during query execution. We call this mechanism *event lineage tracking*.

Bounded memory footprint: *The memory footprint of a temporal operator is bounded by the size of its inputs.*

Even though streaming datasets are long and continuous sequence of events, the temporal operations applied on the streams only need to consider events in a small interval at any given time. We define the dimension(s) of a temporal operator as the size of the interval(s) on which it operates. On top of this, periodic streams can only have at most one event active at any given point in time. Therefore, the total number of events any temporal operators has to process is bounded by its dimensions.

We use the above properties and propose the following three key optimizations during query compilation and execution:

Locality tracing: Most contemporary streaming engines take advantage of the locality of stream processing *only* at the primitive operator-level. LifeStream, on the other hand, uses *linearity property* to estimate the data locality of the end-to-end pipeline through a process called *locality tracing* during query compilation. The key idea behind locality tracing is to adjust the dimensions of each temporal operators in the query such that the intermediate results are consumed immediately by the subsequent operator(s). This ensures the end-to-end data locality is maximized.

Static memory allocation: Once the dimensions of every temporal operators are finalized by locality tracing, LifeStream estimates the upper bound of the memory required for each operation in the pipeline using *bounded memory footprint property* and preallocates the memory for all the intermediate results produced in the stream. This almost completely eliminates the runtime memory allocation and deallocation overhead commonly observed in other streaming engines [12].

Targeted query processing: Most streaming engines perform stream processing in an eager fashion where each temporal operator performs the transformation on the input as soon as it receives the data, and immediately passes it on to the next operator down the pipeline, regardless of whether the next operation would actually need to process that data or not. This could lead to many redundant computations when multiple signals are joined together as the number of mutually overlapping events are generally far fewer than the total number of events in the streams (Figure 2 in the main paper), rendering any computations on non-overlapping events wasteful. To avoid such redundancy we introduce targeted query processing, which uses the *event lineage tracking* mechanism to selectively execute the query on specific regions of the input stream where an output is expected to be produced.

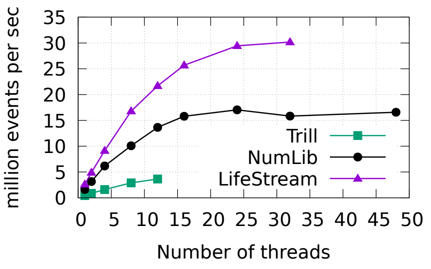

Figure 1: Performance comparison of LifeStream against Trill and numerical libraries (NumLib)

4. Key Results and Contributions

- We showcase the unique challenges involved with physiological data processing and propose solutions that are evaluated on real datasets and workloads used in major hospitals.
- We derive two key properties of temporal operations on periodic streams, namely linearity and bounded memory footprint, which are leveraged to propose three key optimizations: (i) locality tracing, (ii) static memory allocation, and (iii) targeted query processing, that can significantly improve the hardware utilization and query execution performance compared to the state of the art streaming engines.
- We propose LifeStream, a new high performance stream processing engine with extended temporal query language support. As shown in Figure 1, LifeStream can outperform state of the art streaming engines by as much as 7.5× and hand-optimized numerical libraries by as much as 3.2× on the end-to-end data processing performance while also providing much more flexible programming interface.

5. Conclusion

In this paper, we propose and implement LifeStream, a new high performance stream processing engine for periodic streams. We show that LifeStream can surpass the performance of hand tuned numerical libraries without sacrificing the flexibility of the programming interface available in modern stream processing engines. LifeStream is open sourced.[2]

[2]https://github.com/anandj91/LifeStream

References

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam 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] 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*, SIGMOD '18, page 601–613, New York, NY, USA, 2018. Association for Computing Machinery.

[3] 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.

[4] 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. *Proc. VLDB Endow.*, 8(4):401–412, December 2014.

[5] Ambika Choudhury and Deepak Gupta. *A Survey on Medical Diagnosis of Diabetes Using Machine Learning Techniques: IC3 2018*, pages 67–78. 01 2019.

[6] The Hospital for Sick Children. https://www.sickkids.ca/en/about/about-sickkids/.

[7] Andrew J Goodwin, Danny Eytan, Robert W Greer, Mjaye Mazwi, Anirudh Thommandram, Sebastian D Goodfellow, Azadeh Assadi, Anusha Jegatheeswaran, and Peter C Laussen. A practical approach to storage and retrieval of high-frequency physiological signals. *Physiological Measurement*, 41(3):035008, apr 2020.

[8] HHS. Your rights under hipaa.

[9] Travis Oliphant. Numpy.

[10] Travis Oliphant, Pearu Peterson, and Eric Jones. Scipy.

[11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[12] Gennady Pekhimenko, Chuanxiong Guo, Myeongjae Jeon, Peng Huang, and Lidong Zhou. Tersecades: Efficient data compression in stream processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 307–320, Boston, MA, July 2018. USENIX Association.

[13] Jonathan G. Richens., Ciarán M. Lee, and Saurabh Johri. Improving the accuracy of medical diagnosis with causal machine learning. *Nature Communications*, 11(1):3923, Aug 2020.

[14] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.