Measuring stream processing systems adaptability under dynamic workloads

Nicolas Hidalgo[a], Erika Rosas[b], Cristobal Vasquez[c], Daniel Wladdimiro[c]

[a] *Universidad Diego Portales*
Facultad de Ingeniería - Escuela de Informática y Telecomunicaciones
[b] *Universidad Técnica Federico Santa María*
Departamento de Informática
[c] *Universidad de Santiago de Chile*
Departamento de Informática

Abstract

Data streaming belongs to the Big Data ecosystem, which generates high-frequency data streams featuring time-varying characteristics that challenge the traditional stream processing systems capacities. To deal with this problem, many self-adaptive stream processing systems have been proposed. Despite the evolution of self-adaptive systems, there is still a lack of standardized benchmarking systems to enable scientists to evaluate the autonomic capacities of their solutions. In this work, we propose an index called AI-SPS inspired by the human cerebral auto-regulation process. The index quantifies the capacity of an adaptive stream processing systems to self-adapt in the presence of highly dynamic workloads. An index of this nature will help the scientific community generate fair comparisons among literature with the aim of creating better solutions. We validate our proposal by evaluating the adaptive behavior of two state of the art self-adaptive stream processing systems. Tests were performed using real traffic datasets adapted specifically to stress the processing system. Results show that the proposed index quantifies the adaptation capacity of self-adaptive stream processing systems effectively.

Keywords: Adaptation index, Benchmarks, Autonomic systems, Stream processing

Email address: nicolas.hidalgoc@mail.udp.cl (Nicolas Hidalgo)

1. Introduction

Nowadays, most applications and systems focus their operation on online interactions. Every day, billions of these interactions take place around the world, containing information about news, social media, sensors, and trading operations, among others. Every interaction contains valuable information for users, company shares, government authorities, and the like; however, due to their online nature, it is difficult to extract that information in an efficient manner by applying traditional processing tools [1]. The latter issue, and the increasing need for timely responses have pushed the traditional processing paradigm, such as batch processing, towards an online processing manner called data stream processing.

Recently, many Stream Processing Systems (SPSs) have been released based on this processing paradigm, such as Storm[1], S4[2], Samza[3], and Spark[4]. SPSs extract relevant information from data *on-the-fly* without requiring storage, following a graph-based model where each vertex represents a task or operation to be applied to the incoming data, and edges represent the incoming and outgoing data flows. Examples of traditional operations are filtering, classification, ordering, counting, and merge, among others. The data sources may provide any type of event, like social interactions over a given social network, sensor measurements from a smart city application, log entries from a network monitor, and the like [2].

The online nature of these systems imposes challenges related to the dynamic behavior of the data flows and their distribution. In highly dynamic scenarios, an operator may overload a physical machine increasing latency, and in some cases, compromising the precision of the results. In order to solve this problem,

[1] http://storm.apache.org
[2] http://incubator.apache.org/s4/
[3] http://samza.apache.org
[4] http://spark.apache.org

efforts have been made to provide the self-adaptation of the processing system. In this work, we consider a self-adaptive system as a processing system with an autonomic behavior to organize its internal logic. A self-adaptive system may apply one or more mechanisms or strategies to adapt to changing execution conditions. Moreover, it should be able to detect, diagnose, and repair localized problems, and at same time, continually seek ways to improve its operation, identifying and seizing opportunities to make itself more efficient in performance or cost.

Most of these endeavors apply the operator fission technique, which allows the system to make replicas of a given operator to parallelize its tasks attempting to reduce the load of the most critical ones [3] [4] [5] [6] [7]. In Section 2, we present and discuss other techniques that may also be applied in this context.

Although recently many solutions that adapt the processing logic at run-time have been proposed, there is still a lack of common metrics to compare and evaluate the performance of the different self-adaptive strategies proposed in the literature. In this article, we present the design and evaluation of an Adaptability Index for Stream Processing Systems (AI-SPS), a metric that allows for the comparison of the adaptation capacity of the different Stream Processing Systems (SPS). AI-SPS is inspired by the model-free Autoregulation Index (mf-ARI) proposed by Chacon et al. [8], which was conceived to measure the efficiency of dynamic cerebral autoregulation on human patients analyzing arterial blood pressure changes. We draw an analogy between the brain and the SPS, finding similarities between the cerebral autoregulation that the brain faces when arterial blood pressure changes with the adaptation of the SPS when the flow of events increases or decreases.

The remaining sections of this article are organized as follows: Section 2 presents the background and the related literature associated with self-adaptive SPS, traditional metrics to compare SPS, and autoregulation indexes. Section 3 presents the design of the AI-SPS. Section 4 presents the evaluation and the analysis performed to validate our proposal by using different sources of events and applications. Finally, concluding remarks and future work are included in

the last section.

2. Background and Related Work

2.1. Autoregulation Index

The human body is a highly autonomous system that is able to adapt to changing conditions. One of the central organs of the human body is the brain. The brain needs to maintain a relatively constant flow of blood in order to be functional; this process is known as Cerebral Autoregulation (CA). CA aims to maintain stable cerebral blood flow even under changes in blood pressure. Problems with CA are usually associated with brain pathologies. For this reason, the study of CA is a subject undergoing intense study in the medical community. In order to evaluate CA, several methods have been proposed to characterize the efficiency of the dynamic cerebral flow autoregulation response. For example, Tiecks et al. [9] defined a single second-order model to describe the relationships between Cerebral Blood Flow Velocity (CBFV) and Arterial Blood Pressure (ABP). The model defines a set of templates that represent the behavior of CA. Each template is associated with a discrete scale value that ranges from 0 to 9, where 0 reflects the absence of CA, and 9 the best CA possible. This model is one of the most used and has even been included in medical equipment. In 2014, Chacon et al. [8] proposed mf-ARI, which is a model-free version of Tiecks et al.'s autoregulation index (ARI). The mf-ARI uses a linear regression model built from the curves of the Aaslid-Tieck model. This new approach has proven to be more accurate in identifying CA in comparison to the traditional ARI [8].

In general, mf-ARI evaluates the cerebral response to blood pressure changes by observing 3 parameters: (1) the ability of the system to return to an initial state or close to it (K_s), (2) the time that it takes the system to reach the stable or initial state after it reaches the minimum signal value ($\Delta\tau$), and (3) the angle (in degrees) between the lines that represent the CBFV transient response and the ABP recovery signal corresponds to the (ϕ). Figure 1 presents

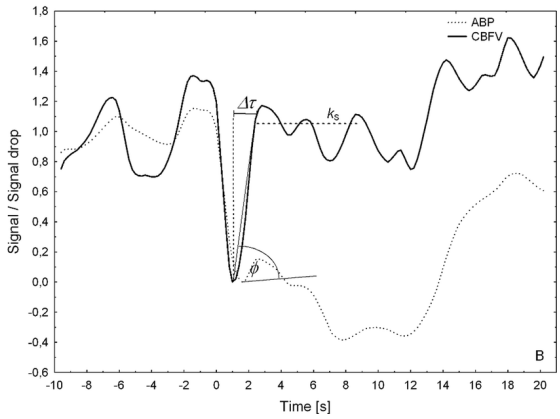


Figure 1: mf-ARI parameters in typical ABP and CBFV signals [8].

a representation of the parameters used to estimate the index. We observe the behavior of both ABP and CBFV when ABV drops due to the application of a maneuver (i.e. thigh-cuff or manguitos). The maneuver's goal is to diminish the ABP, so that it then increases suddenly. Such an abrupt change enables the quantification of CA. ARI and mf-ARI are two indexes that can evaluate CA on a common scale. They quantify the ability of the human brain to adapt to changing conditions. In this work, we provide an index to quantify the ability of adaptive stream processing systems to cope with the traffic dynamism. Inspired by mf-ARI, we propose an index called AI-SPS that is able to characterize the adaptation capacity of self-adaptive SPSs. In Section 3, we provide the design of the proposed metric.

2.2. Adaptive SPSs

Stream processing systems (SPSs) have to deal with highly dynamic traffic behavior, which makes it necessary to adapt the SPS at runtime. However, this characteristic is not always supported by the SPSs. Systems such as S4 [10], Storm [11], and Samza [12] use a graph-based model (DAG), where vertices represent processing operations over the stream of data, and edges represent the flow of data between these operations. In these systems, this graph cannot be modified at run-time, hence limiting the system processing capacity to the

scenario for which they were configured.

If the stream of data maintains its rate over time, the number of resources required to process the events can be configured beforehand. However, the flow of events of several applications changes over time. In the context of Web 2.0, for example, users generate content in unpredictable ways with spikes of traffic generally triggered by real-work events [13]. In this context, the SPSs may be over-provisioned, but resources are being wasted most of the time; they may also offload events (discard events when the system is overloaded), but this may reduce the accuracy of the processing results.

Self-adaptive SPSs are able to change their internal framework/applications and adapt them to the changing conditions and workloads. A traditional technique to self-adapt relies on changing the number of replicas or resources available to deploy a processing element in order to fit the dynamic conditions. Broad categories for self-adaptive SPSs that change the amount of replicas are reactive and proactive approaches. The reactive approach is based on the current state of the system, while the proactive approach tries to detect seasonal data in the past history of the system to anticipate system requirements.

Hidalgo et al. [7] propose a self-adaptive processing system that automatically increases or decreases the number of processing elements. This solution uses a reactive model that increments the number of replicas for a processing element when its utilization goes over a threshold and a predictive model that uses statistical information in Markov Chains to predict future traffic behavior.

In [14], authors present Stela, an elastic system able to support active workload scaling based on a metric defined as Effective Throughput Percentage (ETP). Stela is built on top of Apache Storm and pursues two goals: (1) to optimize throughput and (2) to minimize computation interruptions. Similar to the work of Hidalgo et al., Stela relies on the fission technique where the number of operators is increased when a bottleneck is detected. However, Stela selects the congested operator that optimizes the entire system throughput. To this end, ETP is calculated on each congested operator and they are ranked based on this metric.

In [15], authors propose an elastic SPS that is able to adjust the resources in the presence of dynamic workload. The adaptation is carried out using a strategy based on a Predictive Control Model. This model finds the optimal graph (the application) configuration over a time-window by solving an online optimization problem. The control mechanism is designed based on latency constraints, queuing theory models, and energy consumption models. FUGU is another elastic system proposed by Heinze et at. [16]. The system optimizes the selected scaling policy automatically using an online parameter optimization approach. The decisions are made according to local and global threshold-based rules, which deal with operators' performance metrics (CPU, memory, and network consumption). A vector that represents the operator utilization is used as input to the scaling algorithm. The scaling algorithm derives decisions that mark a host as overloaded or the system as underloaded. The operator selection algorithm decides which operators to move and an operator placement algorithm determines where to move these operators. Similar models are proposed in [6], [5], and [3].

StreamCloud [17], on the other hand, splits logical data into multiple physical data sub-streams that flow in parallel, preventing single bottlenecks. A reactive approach based on throughput and congestion is presented in [18]. Congestion is observed when there is a delay sending tuples and the number of tuples processed per second drops over the last adaptation period. Other models are focused on Quality-of-Service metrics to select the optimal configuration of the topology; this is the case of the work presented in [19].

In Esc [20], the authors propose to dynamically attach and release machines to adjust computation capacities to the current nodes. Similarly, to achieve resource elasticity in Storm, the authors in [21] present a model that monitors the Storm platform and external systems, such as queues and databases. Their approach provides an initial guess of the size of the platform and then decides whether to add or remove virtual machines using a control loop. The proposed index may also be applied to this type of adaptability of resources. Most works use threshold-based approaches to determine when to scale, for example, Enorm

[22] and [23]. The latter work also uses a reinforcement learning approach.

More recently, Cardellini et al. propose a hierarchical and distributed self-adaptive architecture for data stream processing [24]. This solution follows a two-level approach, where at the lower level, distributed components control adaptation of the operators, and at the higher level, a per-application centralized component oversees the performance and coordinates reconfiguration actions. This system was implemented on top of Apache Storm and follows a decentralized MAPE control pattern.

In our evaluation, we use both [24] and [7] as case studies to evaluate the proposed index. Both systems provide self-adaptation based on the MAPE control pattern and they apply two techniques to adapt the system to the workflow.

2.3. Comparing adaptability in Adaptive SPSs

In the literature, it is common to measure the quality of an adaptive system by using the number of events that are lost under changing traffic conditions, the number of resource consumption, the end-to-end delay, or by comparing these numbers to an optimum value extracted from an oracle that knows the traffic in advance [16] [7] [25]. Works such as [6] and [17] use throughput and CPU costs to evaluate their systems. The authors in [5] and [19] only use throughput. Finally, the authors in [14], [15], [16], and [20] include time in terms of the latency ratio and response time of messages.

Even though these metrics are correct and widely used, real-time systems must provide accurate and fast results in order to cope with the applications' need. Therefore, if a metric is isolated, it may show better results, but at the risk of compromising another property in the system. For example, if we consider throughput as the common metric to compare two systems, both systems could present similar results in terms of throughput, but one of them could have discarded more events than the other in order to keep data flowing, in consequence, the precision of the results is compromised. The same scenario may occur when latency is considered. For this reason, some of the previously mentioned works have considered composite metrics. Another scenario arises when

comparing different self-adaptive systems in terms of adaptability; however, to the best of our knowledge there is no mechanism or metric to perform this task effectively. In this article, we attempt to provide an index able to quantify the capacity of a self-adaptive SPS to adapt its framework/applications to traffic spikes. Our index is inspired by the model-free Autoregulation Index (mf-ARI) and attempts to integrate the most relevant literature metrics used to evaluate these systems.

2.4. Benchmarking SPSs

In regard to benchmarking SPSs, in the literature we can find many approaches that attempt to evaluate the performance of SPSs; however, none of them have focused on measuring the capacity to adapt to dynamic traffic conditions as our proposal does. The only exception was the work proposed by [26] in which the authors evaluate the capacity for three commercial SPSs to adapt to traffic. They propose a set of micro-benchmarks where throughput and latency are measured to quantify the adaptation capacity.

One of the most well-known and cited benchmark for SPSs is the Linear Road Benchmark proposed by Arasu et al. [27]. The benchmark consists of a toolkit for synthetic data generation, a data sender, and a data validator. The benchmark simulates a real tolling system for a metropolitan area covering multiple expressways. Linear Road defines one overall metric called the L-Rating. The L-Rating indicates how many expressways a system can handle without violating the defined maximum response times for each query. This metric relies on measuring latency and throughput in the system.

StreamBench[28] proposes a benchmark composed of seven type of queries oriented to process textual and numerical data. Moreover, four out of seven queries contain a single computation step, such as extracting a certain field out of a data record, and three queries comprise multiple computation steps. It combines stateless and stateful operations and proposes four real workload suites. Similar to Linear Road, it measures throughput and latency. It can be categorized as a micro-benchmark, because it measures atomic operations, such

as the execution of a projection rather than those of more complex applications such as in Linear Road.

RIoTBench is a benchmark proposed by Shukla et al. [29]. It defines a total of 27 micro-benchmarks scenarios and application benchmarks which are built by combining micro-benchmarks. These micro-benchmarks represent common Internet-of-Things (IoT) tasks such as data pre-processing, statistical summarization and predictive analytics. Similar to StreamBench, RIoTBench relies on real stream workloads sourced from real IoT observations on smart cities and fitness, with peak streams rates that range from 500 to 10000 messages/sec and diverse frequency distributions. The benchmark metrics concern throughput, latency, and jitter.

Other works create their own evaluation scenarios and valuate different SPSs' performance. In articles [30], [31], and [32], the authors focus their performance analysis on three well-known processing platforms: Storm, Spark, and Flink. In regard to the first, they measured performance in terms of throughput and the resilience to node failures over a threats detection on a network traffic scenario. The dataset and the application used in the experiments is a traffic's threat detection analysis created by the same authors. In regard to the latter, the authors employed scenarios based on real-life industrial use-cases. They measured performance in terms of latency and throughput. They conclude that both, Storm and Flick, are true stream processing systems because they provide lower processing latencies. On the other hand, Spark provides higher throughput while having higher latencies. In general, comparing their performance over all the scenarios, there is no single winner, but rather, each system excels in individual use-cases.

In [33], the authors provide an overview of performance benchmarks that can be used for analyzing data stream processing applications. They describe the shortcomings of these benchmarks and state the need of new benchmarks in this area.

3. Adaptation Index for Stream Processing Systems

In this work, we propose an index to quantify the ability of the different self-adaptive stream processing systems to cope with traffic and data distribution dynamics. The index is inspired by the mf-ARI autoregulation index.

3.1. Model

In our model, we assume that Arterial Blood Pressure (ABP) is similar to the data input rate observed in a stream processing. The input rate can vary dynamically, similar to ABP, inducing changes in the system in order to maintain its functions. For example, an input rate increment may require a higher amount of processing resources to maintain the system's throughput, which is similar to the case of CBFV in the brain. On the other hand, a data input reduction may require a reduction in the processing resources to maintain the expected throughput and efficiently use the processing resources. Our analogy considers that changes in the input workload (such as unexpected events) are similar to the maneuvers performed to evaluate the arterial blood pressure change, then, variations in the workload allow the system to show the behavior we would like to evaluate: *adaptability*.

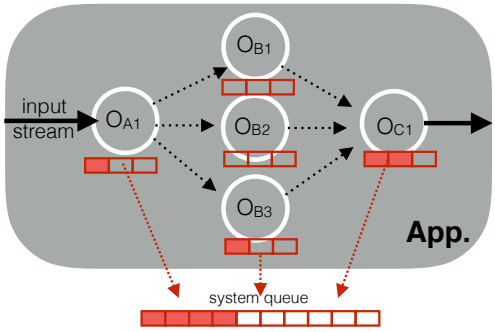


Figure 2: Event queues in SPS

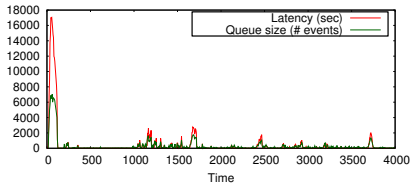
The CBFV is directly related to the ability of the processing system to process the arriving events *as-fast-as-possible*. When the arriving events cannot be processed immediately, they must wait until the release of the processing

resources. To this end, most SPSs present queues of events that allow them to deal with such scenarios and reduce message loss (Figure 2). It is important to notice that physical processing resources may be heterogeneous. Despite the fact that these systems are supported by homogeneous machines clusters, most of the time, this is not mandatory in SPS. On the other hand, processing operators may also differ in complexity; therefore, queuing systems are frequent in most SPS implementations. This fact enables us to apply our index model to most adaptive SPSs.

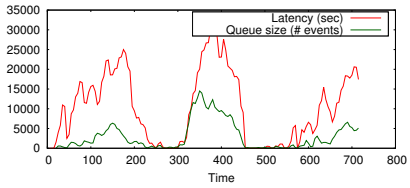
In a traditional SPS, when the input rate suddenly increases, the number of queued events may increase, compromising the end-to-end latency or even the precision of the results if some events are discarded. Queued events are directly correlated with latency, as is presented in Table 1 and Figure 3. In Table 1, we calculate both the Pearson and Spearman coefficient to confirm our claim. We have evaluated the relationship of both variables in 4 different traffic scenarios. Our tests were performed over the self-adaptive stream processing system proposed by Hidalgo et al. [7].

- Scenario 1: consists of 15 million accelerometer registers collected from 31 smartphones.
- Scenario 2: consists of 47 million quotes extracted from news and blog articles. Data was collected over the final three months of the 2008 U.S election.
- Scenario 3: consists of 30 million tweets collected during the FIFA World Cup in Brazil in 2014.
- Scenario 4: consists of 15 million HTTP requests collected at the University of Indiana.

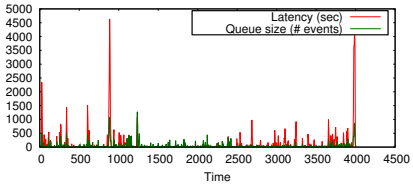
All these scenarios present a relatively constant behavior, with only a few traffic spikes. In order to recreate sudden traffic increments, we synthetically modified the datasets in order to increase the events' spikes. We followed a similar methodology as proposed by Bodík et al. [13].



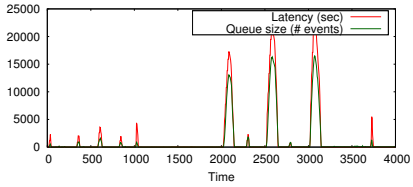
(a) Scenario 1: Mobile



(b) Scenario 2: Quotes



(c) Scenario 3: Tweets



(d) Scenario 4: Web Clicks

Figure 3: Latency and number of queued events correlation under 4 different scenarios running on top of the SPS presented in [7].

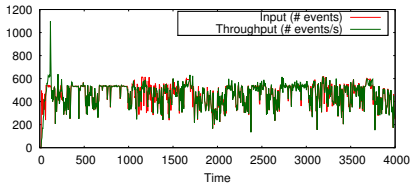
In all the scenarios studied, we observed a clear correlation between the variables' latency and queue size. The relevance of latency depends on the application scenarios. For example, in the context of online data analytics for supporting decision making under critical situations, latency is crucial. The same occurs for any real-time application in the context of IoT or smartcities.

Table 1: Pearson and Spearman coefficient between queue size and latency in 4 different processing scenarios.

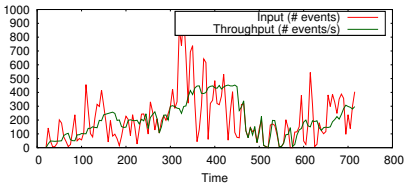
	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
Peason	0.9815	0.7826	0.6574	0.9932
Spearman	0.800	0.8797	0.4280	0.4350

Throughput is another relevant metric to analyze. First, we analyze the behavior of throughput and input rate over the system. In Figure 4 we observe throughput behavior under a sudden increase of input traffic. We can notice that both variables present a high correlation for all the presented scenarios. Basically, if the system is able to adapt to the input traffic, then throughput

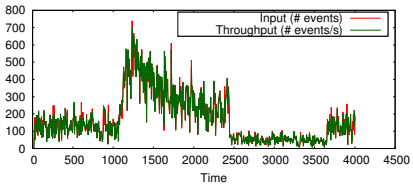
should present a high correlation with the input traffic. A similar result applies for over provisioned systems.



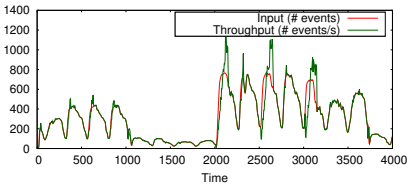
(a) Scenario 1: Mobile



(b) Scenario 2: Quotes



(c) Scenario 3: Tweets

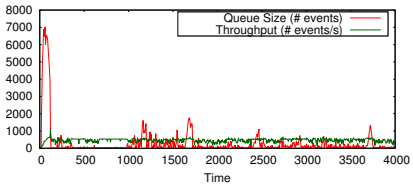


(d) Scenario 4: Web Clicks

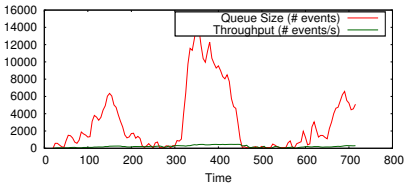
Figure 4: Examples of throughput and input traffic behavior in 4 different scenarios with an adaptive topology.

Another interesting analysis relies on the study of the correlation between the throughput and the queue size. In this case, we claim that these metrics are not strictly related. In fact, it is expected that throughput remains stable when the processing system is unable to adapt to the input traffic and the number of queued events should increase. Figure 5 presents the results of our analysis. These results confirm our claim; a simple correlation analysis was also performed (Table 2) over the same scenarios and the results show that there is no linear correlation between both metrics. Based on our experience, we state that throughput is more related to the processing resources and the operators' complexity. Complex operators may require a longer amount of time to carry out their processing, and also consume more processing resources compared to simpler operators (e.g. filtering).

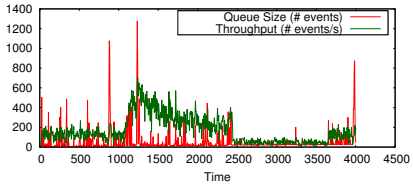
Although throughput and queued events are not directly related, we claim that an autonomous system should consider the amount of queued events when



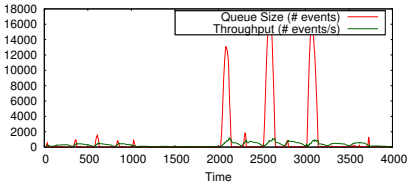
(a) Scenario 1: Mobile



(b) Scenario 2: Quotes



(c) Scenario 3: Tweets



(d) Scenario 4: Web Clicks

Figure 5: Throughput and the number of queued events correlation in 4 different scenarios running on top of the SPS presented in [7].

adapting its topology, otherwise, the end-to-end latency of those events will be compromised affecting the real-time characteristic of the applications built on top of it. In consequence, our hypothesis is that queued events represent a good variable to characterize the performance of any self-adaptive stream processing system. This hypothesis is partially supported by the preliminary results presented in Table 2. We can observe that both variables are related, but not linearly. The focus of our design is on characterizing the behavior of the input traffic and the events' queues on a SPS, and the creation of an index able to quantify the adaptability capacity of the system.

Table 2: Pearson and Spearman coefficient between throughput and queue size in 4 different processing scenarios.

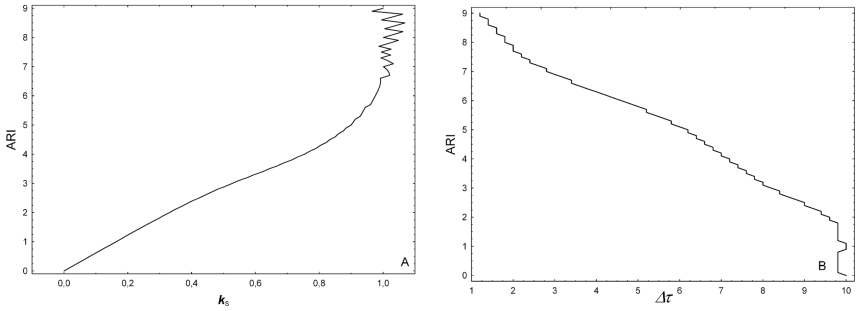
	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
Peason	0.0543	0.7288	0.1797	0.4527
Spearman	0.4224	0.6122	-0.0426	0.5208

3.2. Index

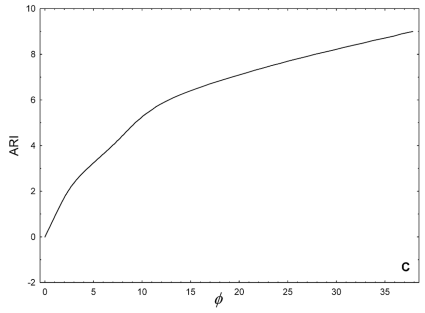
The Adaptation Index for Stream Processing Systems (or AI-SPS) evaluates the adaptability of the system based on the size of the queued events and the capacity of the system to empty all the queues after a traffic spike. Our proposal uses the ARI scale which categorizes the adaptation capacity on a scale from 0 to 9. In humans, the scale states that any ARI over 6 represents a healthy brain, which means it adapts normally to the ABP dynamics.

In order to characterize the adaptation capacity we consider 2 main factors: (1) the capacity of the system to reach a new stable state, and how close the new stable state is to its initial stable state (K_s), and (2) the time required by the system to reach a new stable state (τ). Chacon et. al. [8] proposes 3 parameters to quantify the adaptation capacity of any individual. Two of them are used in this work, and a third called ϕ that represents the angle between the lines that represent the CBFV transient response and the ABP recovery signal. In our case, this angle is useless because in processing systems the input traffic (the equivalent to the ABP in the ARI model) does not always decreases when the system adapts. In humans, the ABP and the CBFV are related and both behave similarly under the activation of the CA process. In our model, queued events are affected when the system adapts; however, input traffic acts independently regardless of the system's response to traffic changes. In order to follow the mf-ARI approach, in this work, we will keep parameter ϕ as a constant, and its value will be studied in the following section.

We assume that initially the system is configured according to "normal" traffic conditions, which can be deduced by analyzing historical data or by applying expert knowledge. In Figure 6 we observe the behavior of the analyzed parameters and their effect on the ARI scale. Based on these results, we define an acceptable ARI for a system between the range of 6 to 9. In Figure 6.b we can observe response time values (or adaptation times) from 1 to 5 seconds. In consequence, in this range, a response time lower than 5 seconds is assured, which is acceptable for most real-time applications. Moreover, in this range (index score 6 to 9) the new stable state is close to the initial stable state; it



(a) K_s (b) τ



(c) ϕ

Figure 6: Parameters behavior and ARI index value [8].

presents less than 10% of variation with respect to the initial state (Figure 6.a). A new state similar to the initial one means that the number of queued events are similar in both stable states. In other words, the system is capable of reacting to and readjusting the system in less than 5 seconds, and this adjustment allows for an up to 90% reduction of queued events. The conditions derived from the study of index values are also valid for most stream processing applications; if we expect real time processing, a 5-second delay is still acceptable.

3.2.1. Stable State

We define a *stable state* (or K_s) as the state where the SPS utilization remains stable and maintains an almost constant number of events in the queues (i.e. between predefined thresholds). The system in a stable state should not be overloaded nor underloaded. In our model, we consider that any SPS starts in a stable state. This assumption is based on the hypothesis that developers or administrators of the processing systems have expert knowledge to set up their processing platforms according to historical traffic workloads. The definition of a stable state allows for the evaluation of the capacity of the adaptive systems to recover from workload dynamics and for the return to their initial stable state or at least close to it. It is important to notice that sometimes the same state is not recovered; however, the closer the new state is to the initial state, the better the capacity of the system to adapt to workload dynamics, and therefore the higher the index value. We observe in Figure 6 that the capacity to recover to a state closer to the initial one is highly related to the higher values of the index (greater than 6).

3.2.2. Recovery Time

Recovery time is an important parameter because it is associated with the total time that it takes the adaptive system to recover (to reach a stable state again). Recovery time or τ is measured when the system reaches the minimum point of the inverse queue of events until it reaches a new stable state again. It is important to mention that in our model, we consider the behavior between

input traffic and the total queued events to be similar to ABP and the CBFV respectively. We assume ABP as the inverse of the input events (Equation 1), and CBFV as the inverse of the total queued events (Equation 2). In consequence, the higher the number of events in the queue, the lower the CBFV.

$$ABP(t) = I^{-1}, \text{where I is the number of input events.} \quad (1)$$

$$CBFV(t) = Q^{-1}, \text{where Q is the number of events in the queue.} \quad (2)$$

A smaller recovery time (or τ) is highly related to higher values of the index. In fact, if we observe Figure 6, we can notice that a recovery time lower than 5 seconds is considered acceptable and is related to higher values of the index. Furthermore, if the system is able to reach a new stable state quickly, the system is capable of reacting in a short period of time to abrupt traffic changes, minimizing the probability of overloading the system. Notice that the system not only has to adapt to the incoming traffic, but it also must consider the number of queued events in its adaptation decision in order to keep events flowing and respect latency restrictions. Reaching a stable state faster means that the system has considered the already queued events in its adaptation planning.

3.3. Methodology

In order to evaluate a SPS using our index, it is necessary to collect information about the input traffic and the total number of queued events. These metrics should be collected using a time window of 0.1 seconds in order to detect the capacity of the system to adapt. Greater time frames could be used by interpolating the behavior of the system on the missed points; however, the precision of the index may be affected. The time frame is a parameter of the system.

With the collected traces we create the equivalent of ABP and the CBFV by applying Equation 1 and Equation 2. Both, input events and total messages in queues are the basic information that the mf-ARI algorithm requires to calculate

the index for the system. The index calculation requires the definition of the initial stable state, to detect the minimum of the inverse queue, and finally define the new stable state reached by the system. These three steps allow for the calculation of the three parameters required by the index.

The initial stable state is easy to detect and is calculated as the average queue size in a time frame of 2 seconds. In order to automatically perform the minimum CBFV detection, our index creates a sliding window that observes and registers the current minimum inverse queue value. Once the minimum is detected, the system registers the time and continues the observation to detect the new stable state. The stable state is calculated in a similar manner as the initial stable state where the average of queued events over a 2 second time frame is calculated. Finally, the stable state difference K_s is calculated as the relative difference between the initial stable state and the new one. On the other hand, the τ value is calculated as the elapsed time between when the system reaches the new stable state and the time the system reaches the inverse queue minimum. With this information, our system is able to calculate the index by observing parameters K_s and τ.

Table 3: Regression coefficients for the standardization procedure proposed by [8] to adjust index to Tiecks's ARI scale.

	Mean ± SD	**Coefficient**
intercept		1.631
K_s	0.69 ± 0.35	3.751
τ	6.19 ± 3.03	-0.137
ϕ	11.43 ± 10.56	0.099

Following the regression method proposed in [8], we can associate these parameters to the ARI scale proposed by Tiecks [9] using Equation 3. The value range for the index was kept similar to the original ARI scale, (i.e. from 0 to 9). In this work, we consider that a system presents a good adaptation capacity when the index value ranges between 6 to 9 on the ARI scale similar to what is proposed by Tiecks. As we discusses in the previous section, under this

range, the index guarantees values for a recovery time (τ) less than 5 seconds and a state (K_s) with less than 10% of variation with respect to the initial states.

Based on the information presented in Table 3 we calculate the adaptation index value following Equation 3.

$$AI - SPS = 3.751 * K_s - 0.137 * \tau + 0.099 * \phi + 1.631 \quad (3)$$

As we discussed previously, Chacon et. al. propose a model that includes the 3 parameters: K_s, τ, and ϕ. In order to follow their regression mechanism, we study the effect of different angle values and observe the index capacity to identify the adaptation capacity under the ARI scale. As ϕ angle cannot be calculated for all the scenarios; therefore, we propose to keep this parameter constant and analyze the capacity of the index to discriminate the systems that adapt well to those which do not. This study is presented in the following section.

4. Evaluation

In this section, we present the experimental evaluation of the AI-SPS index. We have divided our experiments into two. First, we have analyzed the impact of the parameter ϕ over the index calculation. Experiments were performed to analyze the best value for this parameter. Second, we have studied the behavior of the index over different real traffic scenarios applied over two self-adaptive processing systems of literature [7] [24]. Both systems are based on the MAPE control pattern and they apply a two-level strategy to adapt the processing system.

4.1. Datasets Description and Scenarios

For all our experiments we used real traffic datasets extracted from 4 different domains: mobile sensors, news quotes, Twitter, and Web clicks. All these

scenarios generate real-time traffic that can be processed by a SPS to extract relevant data.

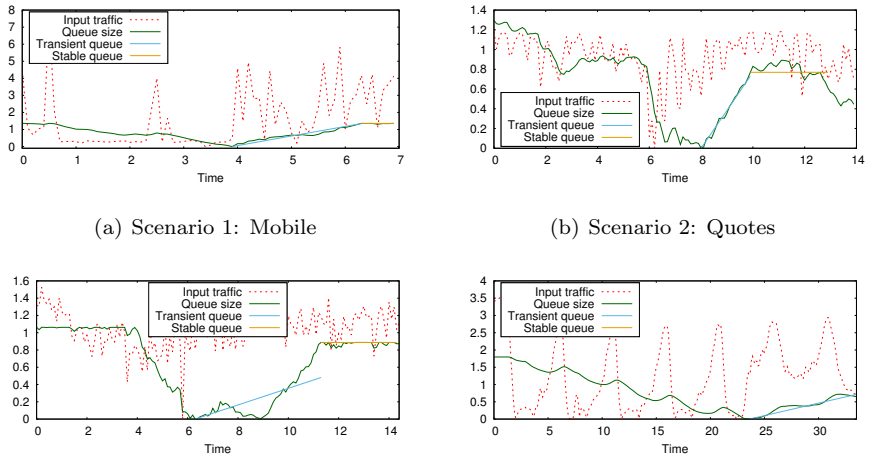
The four datasets used in the evaluation are the same datasets introduced in Section 3. As we previously mentioned, not all these scenarios present traffic spikes, therefore we introduced sudden traffic increments synthetically in order to increase the events' spikes. We followed a similar methodology as proposed by Bodík et al. [13]. Bodik proposed a model of stateful spikes that enables researchers to synthesize volume and data spikes that can be used to stress-test the infrastructure. We have used this model because it is very difficult to obtain realistic traces with high traffic spikes to evaluate adaptability in a SPS. The same model was also used in [7].

The advantage of characterizing the adaptive behavior based on the number of queued events relies on the broad number of SPS that can be evaluated with this technique. Our final goal is to provide a new instrument for the community to measure and compare different adaptive mechanisms. We evaluate 4 traffic spike scenarios over both systems and the results are presented in the following section.

4.2. Experiments

In this section, we present the experiments performed which aim to validate the capacity of the proposed index to quantify the adaptive behavior of the processing platforms. To this end, we employed the same four datasets described previously. All datasets were processed by a generic processing topology built in the adaptive platforms. Both platforms model applications using a directed acyclic graph (DAG). We created a linear topology composed of 4 operators that performed simple processing tasks, such as filtering. The final operator stores the output stream in a non-SQL database such as MongoDB. This topology is implemented in both systems.

Figure 7 presents the results for the 4 previously mentioned scenarios. The spikes that were analyzed were created following the same methodology that Bodik et al. used to introduce traffic spikes.



(a) Scenario 1: Mobile

(b) Scenario 2: Quotes

(c) Scenario 3: Tweets

(d) Scenario 4: Web Clicks

Figure 7: Execution results for the self-adaptive system proposed in [7]. Input traffic and inverse queue size in 4 different scenarios.

The results shown in Figure 7 of the different scenarios were executed on the adaptive stream processing system introduced by Hidalgo et al. in [7]. In each scenario, input traffic (dashed red line), and the queue size behavior is presented (green line). With this information, we estimated parameters K_s and τ. Additionally, the transient queue (blue line) and the stable state (orange line) are presented. The former is the line that represents the inverse queue behavior from its minimum value until reaching a stable state, and the latter, the line that represents the new stable state reached by the system.

For the third parameter (ϕ), we analyzed the behavior of the index in all the scenarios using different ϕ angle values ranging from 0 to 20. For this particular study we only use the system proposed in [7]. This preliminary study attempts to define the τ parameter experimentally. From Figure 6.c, we expect that values close to 15 degrees provide good results. A 15 degree angle is associated with an index value closer to 6, which is within the range that we consider a good adaptive behavior. Table 4 presents the AI-SPS values for the different scenarios and ϕ angles.

Table 4: AI-SPS values for different ϕ angles using the self-adaptive system proposed in [7] .

ϕ	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
0	4.992	3.656	4.111	1.876
5	5.487	4.151	4.606	2.371
10	5.982	4.646	5.101	2.866
15	6.477	5.141	5.596	3.361
20	6.972	5.636	6.091	3.856

Table 5 presents the K_s and τ parameter values obtained from the execution of the scenarios over the platform. These values reflect the behavior of the system when adapting a given scenario. Following the regression model, we calculated the AI-SPS index values for each scenario. The results presented in Table 5 were calculated using a ϕ angle of 15 degrees. This choice is based on the analysis performed on the index values and K_s and τ parameter values. For angles 0 and 5 degrees we can notice that no scenario reaches an index within our acceptable range (6 or more). In consequence, both values seriously compromise the index capacity to discriminate a good adaptation capacity. For example, the result presented in Scenario 1 where the system reaches a K_s=0.98 and a τ=2.4, is a scenario in which the platform has adapted well to the traffic conditions. The K_s and τ values represent a system that was capable of reaching a new stable state with a 2% difference with respect to the initial state, and it took only 2.4 seconds to reach a stable state. This specific scenario is expected to reach an index value greater than 6. Only angles of 15 or 20 degrees are capable of providing such an index value. Between these two options, we selected a ϕ angle of 15 since a 20 degree ϕ angle cannot discriminate well enough. For example, if we focus on Scenario 3, we notice the system reaches a stable state with a difference greater than 10% of the initial state, and τ value of 4.9. Although the latter parameter is within the aceptable range (less than 5 seconds), the initial state is not completely recovered. Therefore a 20 degree ϕ angle is discarded because cannot discriminate well enough. In consequence, for the following results we only consider ϕ with the constant value of 15 degrees for all the

experiments.

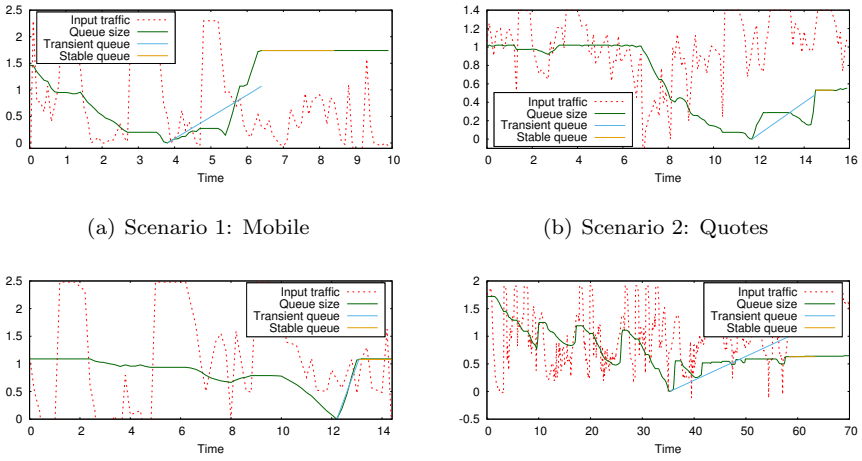
Once the ϕ parameter was fixed, we evaluated the scenarios and their index values. In Table 5, we summarize the AI-SPS results for the different scenarios using the adaptive system from [7]. We observe that the index is able to discriminate scenarios that the system cannot adapt to as well as Scenario 4. In this scenario the system was unable to recover a stable state similar to the initial one. Moreover, the system requires more than 12 seconds to reach that new stable state. Clearly, this is a scenario where the adaptive system cannot adapt well to the traffic dynamic behavior and the index reflects this behavior by scoring the system with a 3.361 on the index scale. On the other hand, the system is able to recognize the scenarios where the system adapts well as in Scenario 1. In this particular case, the system recovers the initial state and reacts in less than 3 seconds scoring 6.477 on the index scale.

Table 5: AI-SPS parameters and AI-SPS index value for the 4 tested scenarios using self-adaptive proposed in [7].

	K_s	τ	ϕ	**AI-SPS**
Scenario 1	0.9837	2.40	15	**6.477**
Scenario 2	0.6055	1.80	15	**5.141**
Scenario 3	0.8400	4.9	15	**5.596**
Scenario 4	0.5327	12.80	15	**3.361**

We also validate the index using the self-adaptive system proposed by Cardellini et al. [24]. Here we use the same scenarios and processing topology previously mentioned. In our tests the system was configured with the default parameters proposed by the author. In Figure 8 we present our results for the different scenarios. From a visual analysis we can notice that the processing system adapts better in Scenario 3, where it reaches a similar stable state after the adaptation, and the adaptation is carried out in a short time frame. This is confirmed by the results presented in Table 6. Here we can notice that Scenario 3 scores 6.706 on the index scale. This score is related to the system's capacity to recover its ini-

tial stable state (0.99) after adaptation and its fast reaction time to the changes in the traffic (0.9 seconds). For the other 3 scenarios the adaptive system was unable to score above a 6. The worst result was obtained in Scenario 4, similar to the previously evaluated processing system.



(a) Scenario 1: Mobile

(b) Scenario 2: Quotes

(c) Scenario 3: Tweets

(d) Scenario 4: Web Clicks

Figure 8: Execution results for the self-adaptive system proposed in [24]. Input traffic and inverse queue size in 4 different scenarios.

We observed that for both self-adaptive systems Scenario 4 was challenging. The systems were unable to react according to the requirements defined in the design of the index; thus, they scored a low AISPS value.

Table 6: AI-SPS parameters and AI-SPS index value for the 4 tested scenarios using the self-adaptive system proposed in [24].

	K_s	τ	ϕ	**AI-SPS**
Scenario 1	0.77	5.20	15	**5.292**
Scenario 2	0.52	2.80	15	**4.683**
Scenario 3	0.99	0.9	15	**6.706**
Scenario 4	0.37	23	15	**1.353**

5. Conclusion

In this work, we have presented the design and evaluation of an Adaptability Index for Stream Processing Systems (AI-SPS), a metric that allows for the comparison of the adaptation capacity of the different Stream Processing Systems (SPS) by analyzing their messages queues. This index is designed for self-adaptive SPSs that are able to dynamically change their internal logic to fit the scenario dynamics. AI-SPS is inspired by the model-free Autoregulation Index (mf-ARI), which was specifically adapted and parametrized for quantifying the adaptation capacity of an adaptive stream processing system.

We evaluated our proposal using two adaptive SPSs from the literature that use the fission technique to increase or decrease their processing resources. We applied our index to these systems using 4 real datasets, from which we extracted the most representative traffic spikes. The index is able to identify the adaptation capacity of the SPS by analyzing two parameters: the queued messages and the time the system takes to return to a new stable state. Using a regression mechanism we are able to map both parameters behaviors to the ARI scale to quantify the capacity of adaptation observed in the evaluated system. Results showed that our index is able to efficiently identify the adaptation capacity of self-adaptive processing systems. From our analysis, we state that any system that scores more than a 6 is considered a system that has a good adaptation capacity.

The lack of a common index is critical when we want to compare different systems. This introduces a bias in the evaluations of different proposals. An index of this nature can help the scientific community to make fair comparisons among literature, and therefore result in better solutions.

As future work, we plan to create a benchmarking framework based on this index. The benchmarking framework considers the development of a set of challenging traffic scenarios and applications. It is also necessary to integrate the index results in order to offer a global adaptation score extracted from the partial results from the different traffic scenarios proposed. Over the next year,

we expect to release this work to the community to enable scientists to develop better adaptive solutions towards the generation of fully autonomic data stream processing systems.

6. Acknowledgment

The authors would like to thank Universidad Diego Portales and *Proyecto Semilla UDP* for partially supporting this research. We would also like to thank the STIC-AmSud collaboration project 17-STIC-05 PaDMetBio.

References

[1] T. White, Hadoop: The Definitive Guide, 1st Edition, O'Reilly Media, Inc., 2009.

[2] S. Appel, S. Frischbier, T. Freudenreich, A. P. Buchmann, Eventlets: Components for the integration of event streams with SOA, in: 2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Taipei, Taiwan, Diciembre 17-19, 2012, 2012, pp. 1–9.

[3] S. Schneider, M. Hirzel, B. Gedik, K.-L. Wu, Auto-parallelizing stateful distributed streaming applications, in: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), ACM, 2012, pp. 53–64.

[4] N. Pollner, C. Steudtner, K. Meyer-Wegener, Operator fission for load balancing in distributed heterogeneous data stream processing systems, in: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, ACM, New York, NY, USA, 2015, pp. 332–335.

[5] X. Wu, Y. Liu, Optimization of load adaptive distributed stream processing services, in: 2014 IEEE International Conference on Services Computing (SCC), IEEE Computer Society Press, Los Alamitos, CA, USA, 2014, pp. 504–511.

[6] E. Zeitler, T. Risch, Scalable splitting of massive data streams, in: Database Systems for Advanced Applications, Vol. 5982 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 184–198.

[7] N. Hidalgo, D. Wladdimiro, E. Rosas, Self-adaptive processing graph with operator fission for elastic stream processing, Journal of Systems and Software 127 (Supplement C) (2017) 205 – 216.

[8] M. Chacón, J. L. Jara, R. B. Panerai, A new model-free index of dynamic cerebral blood flow autoregulation, PLoS ONE 9 (10) (2014) 1–11.

[9] F. P. Tiecks, A. M. Lam, R. Aaslid, D. W. Newell, Comparison of static and dynamic cerebral autoregulation measurements, Stroke 26.

[10] S4, Distributed stream computing platform, [Online] `http://incubator.apache.org/s4/` (October 2016).

[11] Storm, Distributed and fault-tolerant realtime computation, [Online] `http://storm.apache.org` (October 2017).

[12] A. Samza, Samza, [Online] `http://samza.apache.org` (November 2017).

[13] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, D. A. Patterson, Characterizing, modeling, and generating workload spikes for stateful services, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, ACM, 2010, pp. 241–252.

[14] L. Xu, B. Peng, I. Gupta, Stela: Enabling stream processing systems to scale-in and scale-out on-demand, in: 2016 IEEE International Conference on Cloud Engineering (IC2E), 2016, pp. 22–31.

[15] T. De Matteis, G. Mencagli, Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing, in: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '16, ACM, New York, NY, USA, 2016, pp. 13:1–13:12.

[16] T. Heinze, Y. Ji, L. Roediger, V. Pappalardo, A. Meister, Z. Jerzak, C. Fetzer, Fugu: Elastic data stream processing with latency constraints, Data Engineering (2015) 73.

[17] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, P. Valduriez, Streamcloud: A large scale data streaming system, in: Proceedings of the 2010 International Conference on Distributed Computing Systems (ICDCS), Genova, Italy, IEEE Computer Society, 2010, pp. 126–137.

[18] B. Gedik, S. Schneider, M. Hirzel, K.-L. Wu, Elastic scaling for data stream processing, IEEE Transactions on Parallel and Distributed Systems 25 (6) (2014) 1447–1463.

[19] P. A. Smirnov, D. Nasonov, Quality-based workload scaling for real-time streaming systems, Procedia Computer Science 101 (2016) 323 – 332, 5th International Young Scientist Conference on Computational Science, {YSC} 2016, 26-28 October 2016, Krakow, Poland.

[20] B. Satzger, W. Hummer, P. Leitner, S. Dustdar, Esc: Towards an elastic stream computing platform for the cloud, in: Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2011, pp. 348–355.

[21] J. van der Veen, B. Van Der Waaij, E. Lazovik, W. Wijbrandi, R. Meijer, Dynamically scaling apache storm for the analysis of streaming data, in: Proceedings of the 2015 IEEE First International Conference on Big Data Computing Service and Applications (BigDataService), IEEE, 2015, pp. 154–161.

[22] K. G. S. Madsen, P. Thyssen, Y. Zhou, Integrating fault-tolerance and elasticity in a distributed data stream processing system, in: Proceedings of the 26th International Conference on Scientific and Statistical Database Management (SSDBM), ACM, 2014, pp. 48:1–48:4.

[23] T. Heinze, Y. Ji, Y. Pan, F. J. Grueneberger, Z. Jerzak, C. Fetzer, Elastic complex event processing under varying query load, in: Proceedings of the First International Workshop on Big Dynamic Distributed Data (BD3), CEUR-WS, 2013, pp. 25–30.

[24] V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, Towards hierarchical autonomous control for elastic data stream processing in the fog, in: D. B. Heras, L. Bougé, G. Mencagli, E. Jeannot, R. Sakellariou, R. M. Badia, J. G. Barbosa, L. Ricci, S. L. Scott, S. Lankes, J. Weidendorfer (Eds.), Euro-Par 2017: Parallel Processing Workshops, Cham, 2018, pp. 106–117.

[25] G. Mencagli, M. Torquati, M. Danelutto, Elastic-ppq: A two-level autonomic system for spatial preference query processing over dynamic data streams, Future Generation Computer Systems 79 (2018) 862 – 877.

[26] M. R. Mendes, P. Bizarro, P. Marques, Performance evaluation and benchmarking, Springer-Verlag, Berlin, Heidelberg, 2009, Ch. A Performance Study of Event Processing Systems, pp. 221–236.

[27] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts, Linear road: A stream data management benchmark, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, VLDB Endowment, 2004, pp. 480–491.

[28] R. Lu, G. Wu, B. Xie, J. Hu, Stream bench: Towards benchmarking modern distributed stream computing frameworks, in: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 69–78.

[29] A. Shukla, S. Chaturvedi, Y. Simmhan, Riotbench: A real-time iot benchmark for distributed stream processing platforms, CoRR abs/1701.08530. arXiv:1701.08530.

[30] M. A. Lopez, A. G. P. Lobato, O. C. M. B. Duarte, A performance comparison of open-source stream processing platforms, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6.

[31] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, V. Markl, Benchmarking distributed stream processing engines, CoRR abs/1802.08496. arXiv:1802.08496.

[32] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, P. Poulosky, Benchmarking streaming computation engines: Storm, flink and spark streaming, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 1789–1792.

[33] G. Hesse, C. Matthies, B. Reissaus, M. Uflacker, A new application benchmark for data stream processing architectures in an enterprise context: Doctoral symposium, in: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS '17, ACM, New York, NY, USA, 2017, pp. 359–362.