StreamMapReduce

When Stream Processing crosses MapReduce

André Martin*, Andrey Brito†, Christof Fetzer*

*Technische Universität Dresden, Dresden, Germany - Email: andre.martin@tu-dresden.de / christof.fetzer@tu-dresden.de
†Universidade Federal de Campina Grande, Campina Grande, Brazil - Email: andrey@computacao.ufcg.edu.br

***Abstract*—Although Event Stream Processing (ESP) systems exit for already more than a decade, we recently witness a true renaisance for ESP systems that have adopted the popular MapReduce paradigm. In this white paper, we advocate for the StreamMapReduce approach as it allows a (i) quick and easy transition of legacy MapReduce-based applications to ESP, (ii) simplifies the implementation of fault tolerance mechanisms, and (iii) elasticity in order to operate in nowadays cloud environments. We will furthermore showcase two real world applications from the area of SmartGrids and geo-spatial data stream analysis where the StreamMapReduce approach has been successfully applied.**

***Keywords*—ESP, event stream processing, programming model, stateful event processing, fault tolerance, elasticity**

I. Introduction

ESP systems exist for nearly a decade by now: The first generation of such systems evolved mainly as an extension of database systems such as TelegraphCQ [1] while in the following years, a second generation of ESP systems evolved where ESP became its own independent branch with the advent of the Borealis system [2]. Although those systems were already equipped with novel mechanisms such as load shedding, fine grained load distribution and fault tolerance, they all have a CQL-like programming interface in common. However, the advent of Google's MapReduce [3] approach paved the way for a new generation of ESP systems. Mostly driven by industry, a number of MapReduce-inspired open-source ESP systems such as Apache S4 [4], Storm [5] and Samza [6] evolved over the past five years gaining an enormous traction. But, not only industry pushed for the new MapReduce-like ESP approach as the evolvement of ESP systems such as SEEP [7] and StreamMine3G [8] originating from academia shows.

II. Programming Model & Application Examples

The StreamMapReduce programming model is quite trivial as it consists only of a single method, a user-defined function (UDF) often called `process()`. The method takes the (i) incoming event, which can either be provided in un-serialized or serialized form, (ii) a collector object to emit events to the following stage, and (iii) a state object in order to provide stateful ESP. Note that some modern ESP systems such as SEEP do not provide an explicit state object as they require that state is maintained as member variables of the class providing the `process()` method for the event processing loop. Once the user has implemented the operators, a topology defines the flow of events, i.e., how events traverse those operators carrying out the complete application. The advantage of such a design is that it provides scalability similar as in MapReduce where the input stream of an operator can be easily partitioned by a simple hash-based or user-provided partitioner function.

Although the proposed programming model may seem to require a lot of programming effort when working on time series contrary to CQL-like approaches that provide already implicit window semantics, authors in [9] have shown that window semantics can be easily provided and adopted using the StreamMapReduce approach.

Moreover, the simplicity of the MapReduce programming model and the provided state object allows the implementation of standard CQL-like operators such as joins and aggregations that can run on top of ESP systems as a thin layer as authors have shown in [10]. This abstraction allows users to utilize standard CQL operators for convenience paired with UDFs where needed.

Although CQL operators provide a convenient way of processing time series, many real world applications such as provided by the annual DEBS challenge require also access to historical data. For example, in the DEBS 2014 challenge, application developers were asked to provide a short term load prediction using historical data as well as detecting outliers using a sliding window in the context of SmartGrids. Authors in [11] have shown that the given problem can be solved efficiently using the StreamMapReduce approach reducing the size in code significantly in contrast to CQL-based approaches. The approach presented provides not only scalability but also elasticity which allows the application to run also in cloud environments reducing monetary costs as authors have shown in [12].

However, the StreamMapReduce paradigm proved to be successful also in other application areas such when processing geo-spatiel data streams in near real-time: In the DEBS 2015 challenge, application developers were challenged to provide an application that continuously provides a *top-k* of the most frequently driven routes and most profitable areas within a sliding $30\,mins$ window based on the provided stream of taxi rides. Although the operation on the provided time series calls for a CQL-like approach at its first sight, the complexity for the computation of the profit for an area as well the ranking of the most frequently driven routes calls for a more flexible approach such as StreamMapReduce where custom data structures can be utilized as authors have shown in [13].

III. Fault Tolerance & Elasticity

In StreamMapReduce, we consider an operator as a *black-box* where the ESP system is neither aware of its functionality nor the properties of the associated state. Using the black-box approach, fault tolerance can be implemented in the following way: The user solely needs to provide a (i) serialize and de-serialize method for the operator state (if the operator is stateful) and for the incoming and outgoing events, and (ii) timetamps associated with every event. Using the provided methods, the ESP system can now offer fault tolerance based on checkpointing and logging (*rollback recovery*) by performing periodic checkpointing of the operator state as well as maintaining an in-memory log of events for a replay of in-flight events in the event of a system crash. Using the associated timestamps, events can be processed *deterministically* to recover precisely when using rollback recovery, or alternatively for *active replication* using the state machine replication approach which results in an almost instantaneous recovery.

Although the black-box approach provides *exactly-once* processing semantics which is per-se expensive due to event ordering, user-provided annotations can be used in order to reduce the overhead imposed by fault tolerance as authors have shown in [14]. Such an optimization can be applied if for example an operators is commutative and works on tumbling windows where the ESP system is then allowed to omit to enforce a strict event ordering within such windows without distorting the final result. Using this approach, authors have shown that the throughput can be considerably increased.

Since the black-box approach requires the user to provide operator state and events in binary form, many of those mechanisms used initially for fault tolerance can be reused for elasticity. Authors in [7], [12] have shown that the checkpointing mechanism originally designed for state persistence can also be used to move stateful operators to new nodes in order to expand or contract clusters based on the current demand on resources. Hence, the StreamMapReduce abstraction allows ESP systems to scale elastically while providing fault tolerance with exactly-once processing semantics at the same time.

Confidentiality has been always an issue when running applications in cloud environments. Using the black-box approach of StreamMapReduce in concert with the new upcoming Intel SGX [15] extension, we can establish trust by simply having the operator code executed in a so called *enclave*, a trusted execution environment. We therefore believe that StreamMapReduce is a suitable approach for trusted, elastic and fault tolerant stream processing.

IV. Conclusion & Summary

In this paper we presented the StreamMapReduce approach and showcased its benefits using several real world applications originating from different application domains. We highlight that the flexibility of such a paradigm allows users to intermix custom written operators in form of UDFs with higher level abstractions for stream processing such as CQL that can run on top of StreamMapReduce-based ESP systems as an additional thin layer. By considering operators and state as black-boxes, we can furthermore provide fault tolerance using exactly-once processing semantics. Moreover, it is possible to reuse several mechanisms originally designed for fault tolerance for elastic scaling of applications to run cloud environments such as Amazon EC2. To establish trust in those environments, we propose the use of Intel SGX. Especially the strong demand in industry during the past few years confirms our belief that the StreamMapReduce approach can be considered the next generation of ESP systems.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Program (FP7/2012-2015) under grant agreement number 318809 (LEADS) and by the German Excellence Initiative, Center for Advancing Electronics Dresden (cfAED), Resilience Path.

References

[1] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 668–668. [Online]. Available: http://doi.acm.org/10.1145/872757.872857

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *In CIDR*, 2005, pp. 277–289.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[4] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 170–177.

[5] "Apache storm - distributed and fault-tolerant realtime computation," https://storm.incubator.apache.org/, 2014.

[6] "Apache samza - distributed stream processing framework," http://samza.incubator.apache.org/, 2014.

[7] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch, "Integrating scale out and fault tolerance in stream processing using operator state management," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 725–736.

[8] A. Martin, A. Brito, and C. Fetzer, "Scalable and elastic realtime click stream analysis using streammine3g," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: ACM, 2014, pp. 198–205.

[9] A. Brito, A. Martin, T. Knauth, S. Creutz, D. Becker, S. Weigert, and C. Fetzer, "Scalable and low-latency data processing with stream mapreduce," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 48–58.

[10] T. Heinze, Z. Jerzak, A. Martin, L. Yazdanov, and C. Fetzer, "Fault-tolerant complex event processing using customizable state machine-based operators," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12. New York, NY, USA: ACM, 2012, pp. 590–593.

[11] A. Martin, R. Marinho, A. Brito, and C. Fetzer, "Predicting energy consumption with streammine3g," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: ACM, 2014, pp. 270–275.

[12] A. Martin, R. Silva, A. Brito, and C. Fetzer, "Low cost energy forecasting for smart grids using stream mine 3g and amazon ec2," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, ser. UCC '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 523–528. [Online]. Available: http://dx.doi.org/10.1109/UCC.2014.78

[13] A. Martin, A. Brito, and C. Fetzer, "Real time data analysis of taxi rides using streammine3g," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: ACM, 2015, pp. 269–276. [Online]. Available: http://doi.acm.org/10.1145/2675743.2772584

[14] A. Martin, T. Knauth, S. Creutz, D. Becker, S. Weigert, C. Fetzer, and A. Brito, "Low-overhead fault tolerance for high-throughput data processing systems," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 689–699.

[15] "Intel sgx," https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx, 2015.