of the user's control. When considering manual resource scaling deployments, however, provisioned resources are explicitly chosen to meet SLOs. Thus, we expect it to be more relevant to measure how resource demands evolve.

4.4 Capacity as Discrete Values

With our *capacity* scalability metric, we propose to determine the load capacity as a discrete value from a given set of load intensities. In many scalability studies, however, capacity is measured as a continuous value. While this might be feasible for databases [13] or batch processing [6], we consider it to be difficult to achieve in stream processing. To determine capacity as a continuous value, we observe basically two options, both having weaknesses.

The first option is to generate a constant load and measure the throughput, i.e., how much of this load is processed. Although not explicitly described, it looks like this technique was applied in the scalability evaluations of stream processing engines by Karakaya et al. [10] and Nasiri et al. [15] A first weakness of this approach is that the generated load must be sufficiently high as otherwise the throughput would be bounded by the generated load intensity instead of by the capacity of the provisioned resources. Further, it is unclear how much higher the load has to be. As the throughput may vary strongly [7, 11], it may temporarily be higher than the generated load, causing the stream processing engine to not operate at its maximum. Finally, this approach is based on the assumption that for a given amount of provisioned resources the throughput is always the same, independent of the generated load. However, this assumption is questionable, unless explicitly evaluated. It is likely that a high load on the messaging system also influences response times or chunk sizes.

A second option to measure load capacity as continuous values is to steadily increase the load intensity for a given resource amount and determine at which load intensity SLOs are not fulfilled anymore. A similar method is taken by Karimov et al. [11] for their sustainable throughput metric. A weakness of this approach is that dependencies between different load intensities are difficult to rule out. We observe that even with constant load the throughput of stream processing engines varies strongly and, additionally, increases after some warm-up period [7]. To determine a reasonable load capacity, the monitored throughput has therefore to be averaged over some period of time. Furthermore, when increasing the load without restarting experiments, stream processing engines or related software infrastructure might perform optimizations for lower load intensities, which are not ideal for higher loads.

When measuring scalability with our proposed metrics, we therefore strongly recommend to evaluate the slo_s functions in isolated experiments for one resource configuration and a constant load.

5 CONCLUSIONS AND FUTURE WORK

With this paper, we propose two metrics for scalability of distributed stream processing engines, derived from common scalability definitions. While our *demand* metric describes how resource demands evolve with increasing load, our *capacity* metric describes how load capacity evolves with increasing provided resources. Both metrics share the explicit definition of SLOs that always have to be fulfilled and the definition of load intensity and provisioned resources as discrete spaces. We discuss advantages of these decisions in comparison to other metrics as well as compared to each other.

We expect these metrics to lay a solid foundation for benchmarking scalability of stream processing engines. For future work, we also plan to evaluate whether the Universal Scalability Law can be applied to create a ranking of such engines.

ACKNOWLEDGMENTS

This research is funded by the German Federal Ministry of Education and Research (BMBF, grant no. 01IS17084B).

REFERENCES

[1] A. B. Bondi. 2000. Characteristics of Scalability and Their Impact on Performance. In *Proc. International Workshop on Software and Performance*. https://doi.org/10.1145/350391.350432
[2] G. Brataas, N. Herbst, S. Ivanšek, and J. Polutnik. 2017. Scalability Analysis of Cloud Software Services. In *Proc. International Conference on Autonomic Computing*. https://doi.org/10.1109/ICAC.2017.34
[3] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. 2015. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[4] L. Duboc, D. Rosenblum, and T. Wicks. 2007. A Framework for Characterization and Analysis of Software System Scalability. In *Proc. ESEC/FSE*. https://doi.org/10.1145/1287624.1287679
[5] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos. 2020. A Survey on the Evolution of Stream Processing Systems. arXiv:2008.00842 [cs.DC]
[6] N. J. Gunther, P. Puglia, and K. Tomasette. 2015. Hadoop Superlinear Scalability. *Commun. ACM* 58, 4 (2015). https://doi.org/10.1145/2719919
[7] Sören Henning and Wilhelm Hasselbring. 2021. Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures. *Big Data Research* 25 (2021), 100209. https://doi.org/10.1016/j.bdr.2021.100209
[8] N. R. Herbst, S. Kounev, and R. Reussner. 2013. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *Proc. Int. Conference on Autonomic Computing*.
[9] P. Jogalekar and M. Woodside. 2000. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 11, 6 (2000). https://doi.org/10.1109/71.862209
[10] Z. Karakaya, A. Yazici, and M. Alayyoub. 2017. A Comparison of Stream Processing Frameworks. In *Proc. International Conference on Computer and Applications*. https://doi.org/10.1109/COMAPP.2017.8079733
[11] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl. 2018. Benchmarking Distributed Stream Data Processing Systems. In *Proc. International Conference on Data Engineering*. https://doi.org/10.1109/ICDE.2018.00169
[12] D. Kossmann, T. Kraska, and S. Loesing. 2010. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In *Proc. SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/1807167.1807231
[13] J. Kuhlenkamp, M. Klems, and O. Röss. 2014. Benchmarking Scalability and Elasticity of Distributed Database Systems. *Proc. VLDB Endow.* 7, 12 (2014). https://doi.org/10.14778/2732977.2732995
[14] S. Lehrig, H. Eikerling, and S. Becker. 2015. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Int. Conf. Quality of Software Architectures*. https://doi.org/10.1145/2737182.2737185
[15] H. Nasiri, S. Nasehi, and M. Goudarzi. 2019. Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities. *Journal of Big Data* 6, 52 (2019). https://doi.org/10.1186/s40537-019-0215-2
[16] R. Sanders, G. Brataas, M. Cecowski, K. Haslum, S. Ivanšek, J. Polutnik, and B. Viken. 2015. CloudStore – Towards Scalability Benchmarking in Cloud Computing. *Procedia Comput. Sci.* 68 (2015). https://doi.org/10.1016/j.procs.2015.09.225
[17] M. J. Sax, G. Wang, M. Weidlich, and J.-C. Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *Proc. International Workshop on Real-Time Business Intelligence and Analytics*. https://doi.org/10.1145/3242153.3242155
[18] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. 2014. Storm@twitter. In *Proc. SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/2588555.2595641
[19] Vikash, L. Mishra, and S. Varma. 2020. Performance evaluation of real-time stream processing systems for Internet of Things applications. *Future Generation Computer Systems* 113 (2020). https://doi.org/10.1016/j.future.2020.07.012
[20] A. Weber, N. Herbst, H. Groenda, and S. Kounev. 2014. Towards a Resource Elasticity Benchmark for Cloud Environments. In *Proc. International Workshop on Hot Topics in Cloud Service Scalability*. https://doi.org/10.1145/2649563.2649571
[21] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proc. Symposium on Operating Systems Principles*. https://doi.org/10.1145/2517349.2522737