VISP Testbed - A Toolkit for Modeling and Evaluating Resource Provisioning Algorithms for Stream Processing Applications

Christoph Hochreiner

Distributed Systems Group, TU Wien, Vienna, Austria
c.hochreiner@infosys.tuwien.ac.at

Abstract. Inspired by the current transition towards a data-centric society, more and more researchers investigate cost-efficient resource provisioning algorithms for stream processing applications. While these algorithms already cover a variety of different optimization strategies, they are rarely benchmarked against each other. This makes it almost impossible to compare these different algorithms.
To resolve this problem, we propose the VISP Testbed, a toolkit which eases the development of new provisioning algorithms by providing a runtime for stream processing applications, a library of stream processing topologies, and baseline algorithms to systematically benchmark new algorithms.

Keywords: Data Stream Processing, Testbed, Reproducibility

1 Introduction

Due to the current transition towards a data-centric society, the research area on data stream processing gets more and more traction to tackle the challenges regarding the volume, variety and velocity of unbound streaming data [10] as well as different geographic locations of data sources [5]. A common approach, to implement stream processing applications, is to decompose the data processing into single steps, i.e., operators, and compose topologies as shown in Fig. 1. These topologies can then be enacted by Stream Processing Engines (SPEs), like IBM System S [3], Apache Storm [14], Apache Spark [19], CSA [12] or VISP [7].

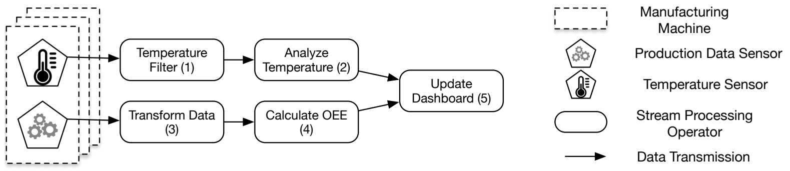

Fig. 1. Stream Processing Topology

The majority of these SPEs is designed, to process a large volumes of data, nevertheless, they often fail to adapt to varying volumes of data. In typical scenarios, like the processing of the sensor data originating from manufacturing machines, as depicted in Fig. 1, the volume of sensor data is often subject to change due to the variable amount of running machines, e.g., due to maintenance downtimes or in error scenarios. This variable volume of data requires different computational resources to process the data in (near) real-time. To comply with the changing resource requirements, it is either possible to over-provision the SPE, i.e., allocate sufficient resources to cover all peaks, or to scale the SPE on demand, i.e., add computational resources only when they are required.

Fig. 2 demonstrates an exemplary scenario for an elastic SPE, which adapts at runtime to the resource requirements of the stream processing application. At the beginning, the incoming data volume is low and it is sufficient to only instantiate one operator of each kind. After some time, the data volume increases and it is required to replicate individual operators to deal with the incoming data volume. As soon as the data volume decreases again, the SPE can release some of the replicated operators while maintaining (near) real-time data processing capabilities. Due to the fact that on-demand resource provisioning is not trivial, several researchers started to investigate different resource provisioning strategies [1, 6, 9, 17].

Although each of these optimization algorithms follows an individual optimization strategy, they have one common challenge when it comes to the evaluation of their algorithm. Up to now, none of the established SPEs provides extensive resource optimization interfaces that can be used to implement and evaluate custom resource provisioning algorithms. While dedicated benchmarking frameworks are available for other areas, like iFogSim [4] for fog environments or Gerbil [15] for semantic entity annotations, there are hardly any projects for resource provisioning algorithms in the stream processing domain. Although there exists several benchmarking projects like streaming-benchmarks[1] or flotilla[2], the scientific community picked up this topic only recently [13]. Nevertheless, to the best of our knowledge there is no framework to benchmark resource provisioning algorithms for SPEs.

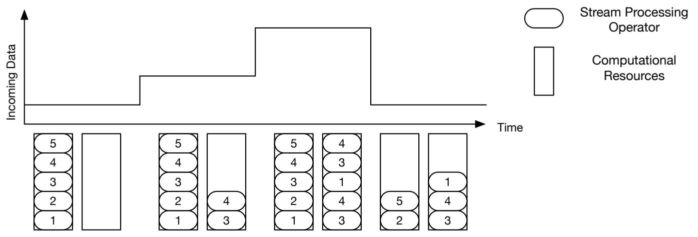

Fig. 2. Resource Requirements for a Deployed Stream Processing Topology

[1] https://github.com/yahoo/streaming-benchmarks
[2] https://github.com/tylertreat/Flotilla

Due to the lack of such a framework, each research group is required to implement an evaluation environment to evaluate their custom algorithm against baseline algorithms instead of state of the art algorithms. To fill this gap, we propose the VISP Testbed as part of our Vienna ecosystem for elastic stream processing, which not only provides a dedicated interface for new resource provisioning algorithms but also different data generation pattern, topologies and baseline algorithms to benchmark new custom algorithms and create reproducible evaluations.

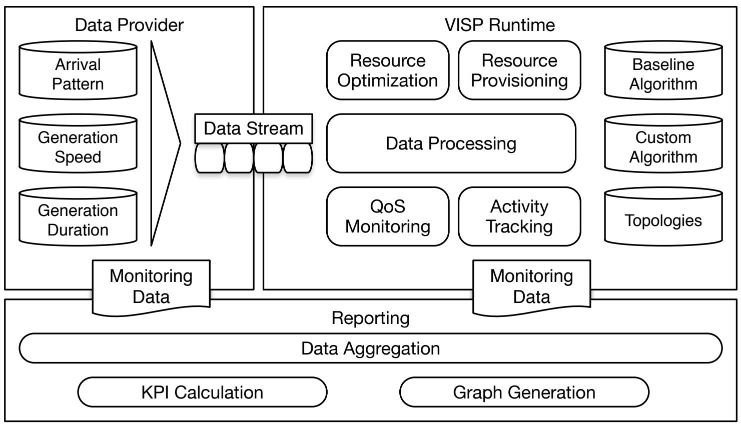

Fig. 3. VISP Testbed

2 System Design

One of the most crucial challenges for realizing a testbed is the data, which is used for the evaluation. This data needs to be recorded from real world systems which makes it very difficult to obtain such data because most data owners do not support the publication of their data. Although there are some data dumps available for scientific purposes, e.g., those published in the course of the DEBS Grand Challenges[3], they are often only suited to evaluate the overall performance of an SPE and not the adoption capabilities of a resource provisioning algorithm. Therefore we decided to implement generic stream processing operators with artificial arrival pattern besides one concrete stream processing topology based on the T-Drive data set [18]. The artificial arrival pattern can be used, to evaluate the adoption capabilities of the resource provisioning algorithms in a clearly defined scenario, before they can be evaluated against real world scenarios without any predefined arrival scenarios. The system design, as shown in Fig 3, consists of three components which are discussed in the remainder of this section.

[3] http://debs.org/?page_id=24

2.1 Data Provider

The Data Provider component[4] is designed to provide a variety of reproducible data streams as input for the VISP Runtime. These data streams can be configured based on their *generation speed*, i.e., how many data items should be generated each second, *generation duration*, i.e., how long should the data be generated and which data generation pattern should be applied. In order to simulate different load scenarios, we have selected four different arrival pattern, based on real world file access pattern [16], as illustrated in Fig 4. These arrival pattern pose different challenges to SPEs as well to its resource provisioning algorithm.

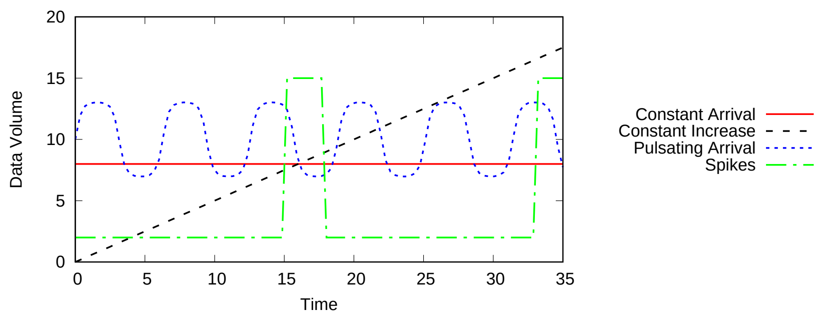

Fig. 4. Arrival Pattern

The simplest arrival pattern is the *constant arrival*, which generates a constant volume of data. This arrival pattern is predestined to test the overall functionality of a resource provisioning algorithm without generating any need for adoption after an initial provisioning.

The *constant increase* pattern is designed to apply stress tests to the SPE and to the resource provisioning algorithm. This pattern is suited to determine the maximal time, for which the resource provisioning algorithm can provide sufficient computational resources for the SPE to comply with given Service Level Agreements (SLAs).

The *pulsating arrival* pattern generates a constantly changing volume of data. This pattern requires the resource provisioning algorithm to constantly update the computational resources for the SPE and is designed to test the adaptation capabilities of the resource provisioning algorithm.

The last arrival pattern is the *spikes* pattern, which has similar properties compared to the constant arrival, apart from short data volume bursts. These short bursts pose high challenges to any resource provisioning algorithm, because they may be prone to allocate a lot of resources for this short time, instead of applying other compensation mechanism like tolerating SLA violations for a short time.

[4] https://github.com/visp-streaming/dataProvider

2.2 VISP Runtime

The core component of the VISP Testbed is the SPE, i.e., the VISP Runtime[5], as presented in our previous work [6, 7]. The VISP Runtime is designed to process incoming data according to a predefined topology, like the one shown in Fig. 1. Besides the actual data processing of the incoming data stream, the VISP Runtime also takes care of *resource provisioning* for operators and the *resource optimization* thereof based on given algorithms. The *Quality of Service (QoS) monitoring* component of the VISP Runtime monitors different aspects of the data processing, like the processing duration of single data items, the latency between two operators or the individual resource consumption of a single operator. Furthermore, the VISP Runtime also features an *activity tracking* component, which tracks all activities within the VISP Runtime, like upscaling or downscaling of a operators and leasing or releasing new computational resources from a cloud resource provider. These metrics can be either used for the resource optimization but also for the Reporting component, which automatically interprets the evaluation. In order to support future evaluations, we provide a basic library of topologies as well as algorithms.

Evaluation Topologies Our basic library of topologies is motivated by the SAP R/3 reference model [2]. Although the SAP R/3 reference model was originally designed to provide a large variety of business processes for benchmarking purposes, we realized, that today's topologies have similar structures and operations, like data replication (AND-operations) or conditional routes (XOR-operations). Based on this realization, we have selected 10 exemplary topologies based on our previous work [8] to evaluate different scenarios. These exemplary topologies can be equipped with different operators, such as an instant forward operator, a data aggregation operator or a busy-waiting operator to trigger a specific CPU or memory load for each data item.

Baseline Algorithms Currently, the VISP Testbed offers two baseline algorithms, whose configuration, e.g., thresholds, can be parametrized if required.

For the first algorithm, we have selected the *fixed provisioning algorithm*, where the resource provisioning is specified before the evaluation starts and does not change, regardless of the actual data volume. This approach allows to model under-provisioning scenarios, where the SPE has not enough resources at hand to process all data in (near) real-time at peak times as well as over-provisioning scenarios, where a segment of the computational resources is not required most of the time. These two scenarios are suitable to provide a lower baseline for the QoS related attributes, i.e., in an under-provisioning scenario, and the upper baseline for the computational cost, i.e., in an over-provisioning scenario.

For the second baseline algorithm, we have selected a *threshold based* algorithm to adopt the computational resources on-demand. This algorithm replicates operators, when a specific Key Performance Indicator (KPI), e.g., processing

[5] https://github.com/visp-streaming/runtime

duration, exceeds a threshold and scales it down when these resources are not required anymore. This algorithm provides a more realistic baseline for custom resource provisioning algorithms than the fixed one because it is able to elastically react to varying incoming data volumes.

2.3 Reporting

The final component for the VISP Testbed is the Reporting component, which is currently integrated within the VISP Runtime. This component aggregates the monitoring data from both the Data Provider and the VISP Runtime to automatically generate evaluation reports. For our evaluation reports we distinguish between textual reports, which feature quantitative KPIs, such as the total cost for computational resources, performed provisioning operations or SLA-compliance for the overall data processing, and a graphical representation. For the graphical representation, the reporting component interprets the monitoring time series of the evaluation and generates diagrams with the help of gnuplot[6]. This graphical representations can the be used for more detailed analysis of the resource provisioning algorithm and to identify further optimization potentials.

3 Conclusion

In this paper, we have presented the VISP Testbed, which provides a basic toolkit to conduct reproducible and comparable evaluations with the goal to support the design of future resource provisioning algorithms. Until now, the VISP Testbed only features a small library of topologies and baseline algorithms, but we plan to extend the topology library with concrete topologies from the manufacturing domain [11].

Furthermore, we plan to implement more resource provisioning algorithms based on our ongoing research as well as those from other researchers to provide more competitive baselines. At last, we also want to provide an additional probing component for the VISP testbed, to identify the minimal resource requirements for a specific stream processing operator, which can then be latter used for the resource provisioning algorithms.

Acknowledgments This paper is supported by TU Wien research funds and by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066).

References

1. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proc. of the 10th Int. Conf. on Distributed and Event-based Systems (DEBS). pp. 69–80. ACM (2016)

[6] http://gnuplot.sourceforge.net

2. Curran, T.A., Keller, G.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Prentice Hall PTR, Upper Saddle River (1997)
3. Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., Doo, M.: SPADE: The System S Declarative Stream Processing Engine. In: 2008 ACM SIGMOD Int. Conf. on Management of Data. pp. 1123–1134 (2008)
4. Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R.: iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. arXiv preprint arXiv:1606.02007 (2016)
5. Hochreiner, C., Schulte, S., Dustdar, S., Lecue, F.: Elastic Stream Processing for Distributed Environments. IEEE Internet Computing 19(6), 54–59 (2015)
6. Hochreiner, C., Vögler, M., Schulte, S., Dustdar, S.: Elastic Stream Processing for the Internet of Things. In: 9th Int. Conf. on Cloud Computing (CLOUD). pp. 100–107. IEEE (2016)
7. Hochreiner, C., Vögler, M., Waibel, P., Dustdar, S.: VISP: An Ecosystem for Elastic Data Stream Processing for the Internet of Things. In: 20th Int. Enterprise Distributed Object Computing Conf. (EDOC). pp. 19–29. IEEE (2016)
8. Hoenisch, P., Schuller, D., Schulte, S., Hochreiner, C., Dustdar, S.: Optimization of complex elastic processes. Trans. on Services Computing 9(5), 700–713 (2016)
9. Lohrmann, B., Janacik, P., Kao, O.: Elastic stream processing with latency guarantees. In: 35th Int. Conf. on Dist. Comp. Systems (ICDCS). pp. 399–410 (2015)
10. McAfee, A., Brynjolfsson, E., Davenport, T.H., Patil, D., Barton, D.: Big data. The management revolution. Harvard Bus Rev 90(10), 61–67 (2012)
11. Schulte, S., Hoenisch, P., Hochreiner, C., Dustdar, S., Klusch, M., Schuller, D.: Towards process support for cloud manufacturing. In: 18th Int. Enterprise Distributed Object Computing Conf. (EDOC). pp. 142–149. IEEE (2014)
12. Shen, Z., Kumaran, V., Franklin, M.J., Krishnamurthy, S., Bhat, A., Kumar, M., Lerche, R., Macpherson, K.: Csa: Streaming engine for internet of things. Data Engineering pp. 39–50 (2015)
13. Shukla, A., Simmhan, Y.: Benchmarking distributed stream processing platforms for iot applications. In: Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things - 8th TPC Technology Conference, TPCTC. pp. NN–NN (2016)
14. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D.: Storm@twitter. In: 2014 ACM SIGMOD Int. Conf. on Management of Data. pp. 147–156 (2014)
15. Usbeck, R., Röder, M., Ngonga Ngomo, A.C., Baron, C., Both, A., Brümmer, M., Ceccarelli, D., Cornolti, M., Cherix, D., Eickmann, B., et al.: GERBIL: General entity annotator benchmarking framework. In: Proc. of the 24th Int. Conf. on World Wide Web. pp. 1133–1143. ACM (2015)
16. Waibel, P., Hochreiner, C., Schulte, S.: Cost-efficient data redundancy in the cloud. In: 9th International Conference on Service-Oriented Computing and Applications (SOCA). pp. 1–9. IEEE (2016)
17. Xu, L., Peng, B., Gupta, I.: Stela: Enabling stream processing systems to scale-in and scale-out on-demand. In: Int. Conf. on Cloud Engineering (IC2E). IEEE (2016)
18. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: Driving directions based on taxi trajectories. ACM SIGSPATIAL GIS 2010 (2010)
19. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud 10, 10–17 (2010)