Proactive Auto-scaling of Resources for Stream Processing Engines in the Cloud

Tarek M. Ahmed Farhana H. Zulkernine James R. Cordy

School of Computing, Queen's University
Kingston, ON, Canada

{tahmed,farhana,cordy}@cs.queensu.ca

ABSTRACT

Large scale applications nowadays continuously generate massive amounts of data at high speed. Stream processing engines (SPEs) such as Apache Storm and Flink are becoming increasingly popular because they provide reliable platforms to process such fast data streams in real time.

Despite previous research in the field of auto-scaling of resources, current SPEs, whether open source such as Apache Storm, or commercial such as streaming components in IBM Infosphere and Microsoft Azure, lack the ability to automatically grow and shrink to meet the needs of streaming data applications. Moreover, previous research on auto-scaling focuses on techniques for scaling resources reactively, which can delay the scaling decision unacceptably for time sensitive stream applications. To the best of our knowledge, there has been no or limited research using machine learning techniques to proactively predict future bottlenecks based on the data flow characteristics of the data stream workload.

In this position paper, we present our vision of a three-stage framework to auto-scale resources for SPEs in the cloud. In the first stage, the workload model is created using data flow characteristics. The second stage uses the output of the workload model to predict future bottlenecks. Finally, the third stage makes the scaling decision for the resources. We begin with a literature review on the auto-scaling of popular SPEs such as Apache Storm.

Keywords

Streaming data, auto-scaling, elasticity, machine-learning

1. INTRODUCTION

Stream Processing Engines (SPEs) are frameworks that can reliably process and query stream data at high volume and high speed. SPEs, such as Apache Storm [3] and Flink [1], are becoming increasingly popular with the emergence of new data sources that can produce massive amounts of data in short periods of time. Examples of such sources include social networks such as Facebook and Twitter, and networks of smart devices in Internet of Things (IoT) [11]. Cisco [5] expects that about 50 billion devices will be connected to the internet by 2020, generating massive amounts of fast streaming data.

Modern organizations require real-time analysis of their high speed streaming data, and use SPEs to provide timely feedback and decision-making. Because of its cost effective pay-as-you-go model, organizations increasingly choose to host their systems including SPEs in the cloud.

To benefit from the pay-as-you-go model, a cloud service should be able to optimize the usage of resources and minimize latency. Major cloud vendors have well established auto-scaling techniques to handle fixed, predictable workloads such as database queries. Streaming data on the other hand is dynamic, unbounded and unpredictable, and traditional auto-scaling techniques are not adequate. New auto-scaling techniques are, therefore, required to analyze the data flow characteristics and use the knowledge hiding within this data to make more reliable and adaptive scaling decisions.

In this position paper we present our vision of a novel framework to auto-scale cloud resources for streaming data in popular SPEs such as Apache Storm. Our vision explores two fundamental ideas. First, we examine the straming data flow characteristics such as speed and acceleration. For example, the speed of a stock market stream is directly affected by the occurrence of a major event such as a natural disaster. By analyzing the data stream and measuring its speed and acceleration, we can expect the upcoming stream volume, and further predict the required cloud resources.

Second, we explore the use of streaming machine learning techniques to classify workloads and predict future bottlenecks. Our motivation is the fact that the data flow characteristics of the streaming data can be thought of as another streaming data source that can be analyzed using an SPE.

While auto-scaling of resources is not a new topic in cloud research, little has been targeted at popular and practical SPEs such as Apache Storm. In this work we specifically aim to address this gap, using the data flow characteristics of streaming data and machine learning techniques to proactively predict scaling of resources.

2. RELATED WORK

Auto-scaling of cloud resources is not a new topic in the literature. Efforts have been made to achieve auto-scaling

© 2016 ACM. ISBN X-XXXXX-XX-X/XX/XX.
DOI: http://dx.doi.org/10.1145/0000000.0000000

and resource provisioning of cloud resources for both streaming and non-streaming data [18, 11, 20].

Generally, auto-scaling techniques can be grouped into two categories: reactive and proactive. Reactive auto-scaling techniques collect metrics about the utilization of a monitored system, then use these metrics to decide whether to grow or shrink the required resources. Such reactive techniques can lead to losing a large amount of data when a sudden bottleneck occurs before the scaling decision can be made.

The real-time nature of streaming data introduces new challenges to the field of auto-scaling, including the need to achieve low latency and guaranteed throughput [11]. Therefore, reactive techniques may not be a suitable choice for scalability decisions. Instead, proactive auto-scaling techniques can be used predict future bottlenecks and make scalability decisions in advance of the actual bottleneck.

The remainder of this section provides a literature review on auto-scaling studies for SPEs, including popular SPEs such as Apache Storm and its successor Heron, Apache Spark and Apache Flink.

In practice there is only a limited body of work targeted at automatically scaling existing popular SPEs. Most of the techniques require human intervention to issue a scaling command [11, 14, 8, 23]. In addition, most existing techniques aimed at auto-scaling of SPEs are reactive, where a monitoring system records and reacts to observed metrics such as CPU utilization, memory consumption and network latency. The monitoring system then makes the scalability decision after the bottleneck actually occurs.

2.1 Conceptual Architecture of an SPE

Fig. 1 shows a high level architecture of a typical SPE deployed in a cluster environment. The figure shows a cluster of n nodes, each node executing an instance of the streaming data pipeline. Typically, a pipeline consists of a directed graph of processing units, also known as operators, that handle the logic of the pipeline. An operator is a single unit of processing that handles the logic of the data streaming system.

There are two types of operators, stateful and stateless. A stateful operator stores the current state of its processing to be used in future processing, whereas a stateless operator does not store any state. For example, an operator that calculates an aggregate function of the data such as the sum or the average of a specific input is typically a stateless operator.

Generally, streaming data passes from the data source, a system that generates the data such as a Twitter feed, to the SPE. The SPE has a predefined node that acts as a coordinator (e.g., the Nimbus node in Apache storm). The objective of the coordinator node is to assign work to a set of worker nodes in the cluster. Each worker node receives assignments from the coordinator node and creates instances of operators needed for each job (nodes 1 to 4 in Fig. 1). Finally, sinks are responsible for handling the output of the pipeline. A sink is the termination point of the pipeline where the output data is consumed such as writing to a database or a file.

2.2 Comparison between Popular SPEs

Auto-scaling of SPEs requires a clear understanding of the specifications of the popular SPEs in order to be able

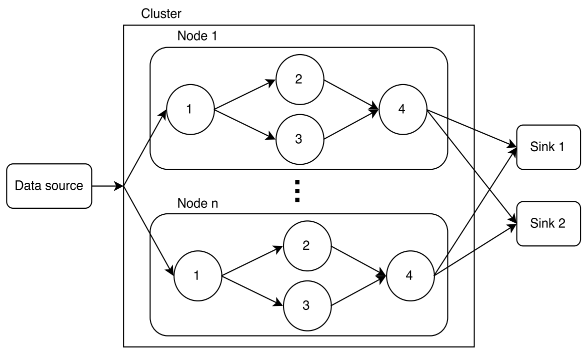

Figure 1: SPE conceptual architecture

to consider which may be more suitable to for auto-scaling. In this subsection we compare the three mainstream open source SPEs, Apache Storm, Spark streaming, and Apache Flink. Our comparison concentrates on the properties relevant to auto-scaling research that can directly affect our choice of SPE. Table 1 summarizes our comparison of the three SPEs that we are considering.

2.2.1 *Processing Technique*

In streaming data research, there are two main techniques for processing data, namely, instance-based and micro-batch. In instance-based processing, each data tuple in the stream is processed once it is received by the SPE individually. Instance-based techniques usually have lower latency than micro-batch techniques because each data tuple is processed as soon as it arrives. On the other hand, Failure handling in instance-based processing can be more complex, as we discuss in subsection 2.2.2 below. Both Apache Storm and Flink use instance-based processing to process their input data stream.

In a micro-batch technique, the data stream is divided into chunks of small batches that are processed at every specific time duration. This time duration is usually a configurable parameter. Spark streaming uses a micro-batch processing technique. The advantage of the micro-batch technique is its relative simplicity, since it can be built on top of traditional data processing engines. However, the disadvantage of micro-batch techniques is the higher latency since the system is required to wait for a specific time duration before processing starts.

2.2.2 *Dealing with Failures*

In large scale systems that process massive amount of data, failures will happen. All SPEs provide techniques to handle such failures seamlessly. Generally, SPEs use two techniques to replay failed data tuples when a certain node fails, namely, acknowledgment (ACK) and checkpoints. Apache Storm uses an ACK mechanism where the source of the data keeps a backup of each tuple until it receives an ACK from each operator in the path of the tuple up to the data sink [26].

By contrast, Apache Flink uses a checkpoint mechanism. Instead of acknowledging each individual tuple, Flink injects a marker in the data stream to indicate that previous tuples were processed successfully. The duration between checkpoints can be configured by the user. In case of a failure, Flink replays all the tuples that occurred after the last

Table 1: Comparison between popular SPEs

	Storm	Spark	Flink
Processing	Instance-based	Micro-batch	Instance-based
Failures	ACKs	Checkpoints	Checkpoints
Guarantees	At least once	Exactly once	All

checkpoint. Flink does not store the backup in its internal state, rather it uses external storage.

2.2.3 *Processing Guarantees*

A typical SPE guarantees that all of the data is processed. Three kinds of guarantee may be offered: at-most-once, at-least-once and exactly-once. The guarantee level is closely related to the failure mechanism used by the SPE. For example, the ACK mechanism used by Storm can lead to duplicate processing of tuples. Consider a tuple that was processed by all operators, however a node failure caused the ACK not to be delivered to the data source. Hence, this data tuple will be processed the second time. Therefore, Storm guarantees at-least-once processing of data tuples.

Spark streaming uses a simpler mechanism to deal with failures. Spark maintains the state of the batches that it processes, therefore, a micro-batch is only replayed whenever a failure happens. Hence, Spark only supports exactly-once processing of tuples.

Finally, Flink can be configured to support any of the three kinds of guarantee, based on the application requirements. Its default checkpoint mechanism allows for an exactly-once guarantee, however, this can be downgraded to at-least-once to reduce latency.

In our work, we aim at investigating an SPE that deals with data in its natural streaming form in order to be able to calculate data flow characteristics. Therefore, we will focus our work on either Apache Storm or Flink. Apache Flink uses more advanced features than the other SPEs for dealing with failures to provide better data processing guarantees. These features are needed in real life deployments. Thus, for our auto-scaling research, we will begin with the simpler approach of Apache Storm, and later extend our work to Apache Flink.

2.3 Auto-scaling in Popular SPEs

Despite the large body of research in the field of auto-scaling of resources, current popular SPEs lack the ability to automatically grow and shrink to meet the needs of streaming data applications.

2.3.1 *Apache Storm and Heron*

Apache Storm does not provide an auto-scaling feature either on the level of individual topology operators or on the level of nodes [6]. Rather, the administrator is required to issue a special command *"storm rebalance"* to change the number of nodes or workers. Storm then deactivates all the working nodes and reactivates them in the new arrangement [24].

Several research papers use Apache Storm in their experiments. Some of these papers use the famous Monitor, Analyze, Plan and Execute (MAPE) cycle to make the scalability decision. [23, 22].

Xu. et *al*. propose Stela [24], a technique to scale Apache storm seamlessly. The goal of this technique is to maximize the number of processed events and to minimize the duration of interruption during the scaling operation. However, this technique is invoked on-demand, where an external event must trigger the scaling operation.

In recent work, Cooper [12] proposes a proactive framework that uses a time series analysis to predict SLA breaches for streaming data in cloud environments. Despite the large body of research in the field of auto-scaling of resources for steaming data, we believe that Cooper's work and our work are the first to propose proactive resource provisioning for streaming data in popular SPEs such as Apache Storm and Apache Flink.

Having been only recently released (mid 2015) [15], Apache Heron has yet to receive attention in auto-scaling research.

2.3.2 *Apache Flink*

Although Apache Flink is reported to have significantly higher throughput than Apache Storm [13], we were not able to find any work that uses Apache Flink in auto-scaling research. Some papers that use Apache Storm claim that the proposed techniques can be easily extended to Apache Flink or other streaming data frameworks.

By contrast with Apache Storm, Flink has a simpler data structure called the Dataset to define the processing pipeline. This API hides most of the complexities of Storm to define separate operators. However, this simplified API makes it more complex for researchers to understand the underlying scalability mechanisms. Therefore, researchers prefer Apache Storm as a platform of choice for performing their experiments.

2.3.3 *Apache Spark Streaming*

Spark has auto-scaling features for its batch mode processing, where it is able to scale up and down based on the workload [21]. However, this feature is missing when dealing with streaming data. Several research papers have addressed the scalability of Spark as a map-reduce framework, however, auto-scaling for streaming data in Spark has yet to receive any attention in the literature.

2.4 Auto-scaling in Other Frameworks

In addition to these popular SPEs, auto-scaling for other SPEs in the cloud have been proposed in the literature. Esper [4] is a popular choice in several research papers. Zacheilas et *al*. propose a mechanism to predict the latency and the load of Esper engine [25]. In contrast to popular SPEs that are distributed in nature, Esper runs on a single node and it has to be combined with the popular SPEs to achieve the high availability. Our work focuses on resource provisioning of the popular distributed SPEs such as Apache Storm and Apache Flink.

Stormy [17] is a scalable streaming service for the cloud. To achieve scalability and elasticity, Stormy has a mechanism to distribute queries from the overloaded nodes. However, authors do not specify the criteria for selecting the overloaded nodes.

StreamMine3g [19] promises flexible topologies that can change at runtime in addition to the application of linear horizontal scaling when nodes are overloaded. Experiments in this work use a threshold of 80% CPU utilization to make reactive scalability decisions.

Abrantes et *al*. [8] present a different approach for auto-

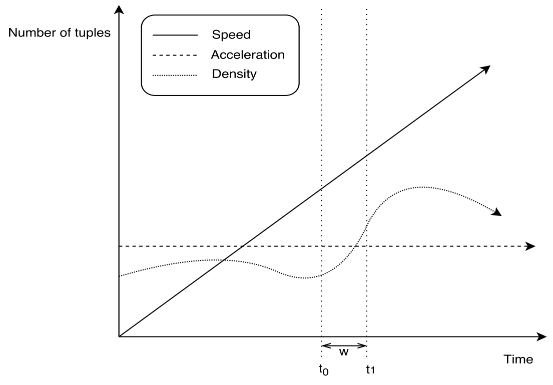

Figure 2: Streaming data flow characteristics

scaling, using the actual content of a Twitter feed to predict future workload. The authors use sentiment analysis of soccer-related tweets to detect rapid changes in the crowd's behavior and to predict bursts of messages before they actually occur. They find that in several cases, their algorithm can help prevent SLA violations compared to other threshold-based methods.

Major vendors such as IBM, Amazon and Google have their own streaming data solutions, however, they require manual intervention to adjust the required resources based on the workload. Microsoft [16] proposes a new SPE, StreamScope, a distributed SPE running on a cluster of 20,000 servers. However, the framework is not open source and the paper does not describe any auto-scaling strategy.

3. OUR APPROACH: SMART PROACTIVE AUTO-SCALING

In this section we propose a three-stage framework to provide proactive auto-scaling of cloud resources for streaming data. The novelty of our proposal lies in:

1. The use of the data flow characteristics of streaming data to construct machine learning models of varying workloads.
2. The definition of performance models for different classes of workloads and resources.
3. Application of a decision model to first, decide whether to grow or shrink the resources and second, to decide the amount of needed resources to scale.

3.1 Key Challenges

In this section we describe the two fundamental challenges that we address in our work.

Characterizing Streaming Data

Streaming data has a fundamentally different nature than static data. Querying static data has been well-established, and effective query optimization techniques exist for such systems. On the other hand, querying dynamic and unbounded streaming data requires different mechanisms. The main difference is that static data residing on a storage medium can be measured in terms of size and latency of retrieval, whereas such measurements are not applicable to dynamic streaming data.

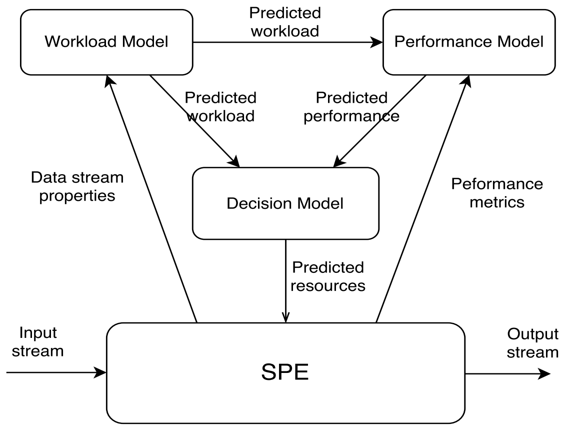

Figure 3: Proposed framework

Streaming data cannot be measured using traditional techniques. Fig. 2 shows the high level conceptual relation between time and the number of streaming data tuples. For simplicity, the figure shows a data stream with a linear increase in its velocity over time. The data stream is divided into windows for processing purposes. For each window, the speed, acceleration and density of the stream can be calculated. In the figure, the acceleration is a non-zero value since the speed is increasing linearly. The density however has an arbitrary value to demonstrate the idea. The density can be calculated as the average size of the tuple in the window w.

Machine Learning Techniques

Since streaming data flow characteristics are themselves streaming data, traditional machine learning techniques can not be applied. A great deal of research has been done in the field of machine learning for streaming data by Bifet et *al.* in several research papers [10], including the distributed scalable machine learning framework Apache SAMOA [2] (Scalable Advanced Massive Online Analysis).

In our work, we plan to explore the regression techniques proposed in literature for streaming data and the available techniques in current frameworks such as Apache SAMOA and Spark MLlib.

We will explore the applicability of workload classification techniques using streaming machine learning techniques. The purpose of workload classification is to classify workload characteristics into groups such as low, medium or high load. If we can train a classifier based on observed data flow characteristics to predict the onset of a change in load, we may be able to proactively reallocate the required resources.

3.2 Proposed Framework

Fig. 3 shows the high level architecture of our three-stage framework for proactive auto-scaling of cloud resources for SPEs. The framework has three main components: (1) the Workload model, (2) the Performance model, and (3) the Decision model.

Workload Model

A monitoring component will be used to record data flow characteristics. Simple characteristics can be used, such as:

1. Speed: The average arrival rate of tuples to the SPE.

2. Acceleration: The average rate of increase/decrease of speed.

3. Density: The average size of a tuple.

More complex characteristics can also be investigated, such as:

1. Variability of the speed: A property to measure how frequently the speed changes. This can be calculated using the standard deviation of the speed for a specified number of windows. For example, a higher variability means that the cloud resources should be scaled to withstand a higher load until the load stabilizes to prevent rapid fluctuations in system performance.

2. Operator throughput: A property to measure the ratio of the number of tuples leaving to the number of tuples arriving at the operator in a specific time window. An ideal operator that performs modifications to the input without performing any aggregation should have a throughput of one. Lower throughput might indicate a bottleneck in the operator, which would require expanding the resources assigned to it.

3. Average processing time: The average time required by an operator to process a tuple. Higher processing time than the average indicates a bottleneck in the operator, which would require more resources.

Workload characteristics collected during system's operation will be the independent variables in our prediction model. We will explore whether a single model is enough, or whether an ensemble of models should be created to predict future workloads

In order to construct the machine learning model, we will explore the streaming data regression algorithm provided in SAMOA and the Adaptive Model Rules Regressor algorithm (AMRules for short) originally proposed by Almeida et *al*. [9].

Performance Model

The purpose of the performance model is to use traditional performance measures such as CPU utilization, memory consumption and network latency to predict the future bottlenecks for a specific *workload model*. We will explore how to minimize a specific metric such as the processing latency or number of SLA breaches.

Performance models have been studied extensively in literature for static data, for example in the work done by Mian et *al*. [20]. We will explore the applicability of these models to our streaming data.

Decision Model

The final step in the framework will use both the performance and workload models to make scalability decisions such as whether to grow or shrink the allocated resources, which resources and by what amount .

To begin with, the decision model will be a simple rule-based model. In later phases of our work, we will explore using a more sophisticated prediction model in this stage rather than a simple rule-based model.

3.3 Datasets

In order to evaluate our framework, we plan to use datasets that naturally provide typical fluctuations in the speed and volume of the data stream for extended periods of time. We have explored the existing benchmarks and datasets used in the literature and unfortunately, very little work has been done to standardize these datasets. Furthermore, some research papers have used synthesized datasets [23, 22], which may not be suitable for research on auto-scaling of streaming data, because they may not reflect the real nature of streaming data and thus will impose several threats on the validity of our results.

We evaluated several benchmarks and datasets proposed in the literature to decide which are suitable for the needs of our research. We found one benchmark for Apache Storm which is provided by Yahoo [7]. The stream in this benchmark is composed of 1000 tuples for 10 seconds. We excluded this benchmark since the workload does not naturally vary with time and hence does not meet our needs.

We are considering a few other candidate stream datasets for our work as the following:

1. Frankfurt Stock Exchange: This dataset is used by Heinze et *al*. [14]. Naturally stock exchange data are affected by external factors such as political events or natural disasters. Therefore, we are considering to use it in our research.

2. Smart City Sensor Networks: This dataset will be used by Cooper [12]. This dataset is of varying characteristics that will be suitable for our research.

3. Twitter data: In a recent paper, Abrantes et *al*. [8] use tweets for a crowd of a football tournament to predict SLA breaches. This data is also of varying nature, hence we will investigate to use it in our research.

Similar to the state-of-the-art research in the field of resource provisioning for SPEs, we will measure the effectiveness of our proposed framework by recording metrics such as processing latency and percentage of failed tuples. These metrics will then be compared against the results of other techniques such as the traditional threshold approach.

4. OPEN ISSUES

This paper proposes a high level conceptual framework for auto-scaling of streaming data in SPEs. We are aware of a number of challenges that we will be investigating in our work.

First, a distinction has to be made between stateful and stateless operators. In a local cluster, the latency of copying an operator state is negligible because of high speed local connections between individual nodes. In a cloud environment however, where larger distances and higher latencies between nodes exist, moving state is challenging. Initially we will investigate stateless operators, and then attempt to extend our work to stateful operators.

Second, earlier studies often neglect the time delay needed to create new resources such as virtual machines. We will begin with the same assumption for our initial studies, and then we will add an extra time penalty factor to our performance model to consider this delay.

5. CONCLUSIONS

In this work, we aim at investigating the data flow characteristics of streaming data workloads, and the problem of how to automate the scaling of resources in the presence of these uncertainties. Creating performance and utilization models using these characteristics (i.e. meta-information) can be useful in predicting future bottlenecks in cloud environments, and hence in enabling more reliable and accurate auto-scaling mechanisms for stream processing applications.

Our work will focus on machine learning based on streaming data, and in particular meta-information such as the speed, acceleration and density of stream data sources. We plan to leverage the use of existing streaming data regression models to act on the meta-information of streaming data, in order to predict the most efficient way to shrink or grow the required resources in the presence of a rapidly varying streaming data workload.

6. REFERENCES

[1] Apache Flink (last accessed: 26/6/2016). http://flink.apache.org.

[2] Apache SAMOA (last accessed: 26/6/2016). https://samoa.incubator.apache.org/.

[3] Apache Storm (last accessed: 26/6/2016). http://storm.apache.org.

[4] Esper (last accessed: 22/8/2016). http://www.espertech.com/esper/.

[5] The internet of things in motion (last accessed: 6/6/2016). https://newsroom.cisco.com/articleId=1208342.

[6] STORM-594: Auto-scaling resources in a topology (last accessed: 26/6/2016). https://issues.apache.org/jira/browse/STORM-594.

[7] Yahoo streaming benchmark. https://github.com/yahoo/streaming-benchmarks.

[8] A. Abrantes, D. P. Souza, and M. A. S. Netto. Using application data for SLA-aware auto-scaling in cloud environments. In *Proc. IEEE 23rd Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Atlanta, GA, USA, 2015.

[9] E. Almeida, C. Ferreira, and J. Gama. Adaptive model rules from data streams. In *Proc. Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*, pages 480–492, Prague, Czech Republic, 2013.

[10] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Intl. Symposium on Intelligent Data Analysis*, pages 249–260, Lyon, France, 2009.

[11] J. Cervino, E. Kalyvianaki, J. Salvachua, and P. Pietzuch. Adaptive provisioning of stream processing systems in the cloud. In *Proc. IEEE 28th Intl. Conf. on Data Engineering Workshops*, pages 295–301, Arlington, VA, USA, 2012.

[12] T. Cooper. Proactive scaling of distributed stream processing work flows using workload modelling. In *Proc. 10th ACM Intl. Conf. on Distributed and Event-based Systems*, pages 410–413, Irvine, California, 2016.

[13] J. Grier. Extending the Yahoo! streaming benchmark (last accessed: 26/6/2016). http://data-artisans.com/extending-the-yahoo-streaming-benchmark/, 2016.

[14] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *Proc. 8th ACM Intl. Conf. on Distributed Event-Based Systems*, pages 318–321, Mumbai, India, 2014.

[15] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *Proc. Intl. Conf. on Management of Data*, pages 239–250, Melbourne, Victoria, Australia, 2015.

[16] W. Lin, H. Fan, Z. Qian, J. Xu, S. Yang, J. Zhou, and L. Zhou. Streamscope: Continuous reliable distributed processing of big data streams. In *Proc. 13th Usenix Conf. on Networked Systems Design and Implementation*, pages 439–453, Santa Clara, CA, 2016.

[17] S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann. Stormy: An elastic and highly available streaming service in the cloud. In *Proc. Joint EDBT/ICDT Workshops*, pages 55–60, Berlin, Germany, 2012.

[18] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proc. Intl. Conf. on High Performance Computing, Networking, Storage and Analysis*, pages 49:1–49:12, Seattle, Washington, 2011.

[19] A. Martin, A. Brito, and C. Fetzer. Scalable and elastic realtime click stream analysis using streammine3g. In *Proc. 8th ACM Intl. Conf. on Distributed Event-Based Systems*, pages 198–205, Mumbai, India, 2014.

[20] R. Mian, P. Martin, F. Zulkernine, and J. L. Vazquez-Poletti. Cost-effective resource configurations for multi-tenant database systems in public clouds. *Intl. Journal of Cloud Applications and Computing*, 5(2):1–22, 2015.

[21] M. L. Michael Le. Towards the true elasticity of spark (last accessed: 6/6/2016). https://spark-summit.org/2015/events/towards-the-true-elasticity-of-spark/.

[22] M. Nardelli. A framework for data stream applications in a distributed cloud. In *Proc. of the 8th ZEUS Workshop*, pages 56–63, Vienna, Austria, 2016.

[23] J. S. v. d. Veen, B. v. d. Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer. Dynamically scaling apache storm for the analysis of streaming data. In *Proc. IEEE 1st Intl. Conf. on Big Data Computing Service and Applications*, pages 154–161, San Francisco Bay, USA, 2015.

[24] L. Xu, B. Peng, and I. Gupta. Stela: Enabling stream processing systems to scale-in and scale-out on-demand. In *Proc. IEEE Intl. Conf. on Cloud Engineering*, pages 22–31, Berlin, Germany, 2016.

[25] N. Zacheilas, V. Kalogeraki, N. Zygouras, N. Panagiotou, and D. Gunopulos. Elastic complex event processing exploiting prediction. In *IEEE International Conference on Big Data*, pages 213–222, Oct 2015.

[26] P. Zapletal. Comparison of Apache stream processing frameworks. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, 2016.