Information and Software Technology 111 (2019) 1–21

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Enactment of adaptation in data stream processing with latency implications—A systematic literature review

Cui Qin*, Holger Eichelberger, Klaus Schmid

University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Universitätsplatz 1, Hildesheim 31141, Germany

ARTICLE INFO

Keywords:
Stream processing
Big data
Runtime adaptation
Enactment
Latency
Systematic literature review

ABSTRACT

Context: Stream processing is a popular paradigm to continuously process huge amounts of data. Runtime adaptation plays a significant role in supporting the optimization of data processing tasks. In recent years runtime adaptation has received significant interest in scientific literature. However, so far no categorization of the enactment approaches for runtime adaptation in stream processing has been established.

Objective: This paper identifies and characterizes different approaches towards the enactment of runtime adaptation in stream processing with a main focus on latency as quality dimension.

Method: We performed a systematic literature review (SLR) targeting five main research questions. An automated search, resulting in 244 papers, was conducted. 75 papers published between 2006 and 2018 were finally included. From the selected papers, we extracted data like processing problems, adaptation goals, enactment approaches of adaptation, enactment techniques, evaluation metrics as well as evaluation parameters used to trigger the enactment of adaptation in their evaluation.

Results: We identified 17 different enactment approaches and categorized them into a taxonomy. For each, we extracted the underlying technique used to implement this enactment approach. Further, we identified 9 categories of processing problems, 6 adaptation goals, 9 evaluation metrics and 12 evaluation parameters according to the extracted data properties.

Conclusion: We observed that the research interest on enactment approaches to the adaptation of stream processing has significantly increased in recent years. The most commonly applied enactment approaches are parameter adaptation to tune parameters or settings of the processing, load balancing used to re-distribute workloads, and processing scaling to dynamically scale up and down the processing. In addition to latency, most adaptations also address resource fluctuation / bottleneck problems. For presenting a dynamic environment to evaluate enactment approaches, researchers often change input rates or processing workloads.

1. Introduction

In recent years, we have witnessed the increasing emergence of data-intensive applications [1,2] such as stock trading, network traffic monitoring and sensor-based data analysis. Such applications aim at providing continuous analysis results to end-users, some even, with rather strict time constraints. Data stream processing [3], i.e., continuous processing of conceptually endless data streams, became a popular paradigm to support this type of data processing. Well-known stream processing frameworks include Apache Storm [4] and Apache Spark [5]. They support the customized implementation of arbitrary stream analysis algorithms, in particular, aiming at real-time processing of complex and distributed stream analysis tasks. Other stream processing engines such as Borealis [6], SPADE [7] and RTSTREAM [8] focus on relational algebra stream operators such as aggregations or joins.

In the continuous processing of data streams, the processing situations may vary over time. On the one hand, we have to deal with the dynamic characteristics of data streams. During processing, the volume or velocity of streams can change drastically. For example, in the financial domain hectic markets can cause bursty streams leading to changes of the stream characteristics of several orders of magnitude [9]. On the other hand, the processing environment can also vary abruptly. For example, the failure of processing nodes can lead to a sudden reduction of available processing resources or network bandwidth fluctuations in video streaming can cause unstable video quality [10]. To cope with such varying processing situations, the need of adapting the behavior of the processing becomes critical. There are different types of adaptation in literature aiming at different problems. For instance, a typical way to handle bursty streams is to apply load shedding [11], i.e., to drop unprocessed tuples to reduce system load. For resource fluctuation

* Corresponding author.
E-mail addresses: qin@sse.uni-hildesheim.de (C. Qin), eichelberger@sse.uni-hildesheim.de (H. Eichelberger), schmid@sse.uni-hildesheim.de (K. Schmid).

https://doi.org/10.1016/j.infsof.2019.03.006
Received 31 August 2018; Received in revised form 19 December 2018; Accepted 11 March 2019
Available online 12 March 2019

problems, elastic resource scaling [12] is commonly applied to dynamically increase and decrease resources according to the processing needs.

Many self-adaptation approaches [13,14] based on the MAPE-K cycle (Monitor-Analyze-Plan-Execute over a Knowledge base) [15] were proposed to support the automatic handling of adaptive processing. Typically, they start by collecting processing information in a monitoring phase and then analyze the monitored data to diagnose adaptation-relevant symptoms. Once a symptom is captured, a treatment plan for it will be established and finally performed in the execution phase (runtime) of the adaptation. In this study, our main focus is, however, to investigate the adaptation throughout the execution phase. Other phases of this cycle, e.g., how the adaptation decision is made in the plan phase or how the monitored data is derived in the monitoring phase, are out of our scope. More specifically, we focus on how the adaptation is executed / enacted after an adaptation decision is already made. In our SLR, we use the enactment to refer to the execution phase of the MAPE-K. We mainly aim at identifying different types of enactment and classifying them into enactment categories. In order to learn the enactment from different perspectives, we introduced three levels of enactment terms. Namely, there are the enactment approach referring to the general concept or idea of the enactment, the enactment technique referring to the realization of an enactment approach from the conceptual point of view and the enactment implementation referring to the actual implementation of an enactment. Throughout the paper, we mainly focus on the enactment approach and the enactment technique, and sometimes use the enactment implementation only if necessary. Generally speaking, the enactment of adaptation results in rearrangement of the processing or the overall system. For example, scaling up or down the processing needs to reorganize the parallelism of a processing node or even a group of processing nodes [16]. In this example, the scaling of the processing is the enactment approach and the reorganization of the processing parallelism is the enactment technique according to our enactment terms.

One of the typical requirements towards continuous data processing is to keep up with data ingestion rates, while providing high quality analytical results as quickly as possible, i.e., with as little latency as possible [3]. In particular, low-latency is the aim of many streaming applications. For example, in financial applications it is crucial to process huge amounts of market data that arrive in real time with varying runtime characteristics with low latency to predict financial risks [17]. Previously, we described how to handle bursty streams by switching stream processing algorithms in a distributed environment [18]. In this work, we particularly focused on minimizing the switching time, i.e., the response time of the adaptation. This sparked our interest in obtaining an overview of other adaptation approaches that focus on latency or response time in the field of stream processing. This publication is the result of this endeavor. More specifically, we are interested in how enactment of runtime adaptation in stream processing copes with latency requirements. In this study, we report a Systematic Literature Review (SLR) to investigate the enactment of adaptation in data stream processing from a latency perspective. In this SLR, our scope is to identify the software-based approaches for the enactment of adaptation. We target the following main research questions:

RQ1. In studies analyzing adaptation in data stream processing from a latency perspective, which enactment approaches exist?
RQ2. What techniques are used to realize which enactment?

In summary, the main contributions are:

- We provide a taxonomy of enactment approaches based on the ones identified in our SLR, which results in 17 different categories.
- We extract enactment techniques that represent the realization of each identified enactment approach from the publications and classify them into different categories.
- To gain a better understanding of the purpose of enactment approaches, we extract the addressed problems and the goals from the publications. Based on this, we present a classification of the general problems as well as the general goals.
- We also identify the metrics used to evaluate the enactment approaches to obtain an overview of their performance focus. Moreover, we extract parameters used to trigger the enactment of adaptation in the evaluation to understand the typical ways of using enactment approaches in a dynamic processing environment.

This paper is organized as follows. We describe the background and the related work in Section 2. In Section 3 we present the SLR protocol as our research methodology and in Section 5 we discuss the threats to validity. In Section 4 we provide our results and answer our five major research questions. Finally, we conclude our SLR in Section 6.

2. Background and related work

In this section, we give a brief introduction to data stream processing as the background of our SLR (Section 2.1) and also discuss related work (Section 2.2).

2.1. Data stream processing

Data stream processing continuously executes data processing tasks on potentially unbound streams of data items (also known as tuples) [3]. It focuses on gathering, processing and analyzing a surge of live data streams coming from different domains. Example domains are sensor networks [19], financial markets [20] or video streaming [21]. Typically, a stream processing application is represented as a data flow graph that consists of data sources, data processors and data sinks. Data sources produce input data that are fed into data processors for the actual processing. Data processors are commonly referred as operators or processing nodes (the terms are interchangeably used in this paper). Example processors can be filter, normalizer, aggregator, or other types of processing, e.g., the customized types of processing supported in Storm [4]. After the input data goes through all data processors, the processed results are finally passed to data sinks for delivering to external applications, which present the analysis results to end users.

Data stream processing can be traced back to early Information Flow Processing (IFP) systems [22] that have been grouped into several broad classes such as Continuous Query (CQ) systems and Complex Event Processing (CEP) systems [3]. These two classes are often researched as specific types of stream processing in literature [11,23,24]. Continuous query processing performs a long-running query that needs to be evaluated repeatedly against newly arrived data and continuously provides a stream of results to end-users [2]. The continuous queries express information monitoring requests and are executed based on a set of steps for returning information updates. These steps are so-called query plan. Due to the continuous nature of data streams, queries over streams often return results based on partitions of streams and periodically update their results upon newly arrived data. The partitions of streams here are called windows. There are different window types and a typical one is the sliding window that partitions streams into sub-streams and slides over them with a progress interval [16]. Aurora [25], Borealis [6] and TelegraphCQ [26] are typical engines supporting continuous query processing. Complex event processing aims at detecting interesting events or situations by continuously monitoring and analyzing incoming stream data generated by the observed target applications [2,22]. Cayuga [27], SASE [28] and SnoopIB [29] are typical example complex event processing systems we found in the selected literature. Conditions associated with the events are evaluated continuously during processing and corresponding actions are triggered based on the monitored information. For instance, in a car accident detection and notification system [2], the events are filtered, analyzed and interpreted to understand the currently happening situations and corresponding actions such as notifying the nearest ambulance.

Real-time processing of live streams, i.e., providing analysis insights in real-time to end-users, is the fundamental goal of many stream processing applications [3]. For example, in the financial domain, data is usually provided as time-series and analyzed to generate financial alerts that impose very strict time constraints, e.g., on the scale of milliseconds. However, with the typical problems of limited available resources as well as dynamic characteristics of data streams, real-time processing is still a challenging research topic. In literature, enabling adaptation for stream processing is a common way to achieve (near) real-time data processing, e.g., feedback-based load shedding to adaptively drop unprocessed tuples to guarantee the timeliness requirement [11] or dynamically scheduling processing based on the workload status [30].

2.2. Related work

To identify related work to our SLR, we searched for surveys and SLRs that are close to our topic. Below we discuss related work based on surveys or SLRs of the identified topics, i.e., stream processing systems, stream processing optimizations, adaptive query processing and self-adaptive systems.

Surveys of stream processing. Stephens [31] conducted a literature survey to obtain an overview of the development of stream processing in the late 90s. The study focuses on dataflow literature, programming techniques, languages and the theoretical perspectives of stream processing. Cugola et al. [22] surveyed 34 Information Flow Processing (IFP) systems. They studied different aspects of an IFP system and provided a classification for these systems based on their characteristics. Motivated by the increasing emergence of distributed data processing systems, Kossmann [32] published a work presenting the state of the art of query processing for distributed database and information systems. These studies only focus on the classification and general techniques of stream processing systems, but not on adaptation.

Surveys of stream processing optimizations. Hirzel et al. [33] conducted a survey that covered eleven stream processing optimizations such as operator reordering and operator placement. They discussed each optimization in detail from several aspects, e.g., the conditions to apply the optimization and the constraints for the optimization to preserve correctness. There also exist several surveys explicitly for query optimization. Kremer et al. [34] reviewed various cost models, search strategies of finding optimal query execution plans, resource allocation techniques and load balancing approaches in parallel database systems. Similarly, Aljanaby et al. [35] discussed different search spaces, search strategies and cost models relevant for the distributed query optimizer. Jarke et al. [36] discussed query optimization in the framework of relational calculus queries in centralized database systems. They focused on query representation, query evaluation and different types of database systems in which the query optimization is applied. Some of the presented optimizations in these studies overlap with the enactment approaches in our review. However, our study differs in the following aspects. First, we focus on enactment approaches of adaptation in stream processing from a latency perspective, which is a more specific topic than general optimization. Second, our study is a systematic literature review which differs a lot from a survey in terms of completeness, consistency, etc.

Surveys of adaptive query processing. Gounaris et al. [37] presented a survey on adaptive query processing techniques. The authors provided an overview of these techniques along with their characteristics, e.g., the focus or the aim of the techniques. Deshpande et al. [38] conducted a survey to identify common issues, themes and approaches in adaptive query processing, in particular, focusing on adaptive join processing. Some surveyed techniques in above studies are covered by our identified enactment categories. However, they only focus on query techniques of some known papers and also no systematic categorization of enactment of adaptation is provided.

SLRs and surveys of self-adaptive systems. Mahdavi-Hezavehi et al. [39] performed a systematic literature review on architecture-based methods for handling multiple quality attributes (QAs) in self-adaptive systems. Yang et al. [40] conducted a SLR of requirements modeling and analysis for self-adaptive systems. Other surveys focused on the engineering perspective of self-adaptive systems. Becker et al. [41] presented a taxonomy of self-adaptation and surveyed on engineering self-adaptive systems while Krupitzer et al. [42] studied model-driven approaches for performance analysis and determined a classification for the identified approaches. There are also surveys [43,44] discussing approaches, research challenges, and applications of self-adaptive systems. These studies focus on general approaches of self-adaptive systems but not on stream processing.

To the best of our knowledge, there is no systematic literature review that focuses on the enactment of runtime adaptation with respect to latency in the field of stream processing.

3. Research methodology

In this section, we describe the design and the execution of our systematic review. We adopted the phases and steps recommended by the guidelines of Kitchenham et al. [45]. Following this review process, we discuss our review protocol below. First, we give a detailed description about our research questions. Then we describe the search strategy and the search process for identifying relevant studies in Section 3.1. In Section 3.2 we explain the inclusion and exclusion criteria as well as the detailed study selection procedure. Afterwards we discuss the quality assessment of the included studies in Section 3.3. Finally in Section 3.4 we provide a short introduction to our data extraction process as well as the detailed description about data properties to be extracted.

Table 1 illustrates the *research questions* that we focus on for this systematic review. **RQ1** and **RQ2** are the core questions of our systematic review, which respectively investigate the enactment approaches of adaptation from a latency perspective and their realization techniques (explained in Section 1). Besides the details of enactment approaches, **RQ1** also contributes a taxonomy of the enactment approaches we identified from the publications. **RQ3** explores the general processing problems addressed in the publications to understand the motivation of applying enactment approaches. We are also interested in the (measurable) goals that respective enactment approaches aim at (**RQ4**). Finally, **RQ5** aims to gain an insight into the evaluation of the enactment approaches. It is divided into two sub-questions. **RQ5.1** pertains to the metrics used to evaluate the enactment approaches. These metrics can be linked back to the identified goals in **RQ4** to determine whether the enactment of adaptation is sufficiently evaluated regarding to its predefined goals. **RQ5.2** investigates the evaluation parameters that are used to present the dynamic evaluation environment for triggering the adaptation. From this question, we aim at obtaining an overview of what parameter settings are applied to what enactment for its evaluation.

In a SLR, a *review protocol* is defined to reduce the potential researcher bias and to support the repeatability of the review [46]. We first created an initial review protocol covering all possible technical elements of a review (guided by [45]) and then discussed it in our review team meeting until all members (three) agreed on the protocol content. The review protocol was also evolved during the actual execution of the SLR and revised based on the results of the discussions of review team meetings. A summary of the final protocol is given in Sections 3.1– 3.4.

3.1. Search strategy

In our review, we conducted an automatic search in two digital libraries, namely ACM[1] and ScienceDirect[2], to identify relevant primary studies. For ACM, we searched with "*The ACM guide to Computing Literature*" to include also literature from other publishers (see Appendix A).

[1] http://dl.acm.org.
[2] http://www.sciencedirect.com.

Table 1
Research questions for the systematic review.

ID	Question	Aim
RQ1	In studies analyzing adaptation in data stream processing from a latency perspective, which enactment approaches exist?	To identify and classify the enactment approaches in the studies that address the adaptation in data stream processing from a latency perspective.
RQ2	What techniques are used to realize which enactment?	To identify a set of categories of enactment realization techniques and to obtain an overview of which techniques are usually used for which enactment.
RQ3	What problems are addressed by the publications?	To identify the general processing problems addressed by the publications, which reflects the motivation of applying their proposed enactment approaches.
RQ4	What are the (measurable) goals aimed at in the publications?	To identify the goals that the publications aim at through their proposed approaches. We mainly focus on measurable goals such as minimizing latency or throughput.
RQ5	How are these approaches evaluated in the publications?	To gain an overview of how the enactment approaches are evaluated. This is detailed in the following sub-questions of RQ5.1 and RQ5.2.
RQ5.1	Which metrics are used for the evaluation of these enactment approaches?	To identify the metrics that are used to evaluate the enactment approaches.
RQ5.2	Which parameters are varied / studied in the evaluation of the enacment approaches?	To identify the parameters that are varied to trigger the enactment in the evaluation. In other words, we identify the parameters that are used to simulate the dynamic environment for evaluating the enactment, e.g., varying input rates.

Table 2
Search terms for the identification of relevant work.

Population	Intervention	Outcome
"stream processing" OR "continuous query" OR "stream-based system" OR "data stream system" OR "streaming system" OR "complex event processing"	"adapt*" OR "reconfigur*"	"latency" OR "response time"

We adopted the search strategy suggested by the Kitchenham's guidelines [46]:

- Extract main search terms from the research questions and the related topics to be researched. This initially derives the potential terms relevant to our SLR search.
- Consider possible synonyms, related terms and alternative spellings for the initially identified terms.
- Iteratively conduct pilot searches and use a test set containing relevant papers to validate the completeness of the search.

Again recommended by Kitchenham et al. [46], we broke down the research questions into individual facets of *population, intervention* and *outcome* (see Table 2) and formed the search string by composing these three facets with Boolean ANDs. It is worth mentioning that different search engines have different query syntax and it is not possible to use the same search string for different digital sources. For our search we ensured that the search strings used in different search engines are logically and semantically equivalent. To conduct our search, for both ACM and ScienceDirect we used the functionality of the advanced search to perform a query search string which fits the syntax of the respective search engines (detailed search strings are given in Appendix A).

We iteratively conducted a pilot search to refine our search terms by checking the search result against known primary studies that we obtained from the references of our previous work [18]. This strategy leads to a rather precise set of results as we verified with a test set of already known papers. The test set we used contains 10 relevant papers, i.e., [21,47–55]. The search string captured 9 out of 10 reference publications based on the test set. One paper [50] was not found by our search because the used intervention terms are not appearing in the searched fields. From this paper we extracted another intervention term "optimize" and conducted a trial search by including it. However, this returns too many irrelevant results as it is too general than the adaptation we search for. So we decided not to introduce the "optimize" in our search term but we are aware of the threat that there might be some papers missing (see Section 5). For our final search result, the paper [50] is included. With the overall result (90% found), we believe that we reached a substantial level of the search completeness.

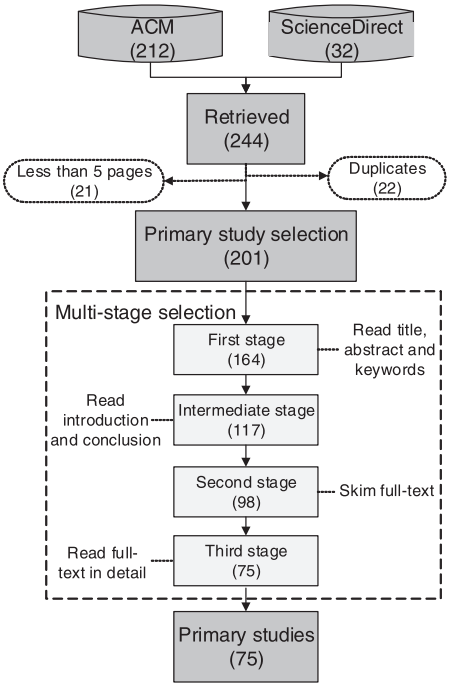

Fig. 1. The result of the study selection process.

The search was conducted on 20-11-2018.[3] We identifed 212 relevant papers including [50] from ACM and 32 from ScienceDirect (see Fig. 1).

[3] Our search results are accessible at: https://doi.org/10.5281/zenodo.2425499.

3.2. Study selection

We conducted the study selection with a multi-stage approach (see Section 3.2.2) by which we incrementally read different parts of the primary studies and removed irrelevant papers from the candidate papers based on the corresponding inclusion and exclusion criteria (see Section 3.2.1). The study selection was mainly conducted by the first author and cross-checked by the other two authors. The final inclusion and exclusion criteria as well as the selection itself were discussed among all three authors. Afterwards, the selection was adapted accordingly.

3.2.1. Inclusion and exclusion criteria

The formulation of the selection criteria aims at identifying those studies that can provide evidence to answer the research questions [45]. Our inclusion and exclusion criteria were initially determined based on our research questions (see Table 1) and later evolved and refined during the detailed reading in the multi-stage selection process. In particular, we defined more specific criteria (see below IC5-6 and EC5-10) when we gained insight into the actual content of the primary studies. Our final inclusion and exclusion criteria are listed below:

- Inclusion criteria (IC): IC1-3 must be met by all papers and IC4-6 are alternatives where it is sufficient that a paper adheres to one of them.
  IC1: Study is written in English.
  IC2: Study was published between 2006 and 2018.
  IC3: Study is in the field of data stream processing.
  IC4: Study discusses the enactment of adaptation in stream processing, i.e., the execution phase of adaptation based on the MAPE-K terminology [15].
  IC5: Study is about adaptation on the level of stream processing algorithms, e.g., input adaptation of a specific algorithm.
  IC6: Study discusses scheduling approaches that cause a change of the processing in the execution phase.

- Exclusion criteria (EC): EC1-4 are our initial exclusion criteria and EC5-10 are more detailed criteria that we extracted while conducting the selection process as a response to border cases.
  EC1: Study is less than 5 pages.
  EC2: Study is not a peer-reviewed paper (e.g., a technical report or a thesis).
  EC3: Study is a survey.
  EC4: Study has no evaluation part.
  EC5: Study deals with the adaptation for the monitoring approaches in stream processing, but does not involve the execution (phase) of the processing or the overall system.
  EC6: Study is only about scheduling policies and does not target their actual execution that could lead to a rearrangement of the processing.
  EC7: Study is about frameworks for supporting the adaptation of the processing but does not focus on the adaptation itself or the adaptation is only mentioned in use cases, examples, scenarios, etc.
  EC8: Study only discusses the impact of the adaptation of the processing but no concrete approach is proposed.
  EC9: Study focuses on hardware-related approaches (e.g., reconfigurable FPGA [56]) and no software solution is provided.
  EC10: Study only mentions adaptation but details are not sufficiently described (not possible to identify the adaptation techniques in the data extraction process).

3.2.2. Study selection procedure

For the study selection, we applied a multi-stage process (adapted from Kitchenham et al. [45]) which is divided into four stages (see the description of the stages below). We incrementally read different parts of the studies and determined the selection decisions based on the respective criteria. We started the early screening on the title and abstract, then read the introduction and conclusion sections, afterwards skimmed the full text and finally read the full text in detail.

We also incrementally applied our inclusion and exclusion criteria to each selection stage (see the described criteria in the respective stage below) depending on the part of the paper being read. For instance, for the very first stage in which only title and abstract are being read, we relaxed our inclusion criteria to consider also studies that discuss any kinds of adaptation (occurring in all phases of MAPE-K) in stream processing (we refer this to relaxed IC4) instead of restricting the adaptation in the execution phase to avoid excluding papers with insufficient information.

The four selection stages are detailed as follows:

- **First stage** by reading title and abstract. For this stage, we consider any sort of adaptation in the field of data stream processing (IC3 and relaxed IC4). The reason to adjust the inclusion criteria for this stage is to avoid excluding studies too early without reading enough information.
- **Intermediate stage** by reading introduction and conclusion to critically check the studies selected from the first stage. For this stage, we applied IC3 and (not-relaxed) IC4 to consider adaptation at execution time, i.e., the enactment of adaptation.
- **Second stage** by skimming the full text (including figures, tables, algorithms, etc.) of the study to double-check the included papers after the intermediary stage (IC3-4 and EC2-4). In particular, we double-check the papers which were marked as uncertain in the last stage.
- **Third stage** by reading the full text in detail to critically double-check the full selection based on all defined inclusion (IC1-6) and exclusion (EC1-10) criteria.

Fig. 1 illustrates the results of the study selection process. From the 244 retrieved publications, we first removed 21 duplicates (the same title and authors or one paper has major overlapping part with another paper but no additional contribution in terms of adaptation approaches, e.g., a conference paper published later as a journal paper) and 22 studies with less than 5 pages (EC1). In total 201 papers were considered in the multi-stage selection process. Afterwards, we followed the stages for selection described above. For example, at the first stage by reading the title, abstract and keywords as well as checking if the study involves any kinds of adaptation, 164 studies remained for the second stage. After conducting all four selection stages, i.e., the detailed full-text reading is finished, 75 studies were included as the result of this selection process.

Following the recommendation by Kitchenham et al. [45], our study selection was performed by one (the first) author and cross-checked by the other two authors. For each selection stage we conducted a cross-checking on all candidate papers. Also, for each stage we set up a review meeting to discuss the cross-checking results and resolve all potential issues that appeared in our selection process. The final results were discussed and agreed among all three authors.

3.3. Quality assessment

For SLRs it is essential to assess the quality of the primary studies and the quality results can be used to guide the interpretation of the synthesis findings [46]. There exist several versions of quality checklists in recent guidelines and experience publications on SLR [45,46,57,58]. The closest one to our research topic is the quality checklist proposed by Dybå et al. [58]. It was developed and used for a qualitative, technology-focused systematic review in software engineering [45]. Based on this checklist as well as taking our research questions into account, we developed a quality checklist that can reflect the quality of the data to be used to answer our research questions. Table 3 (column 2–3) illustrates the quality assessment questions mapped to our research questions.

As recommended by Kitchenham et al. [45], a consistent scoring of the selected studies against the quality assessment criteria is necessary to conduct the study quality assessment. For this we adopted a ternary

Table 3
Quality assessment criteria.

ID	Quality assessment question	Research question(s)	Result	
			Yes	Partly
QA1	Did the paper clearly explain the problem to be addressed within the research?	**RQ3**	61 (81.3%)	14 (18.7%)
QA2	Is there a clear statement of the goal of the research?	**RQ4**	47 (62.7%)	28 (37.3%)
QA3	Is there an adequate description of the context in which the research was carried out?	Overview of the studies	56 (74.7%)	19 (25.3%)
QA4	Did the paper provide enough details about the enactment approach to enable us conduct the required analysis?	**RQ1, RQ2**	68 (90.7%)	7 (9.3%)
QA5	Are the findings of the research clearly stated?	**RQ5** (**RQ5.1** and **RQ5.2**)	71 (94.7%)	4 (5.3%)

Table 4
Extracted properties.

ID	Property	Research question(s)
P1	Research context	Overview of the studies
P2	Problem	**RQ3**
P3	Goal	**RQ4**
P4	Enactment technique	**RQ2**
P5	Enactment approach	**RQ1**
P6	Evaluation metric	**RQ5.1**
P7	Evaluation parameter	**RQ5.2**
P8	Latency implication category	Overview of the studies

scale [45] in which we consider 3 levels of scores, i.e., Yes, Partly and No. The studies with "No" scored were already excluded during the selection process with some of our exclusion criteria, e.g., EC10 excluded the "No" scored papers for QA4. We also conducted a cross-checking for the quality assessment so that the scores were discussed and agreed among two authors. The result (column 4 in Table 3) shows that for all quality questions all included studies were scored with either Yes or Partly, which indicates all quality assessment questions can be positively answered for our remaining papers.

3.4. Data extraction

We performed the data extraction in an iterative manner. Already in the second selection stage, while skimming the full text of the papers, we started to collect data for some properties, including the objectives of the paper, the enactment approach, and the techniques applied in the enactment of adaptation to get an initial overview on the taxonomy of the enactment approaches. From our experience, this is helpful for verifying the extracted enactment approach and also refining the right enactment terms during data extraction later on. For the actual data extraction process, we derived the properties to be extracted based on the defined research questions. Table 4 shows the extracted properties and also the mapping to their relevant research question(s).

In the following subsections we explain the content to be extracted for each data property listed in Table 4 as well as their extraction process. In addition, we discuss their relation to our research questions as well as the focus of their contribution to the final review results.

3.4.1. Research context (P1)

We identified the research context in which the enactment of adaptation is performed. The aim of identifying this property is to categorize the research context from different perspectives in order to obtain an overview of the distribution of the papers to the identified context categories as well as understand the relation between context and applied approaches. We classified the research context based on two perspectives:

(a) **High-level context:** we mainly identified three high-level stream processing classes based on the discussed classification of stream processing systems in Section 2.1.
- *Continuous query processing:* if the study focuses on continuous queries and the adaptation is studied on the level of query processing.
- *Complex event processing:* if the study deals with complex events and the adaptation is researched in the context of event processing.
- *Data stream processing:* if the study is in neither of above two context classifications and discusses the adaptation in the general context of data stream processing.

Studies may target multiple criteria, in particular, CQ and CEP may occur together. For this case, we classify them to both contexts.

(b) **Specific context:** we also identified more specific contexts if they are given in the reviewed study. For example, some studies discuss their approaches for specific applications, e.g., video streaming applications and some focus on specific stream processing frameworks, e.g., Apache Storm [4]. Based on the presented context we categorized the studies as follows:
- *Specific application:* if the study focuses on a specific application domain such as web or video streaming application. Multiple applications can be extracted from one study.
- *Specific framework:* if the study focuses on the techniques of a specific stream processing framework such as Apache Storm [4] or Continuous-MapReduce [59]. Notice that if a processing framework is only used for implementation or evaluation in the study, we do not consider it as context.
- *Specific query:* if the study focuses on a specific query type, e.g., join query.
- *Specific data type:* if the study deals with specific types of data, such as linked data or out-of-order data.
- *General:* if no specific context can be clearly extracted.

Multiple specific contexts may appear in one study. For this case, we categorize the study into each occurred specific context.

3.4.2. Problem (P2)

We extracted the processing problems that are addressed by the publications. This directly answers our research question **RQ3**. It indicates the motivation of the adaptation, i.e., why such an adaptation is needed. From this property, we can obtain insights into the relation between the problems and the enactment of adaptation, i.e., what problems are handled by what enactment.

3.4.3. Goal (P3)

We extracted the goal that the publications aim at. We mainly focus on the measurable goals that indicate the performance metrics to be aimed at. It answers our research question **RQ4**. Later on we mapped this data to the extracted *problem* property as well as the *evaluation metrics* to assess the quality of the proposed adaptation, e.g., whether the goal reflects the processing problem to be addressed in the publication and whether the aimed metrics are evaluated.

3.4.4. Enactment techniques (P4)

We identified the enactment techniques reported in the inspected studies. More specifically, we collected the details of how the enactment is realized in our review data sheets. The extracted details of the technique are used to derive the generalized enactment approaches (see Section 3.4.5) later on.

With this property we aim at identifying how enactment is realized and what specific techniques are applied. This provides an initial overview of enactment approaches and later enables us to obtain a more comprehensive classification of these approaches. As a result, we aim at obtaining an overview of the applied techniques for each individual enactment approach which answers research question **RQ2**.

3.4.5. Enactment approach (P5)

We identified enactment approaches based on the extracted enactment techniques as described in Section 3.4.4. During the extraction, we discovered that the category terms of enactment approaches were not always consistently used in literature. In some studies the enactment categories can be easily identified, e.g., load balancing, while in some no clear terms are mentioned. The aim of identifying this property is to provide a consistent classification of enactment approaches. We applied an iterative method to derive enactment categories, i.e., first defining the initial ones, then evaluating with the identified enactment techniques and finally making the classification decisions (details in Section 4.2). The identified categories of enactment approaches were finally discussed and agreed among all authors.

The classification of enactment approaches answers research question **RQ1**. We mainly aim at a taxonomy which captures potential enactment approaches in literature as well as a quantitative overview of the existing enactment approaches.

3.4.6. Evaluation metrics (P6)

We identified the performance metrics that are evaluated for the enactment approach in the evaluation part of the paper. We only collected the metrics relevant to the enactment approaches if clearly stated in the paper. If they are irrelevant to the enactment or not clear enough to be extracted, e.g., overlapping with the evaluation of other properties of the approach or the respective system, we do not include them. When several enactment approaches exist in one study, evaluation metrics are mapped to the respective enactment approaches, i.e., what metrics are evaluated for what enactment. This aims at answering research question **RQ5.1**.

3.4.7. Evaluation parameter (P7)

For this property we focused on the parameters that are used to simulate the dynamic environment for evaluating the enactment approaches. The parameters we extracted here are the ones used to trigger the enactment of adaptation in the evaluation, e.g., the varying workload for triggering a load balancing approach. We are mainly interested in the parameter itself but not the details of the values of the parameters, e.g., different values of input rates used in the evaluation. This answers research question **RQ5.2**.

3.4.8. Latency implication category (P8)

As we aim at identifying enactment approaches from a latency perspective (see **RQ1**), we are interested in how latency is being addressed in the included studies. For that, we identified the latency context and classified them into the following categories:

- *Latency-oriented*: if the study aims at optimizing the latency, e.g., achieving low-latency.
- *Latency-aware*: if the study aims at a latency bound, e.g., providing latency guarantees or minimizing latency violations.
- *Evaluated*: if the study is neither latency-oriented nor latency-aware but considers the latency in the evaluation.
- *Discussed*: if the study does not explicitly focus on latency but provides a brief discussion of the latency impact.

4. Results and analysis

A total number of 75 papers were included in our review. We first give a short overview (Section 4.1) of the general characteristics of included studies in terms of the publication year (Section 4.1.1), publication type (Section 4.1.2), and research context (Section 4.1.3). Afterwards we discuss the result for each research question that we aim to answer in this SLR in Sections 4.2 to 4.6.

4.1. Overview of the studies

4.1.1. Publication year

Fig. 2 illustrates the distribution of the primary studies over time. The reviewed papers were published between 2006 and 2018. The chart shows considerably increasing research interest on enactment approaches in stream processing over the years. In particular, a huge spike appeared in 2016. Although we observed a lower number of papers in 2017 and 2018 (partial year), there is still an overall upward trend in this topic in the recent decade.

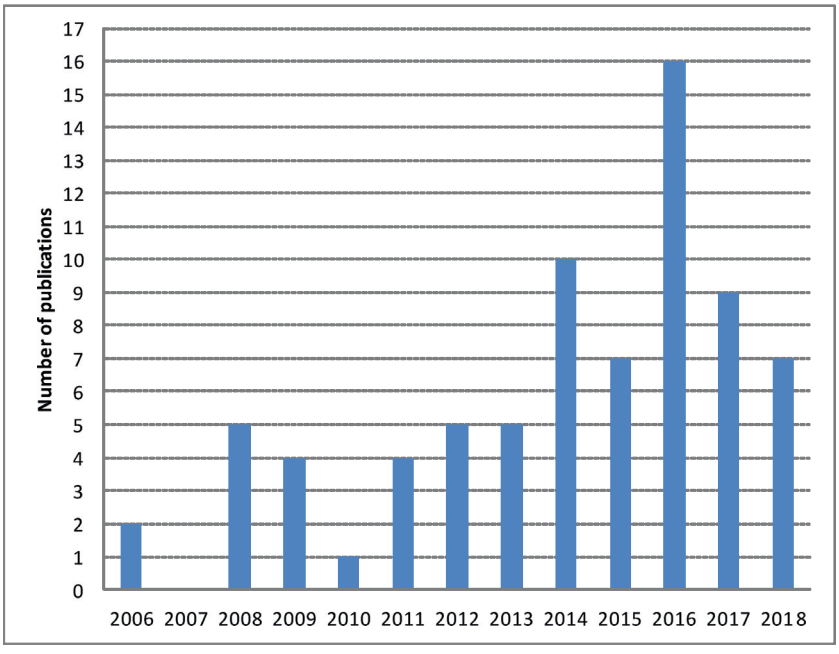

Fig. 2. Distribution of the selected primary studies over time.

Table 5
Overview of the latency implication categories.

Latency implication category	Papers	Number of papers
Latency-oriented	[10,12,13,47,51,53,59,61,63,65,68,69,75,76,78,80,83–93]	27
Latency-aware	[16,21,48,50,52,62,70,74,94,95]	10
Evaluated	[54,66,72,77,79,96,97]	7
Discussed	[11,14,30,60,64,67,71,73,81,82,98–118]	31

4.1.2. Publication type

Among 75 included studies, most were published in conferences (39, 52%) and journals (29, 38.7%). Only a few (7, 9.3%) are workshop papers.

4.1.3. Research context

As indicated in Section 3.4.1, the purpose of identifying the research context is to get an overview of the involved background or environment of the included primary studies. We discuss the result for the study context below:

I. **High-level context**: We categorized the majority of the papers (54, 69.2%) into the general context of data stream processing, followed by continuous query processing (17, 21.8%) and complex event processing (7, 9%). Among them, three papers [60–62] target two contexts, continuous query processing and complex event processing. All other papers could be clearly categorized into one of these three classes.
II. **Specific context**: We identified specific contexts from more than half (43, 57.3%) of the included studies. The "specific application" category is predominant (26 studies), followed by the specific framework (14), the specific query (5) and the specific data type (4) category. As described in Section 3.4.1, multiple specific contexts can appear in one paper. Below we discuss their respective results in detail:
   - *Specific application:* We mainly identified five classes of application domains. The main one is multimedia streaming (21) that focuses on audio and video streaming [10,63–65]. Four papers [60,61,66,67] deal with mobile applications, i.e., mobile data processing. One paper [68] focuses on web applications, one [69] on wide-area streaming systems and one [70] realizes a smart grid application handling power events from electrical power systems.
   - *Specific framework:* For this category, we identified the well-known stream processing framework Apache Storm [71,72], Twitter's Heron [73], the stream processing engine Borealis [52] and nine frameworks specifically created by the authors in the context of their study. The proposed frameworks include Continuous-MapReduce [59], Teddies [74], Enorm [16], ACQUA [60], SPARQL [75], Dhalion [73], Drizzle [76], a parallel stream processing engine [51] and a query modification framework [77].
   - *Specific query:* Four studies [54,78–80] discuss specific adaptation techniques for join queries and one [81] focuses on spatial preference queries.
   - *Specific data type:* In stream processing, data types can vary from different application domains or different data sources. 21 studies focus on multimedia data and three on mobile data (see "specific application"). Additionally, we identified linked stream data in [82] and Resource Description Framework (RDF) data in [75].
   - *General:* We categorized 32 studies (42.7%) into the "general" context, i.e., the high-level context.

4.1.4. Latency implication category

Based on the categories discussed in Section 3.4.8, we identified in total 37 (49.3%) papers that regard latency as their main focus, i.e., 27 for the "latency-oriented" category and 10 for the "latency-aware". 7 (9.4%) papers are identified for the "evaluated" category that consider latency in their evaluation. 31 (41.3%) provide only a discussion of latency. The detailed list of papers is given in Table 5.

4.2. RQ1: In studies analyzing adaptation in data stream processing from a latency perspective, which enactment approaches exist?

The purpose of this research question is to identify runtime enactment approaches and to derive a taxonomy based on the identified ones. As explained in Section 3.4.5, we derive the generalized enactment approaches based on the extracted enactment techniques. The result is illustrated in Fig. 3. We categorized the adaptation into three high-level categories. In this section, we first give an introduction to the taxonomy, and then present a quantitative overview of the identified enactment approaches.

(a) **Resource**: This category includes the enactment approaches dealing with resources used in stream processing. In our taxonomy it represents a physical or virtual computing resource that is used to process data streams. It includes (1) **physical machines**, e.g., physical nodes or servers in a cluster providing computing resources for processing jobs [84]; (2) **virtual (logical) machines**, e.g., virtual processing nodes provided by cloud providers like Amazon and Google [55]; (3) **CPU**, e.g., CPU consumption during data processing [48]; (4) **co-processor**, e.g., Graphics Processing Unit (GPU) as efficient resources for manipulating computer graphics and image processing, e.g., used to accelerate the video analysis [106]; (5) **memory**, e.g., memory required for stream join processing [54]; (6) **cache space**, e.g., specialized memory acting as buffers to prevent transmission overflow [21]; (7) **network bandwidth**, e.g., allocated for video transmission [115]; (8) **storage**, e.g., external or distributed storage system such as HDFS [119].
We identified the following resource-related enactment approaches:
   - **(Dynamic) Resource Allocation**: The approach aims at dynamically assigning/allocating available resources to the data processing, e.g., meeting certain constraints representing the processing requirements to achieve the most efficient resource usage. The determination of resource allocation typically takes into account the characteristics of the processing, e.g., data items or processing nodes, as well as the available processing resources. For example, Babazadeh et al. [68] allocate the most suitable worker for an operator based on its energy consumption, parallelism capability, etc.
   - **Resource Mirror**: The approach uses additional resources in parallel for running the same processing or for backing up the data items or the processing states. Typically, this is applied to achieve fault tolerance, e.g., additional queues are used to backup the unacknowledged data items to enable recovery in case of failures [116].
   - **Resource Switching**: The approach changes the type of resources to be used for processing in order to meet given processing requirements, e.g., in high workload situations switching to a more expensive type of resources such as co-processors to accelerate the data processing [106].

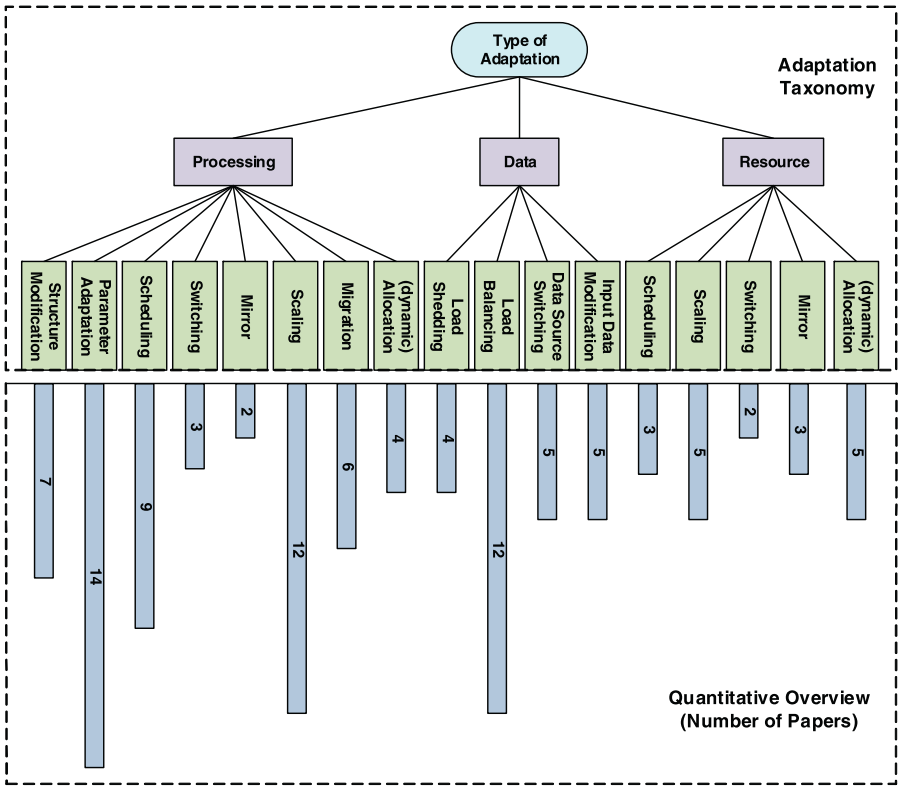

Fig. 3. Taxonomy of identified enactment approaches of adaptation (upper part) as well as their quantitative overview in terms of number of papers (lower part).

- **Resource Scaling**: The approach aims at (automatically) scaling up and down resource usage to cope with the varying processing workloads for achieving optimal resource utilization. This includes studies focusing on elasticity or scalability of processing resources, e.g., scalable join processing [54].
- **Resource Scheduling**: The approach focuses on planning the resource assignment to the processing in advance, typically according to resource constraints and some assignment criteria. For example, Pham et al. [104] schedule the resource to different classes of processing operators based on their priorities.

(b) **Data**: This category reflects adaptations performing some form of modification of the data streams.

- **Input Data Modification**: The approach modifies the structure of the input data streams, e.g., splitting streams for different processing nodes, changing the order of the data items [60] or changing the data size, e.g., the resolution of the video input data [66].
- **Data Source Switching**: An input source is switched to another providing different characteristics of data, e.g., switching from high resolution data to low resolution data in video streaming [65].
- **Load Balancing**: The approach balances the workload among different processing nodes, e.g., redirecting streams from busy processing nodes to idle ones [105] or migrating workload from one processing node to another [16].
- **Load Shedding**: The approach dynamically drops data items to cope with overloaded resources [11].

(c) **Processing:** This category covers adaptation affecting the (data) processing. We use "processing" here as a generic term. It refers to all kinds of processing-related concepts, such as query, operator, processing topology, processing node, processing algorithm and processing task. We identified the following processing-related enactment approaches:

- **(Dynamic) Processing Allocation**: The approach aims at dynamically (re-)allocating processing-related activities such as processing tasks, operators or queries to available resources. This includes studies focusing on (1) the assignment or allocation of the processing tasks to worker nodes [47] and (2) dynamic query/operator placement that determines on which physical machine a logical processing task runs [85]. Typically, the allocation of the processing aims at a certain goal, e.g., achieving an optimal processing allocation in a resource constrained environment.
- **Processing Migration**: The approach focuses on moving processing-related activities such as processing tasks, operators or queries from one processing node to another. Approaches in this category might be enabled by other enactment approaches that involve the migration of the processing, e.g., redistributing queries involving the query migration to achieve load balancing [89].
- **Processing Scaling**: The approach aims at dynamically scaling up and down the processing. This category includes replicating processing operators [90] or reconfiguring the parallelism degree of processing nodes to achieve the processing elasticity [52].
- **Processing Mirror**: The approach runs some processing in parallel, which is typically applied in fault tolerance, e.g., running a processing replica on another machine to enable recovery in case of host failures [48]. Mirroring a logical processing typically implies mirroring or allocating resources but not vice versa.
- **Processing Switching**: The approach switches part of the processing to its alternative or equivalent processing, e.g., switching among different implementation versions of a computation algorithm within a processing node or replacing a processing component with another one [21].
- **Processing Scheduling**: The approach aims at achieving an optimal processing plan for the execution of the processing,

Table 6
Overview of the enactment approaches of adaptation.

Enactment approach of adaptation		Papers
(Dynamic) Resource Allocation		[92,95,111,114,115]
Resource Mirror	Processing-oriented	[48,84]
	Storage-oriented	[48,116]
Resource Switching		[106,112]
Resource Scaling		[12,54,73,81,87]
Resource Scheduling		[12,98,104]
Input Data Modification	Input data compression	[64,66,100,118]
	Input data duplication	[64]
	Data sequence modification	[60]
Data Source Switching		[10,63,65,66,110]
Load Balancing	Load redirection	[30,52,94,101,105]
	Load migration	[16,67,78,83,88,89,117]
Load Shedding		[11,84,104,114]
(Dynamic) Processing Allocation	Dynamic task allocation	[47]
	Dynamic operator/query placement	[13,85,89]
Processing Migration		[16,51,61,68,90,97]
Processing Scaling		[13,14,16,52,62,68,72,80,90,93,102,109]
Processing Mirror		[48,84]
Processing Switching		[21,86,118]
Processing Scheduling		[11,30,59,71,74,75,81,82,98]
Parameter Adaptation	Rate adaptation	[64,69,70,107,108,113]
	Batch size adaptation	[50,53,76,79,91]
	Algorithm-specific	[83,84,116]
Processing Structure Modification	Query graph modification	[13,77,78,103]
	Topology modification	[14,16,96]

e.g., scheduling operators to be processed next based on system conditions [59] or scheduling processing tasks based on their priorities [11].

- **Parameter Adaptation**: The approach aims at tuning the specific parameters or settings of processing algorithms or the overall system, e.g., adapting the buffer/window size [53], adjusting the transmission rate in video stream processing [10] or changing algorithm-specific settings to adapt the behavior of the processing [84].
- **Processing Structure Modification**: The approach modifies the structure of the processing, i.e., the changing of processing-related elements in a query graph or a processing topology. We refer to processing structure modification on a logical level, i.e., the functional rearrangement of processing nodes. This rearrangement can be part of the processing or the entire processing graph. It mainly includes (1) rearranging the processing nodes in the graph, e.g., modifying the query plan to rearrange operator sequences [103]; (2) removing or adding new (not a copy) processing nodes, e.g., removing idle nodes [14]. Note that this does not include the execution (physical) structure of the processing, e.g., replicating the operator instances [109] would change the physical processing execution but not the logical (functional) processing.

The lower part of Fig. 3 depicts the quantitative overview of the identified enactment approaches of adaptation in terms of number of papers (one paper can appear for multiple enactment approaches). Parameter adaptation (14, 13.9%), load balancing (12, 11.9%) and processing scaling (12, 11.9%) are applied by the majority of the studies, followed by processing scheduling (9, 8.9%), processing structure modification (7, 6.9%), etc. The detailed list of papers for each enactment approach is shown in Table 6.

4.3. RQ2: How are they realized?

In this section, we discuss the underlying techniques used to realize the enactment approaches we categorized in Section 4.2. Based on the identified details of techniques as described in Section 3.4.4, we analyze and classify the techniques into different categories. For each technique category we provide representative examples.

First, we discuss the realization techniques for the resource-related enactment approaches:

R1. (Dynamic) Resource Allocation
We identified five studies for this category. Works et al. [114] **allocate available resources to tuples based on their rank** (the importance of tuples) to ensure that critical tuples can be processed with high priority. They use multiple queues to divide tuples into different priority levels. During tuple execution, queues are dynamically allocated to operators for processing based on rank order, i.e., starting from the highest priority queue. Kalim et al. [95] **assign resources to topologies** missing their SLO (Service Level Objective). This reconfiguration is executed periodically by sorting topologies missing their SLOs in descending order of their maximum utility and then greedily picking the head of this sorted queue to allocate resources to. The other three approaches aim at **provisioning optimal server resources** for media streaming systems. Liu et al. [111] focus on a seamless transition of changing the number of channels that are allocated to video streams. For allocating more channels, they evenly assign the video segments of the first channel among the newly added channels and for allocating less channels, they transmit the segments of the channels to be released to the last channel of the remaining ones. To meet the streaming quality, Yuan et al. [115] first collect the surplus of the overall server resources, preallocate them to subsystems and then incrementally adjust the resource allocation to ensure all subsystems reach their server bandwidth demand. Zhao et al. [92] aim at providing the streaming client the best virtual view with current available resources. They first derive the corresponding bit rate level for different synthesized virtual views according to the Quality of Experience (QoE) to be achieved and then allocate the corresponding resource block for the determined bit rate to the user.

R2. Resource Mirror
The resource mirror approach aims at using additional resources to store or process backups of data items or processing states in parallel. Depending on whether the resources are mirrored for

processing or storing, we categorized the extracted techniques into two groups:

- **Processing-oriented**: Standby resources are used to execute the replicated processing in parallel. Heinze et al. [48] execute two processing replicas in parallel on different hosts to enable an immediate hand over in case one of the replicas fails. Also, Hu et al. [84] use neighboring servers to run backup copies of slow tasks in parallel to accelerate the time-critical streaming jobs.
- **Storage-oriented**: Standby resources are used to store the backups such as data items or processing states. The upstream backup presented by Heinze et al. [48] is a mechanism to support the backup of data items, which was also applied in [116]. This technique uses an input queue to cache the unacknowledged items in memory and releases the memory again once the items are acknowledged. Another example presented by [116] is the state backup that uses the local memory to store a copy of the most recent states of operators.

R3. Resource Switching

The resource switching focuses on reconfiguring the type of the resource to be used by selecting from different resource variants with different tradeoffs, e.g., CPU and GPU. Typically they are used to cope with overloaded situations that can be mitigated by a faster but more expensive resource. Li et al. [110] introduced a runtime interface selection to **switch among different network resources** with different bandwidths, e.g., LTE and Wifi, to maximize video quality. Upon the arrival of video packets, the selection manager determines the appropriate interface connecting to a specific type of resource based on monitored metrics such as throughput. Zhang et al. [106] **switch from CPU to GPU** (faster resource) in order to accelerate the video analysis when CPU resources are overloaded and cannot guarantee the required video quality. This is determined based on multiple metrics such as CPU load, memory load and throughput.

R4. Resource Scaling

We extracted five studies that support up- and down-scaling of resources. Lin et al. [54] proposed a bipartite graph structure for join processing to enable efficient resource scaling. The processing units in this graph are organized independent of each other and can be easily added or removed upon resource demands. For up-scaling, new units are allocated and necessary data is moved from other overloaded units to these newly added ones. For down-scaling, data on the units to be removed is first distributed to other running units before these units are released. Floratou et al. [73] provide a policy through a self-regulating system to dynamically provision the topology resources to maximize the overall throughput. They equiped their approach with different resolvers that are used to handle different scaling requests. For example, a scale down resolver would scale down resources of a particular node by decreasing the number of instances that correspond to the node. Once a diagnose for scaling is produced, the corresponding resolver is applied according to the type of the scaling request. Relying on a tier-based server infrastructure, Yin et al. [87] preferentially provision the edge service nodes (SN) that directly serve end users to newly joined clients during scaling. When an edge SN reaches its capacity limit, other edge SNs or other tier peers will be provided for new clients. With the dynamic provision of resources enabled by cloud infrastructure, Wang et al. [12] support adaptively leasing cloud server resources according to the dynamic demands from the user side. When an amount of resources needs to be added for processing, it leases new servers or renews existing ones for a certain duration to meet the capacity needs. Ravindra et al. [62] support elastic scaling in a private/public cloud environment. They adaptively move part of the data or the entire input data stream to the public cloud when the resources of the private cloud are insufficient to maintain the QoS specifications. For example, for up-scaling, the public cloud is initialized with the required number of virtual machines equipped with the stream processing engine and a certain portion of data is sent to the prepared machines.

R5. Resource Scheduling

We identified three studies planning the resource assignment for the processing. Pham et al. [104] proposed a **priority-based scheduling** approach. They consider different classes of QoS as priority levels for continuous queries (CQ) and schedule a higher amount of CPU time to the CQs with higher priority. Moreover, they separate the scheduling into two levels, i.e., first mapping the class priorities to an appropriate amount of CPU time and then scheduling the assigned time to operators. A similar approach was also presented by Bhattacharya et al. [98] for P2P media streaming applications. They use the peer capacity as priority level to schedule the suitable peers for data downloading and uploading. In a cloud environment, Wang et al. [12] proposed a **cloud lease scheduling** approach for live media streaming to handle the diversity of server capacities and lease prices as well as the latency problem of the resource provisioning. They predict the demand of the online population of users, their distributions, etc., and also consider the service availability and streaming quality constraints to determine the starting time and the running duration of the cloud servers to be scheduled.

Now, we discuss the realization techniques of the enactment approaches related to the modification/rearrangement of data streams:

D1. Input Data Modification

Modification of input data can be achieved by changing the data format, the data item order or even the data content. We identified five studies for this category. We classified them as follows:

- **Input data compression**: compresses or packages the original input data into a smaller data size. This is typically applied in video or voice streaming systems. For example, Li et al. [64] apply the compression of a voice stream to change the input data size based on the network capacity. Chen et al. [100] deal with sparse inputs, i.e., containing many zero coefficients, of the Fast Fourier Transform (FFT) algorithm for stream processing. They analyze the input sparsity and the similarity between input samples to adaptively package the input with non-zero coefficients into a small number of bins in order to improve the efficiency of the FFT algorithm.
- **Input data duplication**: duplicates data packets with a certain probability to reduce data loss for lossy wireless links. Li et al. [64] send redundant data packets depending on the duplication ratios.
- **Data sequence modification**: changes data item sequence to improve the processing. The adaptive data acquisition proposed by Lim et al. [60] focuses on changing the order of data streams to be requested from individual sensors to achieve an optimal sequence considering energy overheads on the battery-constrained mobile devices. It also selectively pulls only appropriate subsets of the sensor data streams.

D2. Data Source Switching

We identified five studies aiming at switching among different data source streams. Li et al. [110] switch among different video sources located in different (geographical) places. During switching, they process data from both old and new sources in parallel to reduce the startup delay of the new source. They also consider time constraints to interleave the data delivery of both old and new sources based on their priorities, i.e., executing high-priority data segments as early as possible. A similar approach also in media streaming was presented by Guo et al. [65]. They request the

media server to switch to a source with lower rate streams when the bandwidth drops below a certain threshold, i.e., through a switch message specifying the current stream and the stream to switch to. Lee et al. [66] propose a link-state feedback approach to change the source streams with the desired characteristics, i.e., the size of the video frame data, to meet the present physical transmission rate according to the link state situation. During data transmission, the receiver tracks the link state information and sends it to the transmitter. The transmitter determines the expected data size under the current link quality, e.g., changing to higher resolution data to enhance video quality if the network bandwidth can accommodate.

D3. Load Balancing

Load balancing focuses on the load of the processing systems and (re-)balances the load as needed. We identified twelve studies with the following techniques:

- **Load redirection**: redirects/re-routes the load away from the busy node to achieve a balanced load distribution. This directly happens before the load is allocated to the processing nodes. Balkesen et al. [52] rely on a controlling interface to suspend/resume the data flow and to redirect the stream to different processing nodes based on the monitored statistics of their workloads. Similarly, Schneider et al. [105] reroute tuples to high-capacity connections before a low-capacity connection becomes overloaded. Kuang et al. [94] adjust the load by influencing the load distribution decision, i.e., reducing the chance to forward load to the heavily-loaded processing units or redirecting load to other available processing units.
- **Load migration**: this occurs when the load is already distributed and balancing is demanded to migrate the workload from one processing node to another. Madsen et al. [16] periodically sort processing nodes based on their workloads and migrate the biggest suitable data partition from the node with the highest workload to the one with the lowest workload according to this sorted list. Similarly, Du et al. [83] sort the tasks from fastest to slowest based on their latency criteria and then divide the tasks into two groups of equal size, i.e., faster tasks and slower tasks. They pair up slow and fast tasks and shift a suitable amount of workloads from the slower ones to the faster ones. Vasconcelos et al. [67] organize the system workload into slices representing a group of data items. In overloaded situations, they migrate slices from the overloaded processing nodes to the underloaded ones. During migration, they first pause the processing of the data from the slice to be migrated in the slice-giving node, then duplicate this slice to the slice-taking node and finally synchronize the data items in both nodes to avoid duplicates. Another challenging load migration group of approaches is **the migration involving states**, e.g., migrating load from the stateful operators needs to take care of the accumulated state from the processing related to that load. For example, Wang et al. [78] present a state relocation for multi-way join operators while balancing the load through adding/removing of nodes. During load balancing, this technique evenly (re-)partitions the states into disjoint slices among all nodes and afterwards relocates them to corresponding processing nodes.

D4. Load Shedding

Load shedding is typically applied to drop tuples in overload situations. It saves timeliness-related performance, but sacrifices result accuracy. One way of shedding data is to skip delayed data based on a time interval threshold and to jump to the next interval to continue the stream computation [84]. A more advanced approach is to adaptively determine the number of tuples or even specific types of tuples to be dropped. Kulkarni et al. [11] propose a feedback control approach by first estimating the execution time of queries and then determining which tuples to drop to ensure a certain amount of execution time being reduced. Pham et al. [104] determine load shedding based on the statistics of the system load and the monitored response time of queries. They consider the priority of continuous queries to drop tuples on lower-priority queries. This ensures less data loss on a higher priority query that may contribute for a higher Quality of Data. A similar approach was also applied in [114] to drop the low-priority tuples in case of limited available computation resources.

At last, we discuss the realization techniques for the processing-related enactment approaches:

P1. (Dynamic) Processing Allocation

As introduced in Section 4.2, this category encompasses the dynamic allocation of processing-related activities such as tasks, operators and queries. We identified four studies. Chatzistergiou et al. [47] determine a **dynamic task allocation** to minimize the overall communication cost of the processing. They pair tasks to gain less inter-node communication cost. Moreover, they prioritize the task pairs on a combined metric of their communication cost and processing cost and then co-locate task pairs to nodes based on their node capacity. **Dynamic operator/query placement** is another type of processing allocation. The main idea is to map the operator/query execution into a set of available computing resources according to the overall system requirements. For example, Lakshmanan et al. [85] place dataflow graphs onto a distributed network of machines. They conduct the placement through three stages, i.e., first establishing paths to the destination of a query, then computing the cost of each hypothetical placement, and finally selecting the best placement to execute. They also support incremental updates from the old placement to the new one when a more optimal one is found. Zhou et al. [89] aim at an adaptive query redistribution in reaction to the processing dynamics. They focus on mapping queries to the (physical) network processing graph. To enable this mapping, they employ a hierarchical scheme to construct both query and network graphs. The query distribution starts at its root coordinator and repeatedly performs the mapping to the network graph for each child until all queries are assigned to the processors.

P2. Processing Migration

Processing migration focuses on migrating processing-related activities such as processing tasks, operators and queries from one processing node to another. We identified six studies for this category. Madsen et al. [51] **migrate the processing tasks** of stateful operators among processing nodes. They partition the keys of data items into a set of key groups representing the processing tasks to be migrated. During processing, they also maintain an independent state for each key group. To reduce the migration cost, they apply a check-pointing mechanism that periodically stores the accumulated states in operators. During migration, therefore, it only needs to convert the checkpointing to the most recent state and to replay the queued data items at the new node. Moreover, to achieve low-latency migration, they keep the old operator instance running while initializing the new instance with its checkpointing. When the checkpoint converting is complete, they synchronize the processing so that the new instance continues the computation exactly where the old one stops. This strategy was also applied in [16] for performing migration of window-based operators. Luthra et al. [97] **migrate the operators** to support the transition between operator placement mechanisms adapting to dynamic environments. They proposed two strategies. The first one migrates the operators in a sequential and breadth first manner to their target operator host, i.e., first migrates the leaf operators and then continues level up in the operator graph for their predecessors or dependent operators. During migration, they aim

to capture a minimum state, i.e., an intermediate state, for the target operator host by letting it subscribe to event streams for some time. In contrast, the second one allows the concurrent execution of operator migrations, i.e., the migrations that do not interfere with each other. Matteis et al. [90] directly transmit the keys of data items to the operator to be migrated to based on the reconfiguration message containing a new routing table. For web-related streaming systems, Babazadeh et al. [68] migrate the operators with heavy computation from a mobile device to a desktop or web server machine. This approach does not consider the state of operators. During migration, they create copies of these operators on the new machines and establish connections to other operators based on the given topology structure.

P3. Processing Scaling

We identified twelve studies that support up- and down-scaling of the processing. These studies focus on **replicating operators** or adaptively **configuring the parallelism degree** of the processing nodes or **adjusting the number of parallel tasks**. For example, Hidalgo et al. [13] parallelize an operator execution by creating multiple instances of the operator and then distribute input among these instances. Matteis et al. [90] enable elastic scaling by dynamically increasing or decreasing the number of processing operator replicas. A similar approach was also proposed in [93]. They present two scaling policies to scale-in/out the number of operator replicas. The first one is threshold-based, which supports adding or removing replicas with the threshold of the target utilization level. The second one learns the optimal scaling strategy through experience and direct interaction with the system, which directly optimizes the response time. Jacques-Silva et al. [80] support scaling for streaming joins by adapting the number of parallel tasks of the operator. The processing scaling not only happens on the level of individual processing nodes/operators but also on **a group of several nodes/operators**. To minimize the communication cost across operators, Madsen et al. [16] propose an operator fusion algorithm to group the operators working with the same keys together, which enable the parallelization of a group of operators. They collocate the parallelized instances to reduce bandwidth consumption during data serialization and deserialization.

P4. Processing Mirror

As discussed in Section 4.2, this approach aims at executing the mirror of the processing in parallel. Two studies were identified for this category. Heinze et al. [48] execute two processing replicas in parallel on different hosts to support fault tolerance. Each replica processes all data and sends the results to all successor operators. To avoid duplicated results, they assign an unique sequence id for each data item to enable tracing the data contributed to the result so that they can filter duplicates at the successor operators. Hu et al. [84] run replicas of slow tasks in neighboring agents to maintain low end-to-end delays for time-critical streaming jobs. Each agent maintains a routing table including a neighborhood set and the job master asynchronously replicates the slow tasks to the agent's neighbors.

P5. Processing Switching

Processing switching focuses on switching part of the processing, e.g., the underlying computation in a processing node or a group of processing nodes, to its alternative or equivalent one. We identified three studies for this category. Liu et al. [21] switch among different video encoding segments that contain a sequence of processing components. Upon the switching event, they transform the running configuration from the old segment to the new segment and connect the video source/sink to the input/output port of the new segment respectively. Similarly, Hosseini et al. [118] dynamically switch between different interchangeable quality versions of a point cloud frame. Qian et al. [86] substitute the subgraph of a query with an equivalent one that is guaranteed to produce the same output with the same given input. They take stateful operators into account and record the state in terms of a sequence of the input data, i.e., the dependencies to the output. When it switches to the new subgraph, it first replays the state data to maintain the proper state and then redirects the input streams to the newly placed subgraph.

P6. Processing Scheduling

Scheduling can be performed on different levels of the underlying processing, e.g., scheduling queries, query execution plans, operators, tasks or even tuples. We identified nine studies for this category. For example, Kulkarni et al. [11] aim at **scheduling queries** in a priority scheme, in which queries with shorter execution time are given higher priorities. During scheduling, it first sorts queries based on their execution time and then iteratively schedules queries with their priorities. Chun et al. [75] forecast an update of the data and based on that **schedule a new execution plan** in terms of best actions to be considered. They group all the best actions and pass them to the plan optimizer to evaluate whether they are overdue or duplicated. Afterwards, the accepted best actions are applied to the current execution plan. Backman et al. [59] present a framework to **schedule operators** on the processing nodes. To determine the next operator to be executed on a processing node, it pushes the data of the scheduled operator to the buffer of the respective processing node. Eskandari et al. [71] **schedule tasks** across the processing nodes in a Storm cluster, aiming to minimize the inter-node and inter-process communication. They propose a two-level scheduling approach, i.e., first they partition all tasks into subsets resulting in minimized communication cost, then further partition each subset into the number of parts that is the same to the number of JVMs and finally schedule the subsets to the execution slots. Mencagli et al. [81] dynamically **schedule tuples** to different workers to deal with burstiness of the workload. They set an adaptive threshold for the number of tuples to be emitted to a worker based on the monitored throughput. During scheduling, they prioritize assigning tuples to the currently least loaded worker that has the smallest number of enqueued tuples and use that threshold to determine the worker receiving tuples, e.g., if the threshold is exceeded, they schedule tuples to a newly added worker (created by scaling). Caneill et al. [99] create routing tables at processing operators to co-locate the tuples with correlated keys on the same servers in order to minimize the communication traffic across servers. They first detect the correlations among keys in tuples, then generate new routing tables and finally propagate them to all processing operator instances for reconfiguration.

P7. Parameter Adaptation

For this category, we identified the most studies (14, 13.9%). As indicated in Section 4.2, parameter adaptation aims at modifying specific parameters of a processing task to adjust its behavior. We categorize the effect of these parameters into:

- **Rate adaptation**: adjusts the data rate ingested into streaming systems. Six studies were identified. This is mostly applied in video or voice streaming to guarantee the best possible video/voice quality in response to the varying network conditions. In P2P streaming systems, Birke et al. [107] adapt the delivery rate of data chunks provided by peers by controlling the number of parallel active signaling threads. Pozueco et al. [113] increase or decrease the video bit rates by adding or removing the encoding layers using SVC (Scalable Video Coding) technology to adjust the video quality level. Atzori et al. [108] propose a window-based approach to adjust the source rate for each window of a given size, considering its expected video quality estimated based on the characteristics such as the playback buffer status and the current source rate. Zhang et al. [69] match the data rate to the available

bandwidth in order to minimize latency. They degrade the data quality to provide a lower data rate to ensure bandwidth savings if needed.

- **Batch size adaptation**: adapts the number of tuples to be batched for processing in micro-batch stream processing. Considering the effect of batch size on latency and throughput performance, Das et al. [53] propose a control algorithm to adapt the batch size according to the streaming workloads. It first starts with an initial guess of the batch size and then applies a fixed point iteration method to iteratively search for the batch size that is close to the desired one according to the current workload. Tudoran et al. [91] adaptively select the number of events to batch for cloud-based event streaming systems to maximize the end-to-end delay. They determine the optimal batch size based on CPU usage and latency considering the user-defined maximum acceptable delay as the upper limit. Venkataraman et al. [76] schedule tasks in terms of multiple micro-batches as a group. They adaptively select the group size of the micro-batches in order to bound coordination overheads. When the overhead goes above the upper bound, they increase the group size; otherwise, they decrease the group size.
- **Algorithm-specific**: this categorizes the adaptation by changing algorithm-specific parameters, e.g., setting a new threshold, to adjust the behavior of the processing. For instance, Hu et al. [84] create feedback loops on processing jobs to enable the application-specific control by propagating feedback messages to the system, e.g., a message containing a new job specification. Huang et al. [116] adjust the error threshold parameters to address the tradeoff between performance and accuracy in fault tolerance.

P8. Processing Structure Modification

Processing structure modification focuses on functional rearrangement of the query graphs or the processing topologies. Based on these two groups of structures, we classified the identified seven studies into the following cases:

- **Query graph modification**: this modifies the structure of the query graph, e.g., changing the query plan or rewriting the whole query. Nehme et al. [103] adapt the query plan to be customized for different subsets of data. It first determines the best plan for the current subset of data, then stops the old plan and routes the tuples to the operators in the new plan. Higashino et al. [13] rewrite the query graph by applying rewriting rules to map the old graph to the new one. Esmaili et al. [77] present a query modification framework that supports different query change methods with different levels of correctness and performance guarantees. It equips each query version with a special operator in front of the plan to control the dataflow. When it changes to another query version, it redirects the stream to the new query, stops the old one and at the end merges the output of both query versions into a single stream.
- **Topology modification**: this changes the structure of the processing topology. For example, Madsen et al. [16] split or combine the stateful operator components to minimize inter-component communication. For splitting they partition the state of an individual component. For combining they first pause all relevant instances, then migrate their states to the destination nodes and finally launch the execution of the new combined component. To reduce the reconstruction time for combining, they also support running the old components in parallel during the state migration.Budhkar et al. [96] provided a topology adaptation strategy to reposition peers to replace their ancestors that have less upload capacity. First, the peer sends a request message to its ancestors to collect the list of the replaceable ancestors. The request message consists of the peer's upload capacity to be used to compare with other peers and the time-to-live (TTL) to limit the lifespan of the message. The collected ancestors are eventually ranked based on the evaluation parameters such as upload capacity and propagation delay. The peer replaces the ancestor with the top rank.

4.4. RQ3: What problems are addressed by the publications?

To answer this question, we analyzed data extracted based on **P2**(Problem) property in our data extraction form (see Section 3.4.2). Table 7 illustrates the problem categories (column) in relation to the enactment approaches (row) presented in the publications in terms of the number of paper occurrences. For example, to handle the problem of varying processing workloads (column 4) different enactment approaches were applied in our primary studies. The most common adaptation to address this problem is the processing scaling approach (6 occurrences in column 4, row 12).

The resource bottleneck/fluctuation problem (column 1) was addressed in most of the enactment approaches of our primary studies, e.g., switching data source to a lower resolution data source in video stream processing to handle the bandwidth fluctuations [65]. To meet the timeliness requirement (column 2) such as low response time or low latency, different adaptations were proposed for different processing conditions, e.g., applying the load shedding to skip delayed data to deal with time-critical processing jobs [84] or performing load balancing to adaptively distribute workload among processing nodes to minimize the response time for join processing [78]. It is worth mentioning that we searched "latency" in full-text while identifying the relevant primary studies (see Appendix A.1). This means that "latency" appears anywhere in the study, i.e., it would be collected into the result of *Problem, Goal* and *Evaluation Metrics* if occurs but not limited to any one of them.

While analyzing the identified problems, we observed that runtime adaptation in stream processing is mostly driven by two aspects:

- **Dynamics of the processing**: the dynamic nature of stream or processing (environment) characteristics, e.g., varying input characteristics (column 3) [99], varying processing workloads (column 4) [111] or resource fluctuations (column 1) [21].
- **Performance requirements**: in some cases, the performance issues trigger the adaptation, e.g., the throughput (column 9) and timeliness (column 2) requirement of the data processing [105] and the guarantee of the result quality (column 7) [106].

4.5. RQ4: What are the (measurable) goals aimed in the publications?

For this question, we analyzed the goals extracted based on the data property **P3**(Goal) (see Section 3.4.3). Table 8 illustrates our result in terms of the number of papers in which a specific goal (column) is aimed for a specific enactment approach (row). For example, four papers aimed at achieving low processing overheads by applying a load balancing approach (column 3, row 8).

Overall, we identified six categories based on the extracted goals. Achieving low latency is the goal covered by all identified enactment approaches due to our focus of latency in this SLR. Representative examples are the latency-based load balancing approach [83] applied to reduce the tail latency of stream processing systems and the feedback-controlled load shedding [11] to guarantee the response time. The second largest group is the goal of optimizing resource utilization, followed by the processing overloads and the result quality. Only a few approaches focused on accuracy or throughput as the goal.

As discussed in Section 3.4.3, we also evaluate for each study whether the extracted goals reflect the defined problems (Section 4.4). As we mainly focus on the measurable goals, we compare with the problems only if they indicate performance requirements (measurable),

Table 7
The overview of the problems in relation to the enactment approaches in terms of paper occurrences.

Problem									
Approach	Resource bottlenecks/ fluctuations	Timeliness (Latency)	Varying input characteristics	Varying processing workloads	Varying processing environment	System failures	Result quality	Processing bottlenecks	Throughput
(Dynamic) Resource Allocation	2	1	1	1	2		2		
Resource Mirror		1				2			
Resource Switching	1	1		1	1		1		
Resource Scaling	2	2	2	2	1	1		1	
Resource Scheduling	2	2							
Input Data Modification	3		1		1				
Data Source Switching	4				1				
Load Balancing	2	4	4	2	1		2		1
Load Shedding	1	2	1	1					
(Dynamic) Processing Allocation	1	1	1	1	2				
Processing Migration	2	1	1	1	1	2	1		
Processing Scaling	4	2	3	6	1			2	
Processing Mirror		1				1			
Processing Switching	2			1		1			
Processing Scheduling		2	4	2					
Parameter Adaptation	3	4	3	3	4	2	3		
Processing Structure Modification		2	3		1	1			
Sum	29	26	24	21	16	10	9	3	1

Table 8
The overview of the goals in relation to the enactment approaches in terms of paper occurrences.

Goal						
Approach	Latency	Resource utilization	Processing overloads	Result quality	Accuracy	Throughput
(Dynamic) Resource Allocation	1	3		2		
Resource Mirror	2	1			1	
Resource Switching	1			2		
Resource Scaling	1	1		1		1
Resource Scheduling	1	1	1			
Input Data Modification	1	2		1		
Data Source Switching	3	1		1		
Load Balancing	8	2	4	2		
Load Shedding	3	1			2	
(Dynamic) Processing Allocation	3		2			
Processing Migration	4	2	3			
Processing Scaling	3	2	1			2
Processing Mirror	2	1				
Processing Switching	1	1		1		
Processing Scheduling	5	2	1			
Parameter Adaptation	6	4		3	3	2
Processing Structure Modification	2	3	1			
Sum	47	27	13	13	6	5

i.e., the ones considered in the categories of resource bottlenecks, throughput, timeliness, processing bottlenecks and result quality (see Section 4.4). Overall, 42 (56%) studies meet this comparison condition. Among them, 34.6% (26) defined their goals associated with the problems to be addressed, 10.7% (8) partially reflected their goals to the defined problems and 10.7% (8) has no reflection at all.

4.6. RQ5: How are these approaches evaluated in the publications?

We answer this question in terms of two sub-questions **RQ5.1** and **RQ5.2** (see Table 1). In Section 4.6.1 we discuss the result of the identified evaluation metrics to answer **RQ5.1** and in Section 4.6.2 the evaluation parameters to answer **RQ5.2**.

4.6.1. Evaluation metric

As discussed in Section 3.4.6, we extracted data based on the data property **P6** (Evaluation Metrics) to answer the research question **RQ5.1**. Table 9 shows the extracted evaluation metrics to the enactment approaches with the respective paper occurrences. We identified 9 categories of the evaluated metrics (see columns in Table 9). Except for the general metrics such as latency and throughput, we also identified the adaptation-specific ones:

- **Adaptation impact**: categorizes metrics expressing the impact caused by executing the adaptation, e.g., the number of client redirections and client buffering caused by the stream redirection due to load balancing [52].
- **Adaptation efficiency**: includes metrics measuring how fast the adaptation can be performed, e.g., the complete time for migrating state in the processing migration approach [16] or the settling time for executing the reconfigurations during processing scaling [90].

From our extracted data, the adaptation impact is evaluated for most enactment approaches, followed by latency and resource consumption. This, to some extent, can be interpreted that evaluating the impact of the adaptation on the overall processing is a common perspective to look into. The latency metric indicates the importance of considering the timeliness evaluation in stream processing. Not surprisingly, resource consumption is focused by many adaptations as it is one of the most

Table 9
The overview of the evaluation metrics in relation to the enactment approaches in terms of paper occurrences.

Evaluation Metric									
Approach	Latency	Resource consumption	Throughput	Adaptation impact	Result quality	Processing time	Adaptation efficiency	Processing overloads	Accuracy
(Dynamic) Resource Allocation	1	4	2		2			1	
Resource Mirror	1		1	1			1		1
Resource Switching		1		1	1				
Resource Scaling	1	2	2	1					
Resource Scheduling	2	2		1	1	1			
Input Data Modification		2			3	1			1
Data Source Switching			1	1	3	1	1		
Load Balancing	4	3	6	4	1	3	1	4	
Load Shedding	2	2	1	2					1
(Dynamic) Processing Allocation	1		2	1		2			
Processing Migration	4	2		1		1	3	1	
Processing Scaling	6	5	4	3		1	1		1
Processing Mirror	1			1			1		
Processing Switching	1			2	1				
Processing Scheduling	2	2	3	2	1	2	2	1	
Parameter Adaptation	8	2	4	1	4		1	1	2
Processing Structure Modification	2	2	1	1	1	2	1	1	1
Sum	36	29	27	23	18	14	12	9	7

common problems to be addressed in adaptive stream processing (see Section 4.4). Another interesting aspect is that throughput is far more commonly considered as evaluation metric than as problems or goals (see Tables 7 and 8).

As mentioned in Section 3.4.3, we also evaluate whether the actual evaluated metrics reflect the promised goals in Section 4.5 for each paper. Overall, 62 (82.7%) studies meet the comparison condition that both measurable goals and evaluation metrics are determined in the extracted data. 40 (53.3%) studies evaluated all their defined goals, 14 (18.7%) partially evaluated their promised goals and 8 (10.7%) studies did not reflect any goals.

4.6.2. Evaluation parameter

We analyzed the extracted data based on the data property **P7** (Evaluation Parameter) (see Section 3.4.7) to answer the research question **RQ5.2**. The analyzed result is illustrated in Table 10. Overall, we identified nine different types of parameters that present different dynamic environments for evaluating the enactment approaches. Among them, changing input rate (column 1) is used by most enactment approaches for triggering adaptation, followed by varying workload (column 2). For example, Zhou et al. [89] increase or decrease the stream rates to simulate load imbalances to evaluate their load balancing approach. Claypool et al. [74] add extra instructions for making routing decisions to increase the workload for evaluating their processing scheduling approach.

There are many enactment approaches evaluated with varying resource capacity (column 3) by changing the capacity settings of available resources, e.g., varying CPU utilization [83] and varying transmission bandwidth [21]. Providing different scales of processing clusters (column 4), e.g., in terms of the number of worker nodes in a cluster, is also commonly applied to evaluate adaptations, e.g., varying numbers of peers simultaneously attempting to join/depart the system to evaluate the load balancing approach [101]. The varying window/batch size (column 5) is often used to evaluate window-based operations [53,59]. There are some approach-specific parameters (column 7) such as the varying join selectivity in multi-way join processing [74] and varying locality (percentage of co-located tuples with the same key) [99] used for evaluating the processing scheduling approach. Other interesting parameters are the data sample (column 6) providing different data distributions [103] or changes in stream properties [120] as well as the query characteristics (column 12) providing different types of queries with different predicates [60]. It is worth mentioning that the approaches involving state migration, e.g., the processing migration, commonly adopt the state size (column 8) as the parameter for its evaluation [16].

5. Threats to validity

Below we discuss the potential threats to the validity of our SLR.

Identification of primary studies. Our search strategy aimed at identifying as many primary studies as possible related to runtime adaptation in stream processing (see Section 3.1). However, finding all relevant studies is always the major problem in systematic literature reviews. To minimize this issue, we applied the automated search on ACM with the “guide to computing literature” database to consider not only all articles from ACM but also all possible citations and links to all other publishers in the field of computing. Additionally, we also searched on ScienceDirect to retrieve articles over more possible publishers especially journals.

Also, the search result depends much on how precise the search engines can handle search strings. We have to face the fact that most search engines such as ACM and ScienceDirect often have limited functionalities on retrieving papers based on a given search string. In particular, they have some limitations regarding length and complexity [121], e.g, the ACM search engine does not support complex logical combination of search terms [122]. In our search, we followed the guidelines of both libraries for constructing a valid search string (see Appendix A).

Regarding the search terms, we do not include the term “optimization” as it would result in a far broader range of irrelevant papers, i.e., the general meaning of optimization but no adaptation at all. This may miss some relevant papers that use the term “optimization” as a synonym for “adaptation” or “reconfiguration”. However, we applied the pilot search to refine the search string for identifying more precise results and also used a test sample of known relevant papers to critically validate the completeness of our search. The result gives us confidence that we reached our completeness target (see Section 3.1).

Selection and data extraction consistency. For the study selection, we applied a multi-stage process in which the inclusion and exclusion criteria were applied incrementally (see Section 3.2). However, assuming a common understanding among authors about the criteria and the selection process can cause risks. We realized this in the very beginning and mitigated this threat by having discussion meetings for each selection stage to ensure all authors have a consistent understanding of the protocol.

Table 10
The overview of the evaluation parameters in relation to the enactment approaches in terms of paper occurrences.

Evaluation Parameter												
Approach	Input rate	Workload	Resource capacity	Scale of processing cluster	Window/batch size	Data sample	Approach-specific	State size	Tuple size	Parallelism	System failures	Query characteristics
(Dynamic) Resource Allocation	1	3		1								
Resource Mirror	1			1	1		1				1	
Resource Switching									1			
Resource Scaling	2	2	1									
Resource Scheduling	1		1	1								
Input Data Modification			1		1	1						1
Data Source Switching		1	3	1								
Load Balancing	3	4	1	3		1	2	1				
Load Shedding	1	2		1								
(Dynamic) Processing Allocation	2	2	1	2								
Processing Migration	1	1	1		1			3	1			
Processing Scaling	3	4			2	1		1		1		
Processing Mirror	1			1			1					
Processing Switching		1	1									
Processing Scheduling	3	3			2		2		1	1		
Parameter Adaptation	7	1	5		3	1					1	
Processing Structure Modification		1		1	1	2	1	1				
Sum	26	25	15	12	11	6	7	6	3	2	2	1

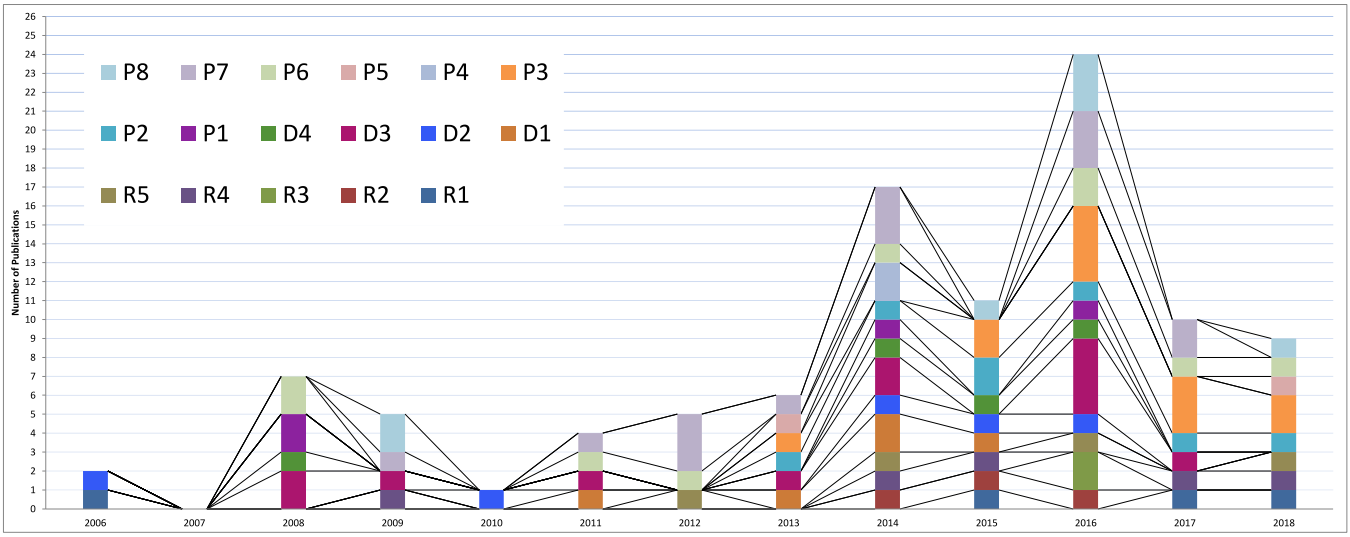

Fig. 4. Overview of how the number of publications for each enactment approach changed over time.

During data extraction, we excluded some papers that insufficiently described the actual enactment approach based on the exclusion criterion EC10. To avoid bias on understanding of the "insufficient information", we discussed all exclusion decisions and rejected only papers for which all authors came to the same conclusion.

The whole data extraction was done by one (the first) author and crossed-checked by one of the other authors in order to minimize the extraction bias. For cross-checking each paper, the authors added comments on the extracted data in terms of five categories: *clarification* if something to be explained from the extracted data; *addition* if something to be added; *disagreement* if something to be disagreed; *classification* if comments are about the adaptation classification; *guidelines* if comments are related to the guidelines. The cross-checking result shows 51 clarification, 65 addition, 20 disagreement, 13 classification and 3 guideline issues. After the discussion meeting, 83 of these issues were finally taken into account while 69 were resolved without any changes. Although we carefully cross-checked the data, there might be still bias. For example, for some data properties such as problems we may not be able to determine whether the extracted data is for the adaptation or for the overall approach in the paper, or even for a general research field.

6. Conclusion

In this paper, we presented a systematic literature review that investigates runtime enactment of adaptation in data stream processing from a latency perspective. The aim is to identify and characterize enactment approaches of adaptation applied in stream processing that take into account latency. Our results cover:

- **Research context**: we discussed the research context of the included papers from two different levels of perspectives (Section 4.1.3). For the high-level context, overall, the majority of the papers (54, 69.2%) are in the general context of data stream processing. For the specific context, we identified four categories and for each of them we provided a detailed discussion about their concrete context.
- **Enactment approach:** we identified 17 enactment approaches (**RQ1**) and provided a taxonomy classifying them from the perspective of resource, data, and processing (see Section 4.2). The most commonly applied enactment approaches are Parameter Adaptation (14, 13.9%), Load Balancing (12, 11.9%) and Processing Scaling (12, 11.9%).
- **Enactment technique:** for each enactment approach, we identified the enactment realization techniques (**RQ2**) to gain insight into the diverse realizations of adaptation in different contexts (see Section 4.3).
- **Purpose of adaptation:** we extracted the problems (**RQ3**) to be addressed by adaptations in the included studies to analyze the purpose of each identified enactment approach (see Section 4.4). In addition to the Timeliness (Latency) problem (26, 18.7%), Resource Bottleneck/Fluctuation (29, 20.9%) and Varying Input Characteristics (24, 17.3%) are the most commonly addressed problems.
- **Goal of adaptation:** we investigated the goals (**RQ4**) of the identified enactment approaches (see Section 4.5). In addition to Latency (47, 42.3%), Resource Utilization (27, 24.3%) is also a major goal that most of adaptations focus on.
- **Evaluation of adaptation:** we dived into the evaluation (**RQ5**) of enactment approaches to capture their evaluation metrics (**RQ5.1**) and their parameters (**RQ5.2**) that are used to trigger the adaptation (see Section 4.6). In addition to Latency (36, 20.6%), Resource Consumption (29, 16.6%), Throughput (27, 15.4%) and Adaptation Impact (23, 13.1%) are the most common metrics evaluated for enactment approaches. Input Rate (26, 22.4%) and Workload (25, 21.6%) are the most commonly used parameters to evaluate the enactment of adaptation.
- **Latency Implication Category**: we provided an overview of latency implication categories (Section 4.1.4). Overall, 37 (49.3%) papers regarded latency as their main focus. For the rest of them, 31 (41.3%) discussed latency and 7 (9.4%) evaluated latency.

As discussed above, parameter adaptation approach has the most paper occurrences compared to others (see Fig. 3). This is followed by load balancing and processing scaling. We can conclude that these approaches are commonly applied in the field of stream processing for the enactment of adaptation from 2006 to 2018. However, this accumulated quantitative overview does not represent a trend in the research. In order to derive a research trend, we provide an overview of the number of papers per enactment approach over the years in Fig. 4. Unfortunately it is not possible to identify a clear trend per approach. For example, for the parameter adaptation approach (P7) there are zero, three, two and zero papers from 2015, 2016, 2017 and 2018, respectively. It seems that the numbers are just too small per approach to show meaningful trends. On a summary level, we can see clearly that the trend is overall positive, with a strong peak in 2016[4]. We can also observe that even

[4] Note, that the column for 2018 only represents the numbers until November 2018. So it will probably be higher for the full year than 2017.

```
("stream processing" OR "continuous query" OR "stream-based system"
    OR "data stream system" OR "streaming system" OR "complex
    event processing") AND ("adaptation" OR "reconfiguration") AND
    content.ftsec:("latency" OR "response time")
```

Listing 1. Search string for ACM.

```
tak("stream processing" OR "continuous query" OR "stream-based
    system" OR "data stream system" OR "streaming system" OR "
    complex event processing") AND tak("adaptation" OR "
    reconfiguration") AND ("latency" OR "response time")
```

Listing 2. Search string for ScienceDirect.

the earliest approaches that we identified do still appear very recently. On the other hand, over time, new approaches were presented in literature, typically reappearing up to now. Thus, we expect a similar trend for the future further approaches that we have not identified so far that-once-existing-trend to reappear over time. Overall, we also expect an increased importance of such approaches due to the shift towards more flexibly adaptive systems.

Appendix A. Search Strings

Here, we present the actual search strings we used to conduct the paper search in ACM and ScienceDirect digital libraries. Different search engines have different query syntax. We adjusted the search strings according to their syntax requirements.

Appendix A.1. ACM

For ACM, we used the *"Advanced Search"* functionality to directly enter the search query. Further, we selected the search option *"The ACM Guide to Computing Literature"*. Listing 1 shows the actual search string for ACM. As explained in Section 3.1, we conducted a pilot search to refine the search terms. During the pilot search on ACM we realized that searching all terms in full text returns too many irrelevant results, e.g., the terms only appear in the references. In order to obtain a more precise search result, i.e., identifying as many as possible relevant papers but avoiding the irrelevant ones, we decided to search the "population" and "intervention" in title, abstract and keywords and the "outcome" in full text (see Table 2). In ACM, the default option *"Any field"* is the basic search that focuses on the search fields of title, abstract and keywords. To search in full text, we use the field code "content.ftsec".

Appendix A.2. ScienceDirect

For ScienceDirect, we used the *"Expert Search"* functionality to directly enter the search query. We selected the options of journals, books and all. Further, to search in the related science field, we selected the *"Computer Science"* filter. Listing 2 shows the actual search string for ScienceDirect. In ScienceDirect, the default option searches in full text and we need to use the field code "tak" in order to search in the fields of **t**itle, **a**bstract and **k**eywords.

References

[1] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly Media, Inc., 2017.

[2] S. Chakravarthy, Q. Jiang, Stream Data Processing: A Quality of Service Perspective: Modeling, Scheduling, Load Shedding, and Complex Event Processing, vol. 36, Springer Science & Business Media, 2009.

[3] H.C. Andrade, B. Gedik, D.S. Turaga, Fundamentals of Stream Processing: Application Design, Systems, and Analytics, Cambridge University Press, 2014.

[4] Apache Storm. Distributed and fault-tolerant realtime computation, (http://storm.apache.org/), Accessed: 2017-12-20.

[5] Apache Spark. Lightning-fast cluster computing, (https://spark.apache.org/), Accessed: 2017-12-20.

[6] D.J. Abadi, Y. Ahmad, M. Balazinska, et al., The design of the borealis stream processing engine, in: CIDR, vol. 5, 2005, pp. 277–289.

[7] B. Gedik, H. Andrade, K.-L. Wu, et al., SPADE: the system S declarative stream processing engine, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2008, pp. 1123–1134.

[8] Y. Wei, S.H. Son, J.A. Stankovic, RTSTREAM: real-time query processing for data streams, in: Object and Component-Oriented Real-Time Distributed Computing, ISORC'06, 2006, pp. 10–pp.

[9] M. Vlachos, K.-L. Wu, S.-K. Chen, S.Y. Philip, Correlating burst events on streaming stock market data, Data Min. Knowl. Discov. 16 (1) (2008) 109–133.

[10] A. Hamza, M. Hefeeda, Adaptive streaming of interactive free viewpoint videos to heterogeneous clients, in: Proceedings of the 7th International Conference on Multimedia Systems, in: MMSys'16, 2016, pp. 10:1–10:12.

[11] D. Kulkarni, C.V. Ravishankar, M. Cherniack, Real-time, load-adaptive processing of continuous queries over data streams, in: Proceedings of the Second International Conference on Distributed Event-based Systems, in: DEBS'08, 2008, pp. 277–288.

[12] F. Wang, J. Liu, M. Chen, H. Wang, Migration towards cloud-assisted live media streaming, IEEE/ACM Trans. Netw. 24 (1) (2016) 272–282.

[13] W.A. Higashino, C. Eichler, M.A.M. Capretz, L.F. Bittencourt, T. Monteil, Attributed graph rewriting for complex event processing self-management, ACM Trans. Auton. Adapt. Syst. 11 (3) (2016) 19:1–19:39.

[14] Y. Su, F. Shi, S. Talpur, Y. Wang, S. Hu, J. Wei, Achieving self-aware parallelism in stream programs, Cluster Comput. 18 (2) (2015) 949–962.

[15] D.G.D.L. Iglesia, D. Weyns, MAPE-K formal templates to rigorously design behaviors for self-adaptive systems, ACM Trans. Auton. Adapt. Syst. (TAAS) 10 (3) (2015) 15.

[16] K.G.S. Madsen, Y. Zhou, L. Su, Enorm: Efficient window-based computation in large-scale distributed stream processing systems, in: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, in: DEBS'16, 2016, pp. 37–48.

[17] H. Eichelberger, C. Qin, K. Schmid, C. Niederée, Adaptive application performance management for big data stream processing, Softwaretechnik Trends 35 (3) (2015) 35–37.

[18] C. Qin, H. Eichelberger, Impact-minimizing runtime switching of distributed stream processing algorithms., EDBT/ICDT Workshops, 2016.

[19] J. Gama, M.M. Gaber, Learning From Data Streams: Processing Techniques in Sensor Networks, Springer, 2007.

[20] J. Smailović, M. Grčar, N. Lavrač, M. Žnidaršič, Stream-based active learning for sentiment analysis in the financial domain, Inf. Sci. 285 (2014) 181–203.

[21] Y. Liu, R. Meier, AdaptStream: towards achieving fluidity in adaptive stream-based systems, in: Proceedings of the 2011 ACM Symposium on Applied Computing, in: SAC '11, 2011, pp. 217–223.

[22] G. Cugola, A. Margara, Processing flows of information: from data stream to complex event processing, ACM Comput. Surv. (CSUR) 44 (3) (2012) 15.

[23] H.-S. Lim, J.-G. Lee, M.-J. Lee, K.-Y. Whang, I.-Y. Song, Continuous query processing in data streams using duality of data and queries, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006, pp. 313–324.

[24] D. Anicic, S. Rudolph, P. Fodor, N. Stojanovic, Stream reasoning and complex event processing in ETALIS, Semant. Web 3 (4) (2012) 397–407.

[25] D.J. Abadi, D. Carney, U. Çetintemel, et al., Aurora: a new model and architecture for data stream management, Int. J. Very Large Data Bases (VLDB) 12 (2003) 120–139.

[26] S. Chandrasekaran, O. Cooper, A. Deshpande, et al., TelegraphCQ: continuous dataflow processing, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003. 668–668

[27] A.J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W.M. White, et al., Cayuga: a general purpose event monitoring system, in: CIDR, vol. 7, 2007, pp. 412–422.

[28] E. Wu, Y. Diao, S. Rizvi, High-performance complex event processing over streams, in: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, 2006, pp. 407–418.

[29] R. Adaikkalavan, S. Chakravarthy, SnoopIB: Interval-based event specification and detection for active databases, in: ADBIS, Springer, 2003, pp. 190–204.

[30] Y. Zhang, Q. Liu, S. Klasky, M. Wolf, K. Schwan, G. Eisenhauer, J. Choi, N. Podhorszki, Active workflow system for near real-time extreme-scale science, in: Proceedings of the First Workshop on Parallel Programming for Analytics Applications, in: PPAA'14, 2014, pp. 53–62.

[31] R. Stephens, A survey of stream processing, Acta Inf. 34 (7) (1997) 491–541.
[32] D. Kossmann, The state of the art in distributed query processing, ACM Comput. Surv. (CSUR) 32 (4) (2000) 422–469.
[33] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, R. Grimm, A catalog of stream processing optimizations, ACM Comput. Surv. (CSUR) 46 (4) (2014) 46.
[34] M. Kremer, J. Gryz, A Survey of Query Optimization in Parallel Databases, Rapport Technique, 1999.
[35] A. Aljanaby, E. Abuelrub, M. Odeh, A survey of distributed query optimization, Int. Arab J. Inf. Technol. vol. 2 (1) (2005) 48–57.
[36] M. Jarke, J. Koch, Query optimization in database systems, ACM Comput. Surv. (CSUR) 16 (2) (1984) 111–152.
[37] A. Gounaris, N. Paton, A. Fernandes, R. Sakellariou, Adaptive query processing: a survey, Adv. Databases (2002) 882–940.
[38] A. Deshpande, Z. Ives, V. Raman, et al., Adaptive query processing, Found. Trends® Databases 1 (1) (2007) 1–140.
[39] S. Mahdavi-Hezavehi, V.H. Durelli, D. Weyns, P. Avgeriou, A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems, Inf. Softw. Technol. 90 (2017) 1–26.
[40] Z. Yang, Z. Li, Z. Jin, Y. Chen, A systematic literature review of requirements modeling and analysis for self-adaptive systems, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2014, pp. 55–71.
[41] M. Becker, M. Luckey, S. Becker, Model-driven performance engineering of self--adaptive systems: a survey, in: Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures, 2012, pp. 117–122.
[42] C. Krupitzer, F.M. Roth, S. VanSyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive Mob. Comput. 17 (2015) 184–206.
[43] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Trans. Auton. Adapt. Syst. (TAAS) 4 (2) (2009) 14.
[44] F.D. Macías-Escrivá, R. Haber, R. Del Toro, V. Hernandez, Self-adaptive systems: a survey of current approaches, research challenges and applications, Expert Syst. Appl. 40 (18) (2013) 7267–7279.
[45] B.A. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, vol. 4, CRC Press, 2015.
[46] B.A. Kitchenham, S.M. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical report, Ver. 2.3 EBSE Technical Report, sn, 2007.
[47] A. Chatzistergiou, S.D. Viglas, Fast heuristics for near-optimal task allocation in data stream processing over clusters, in: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, in: CIKM'14, 2014, pp. 1579–1588.
[48] T. Heinze, M. Zia, R. Krahn, Z. Jerzak, C. Fetzer, An adaptive replication scheme for elastic data stream processing systems, in: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, in: DEBS'15, 2015, pp. 150–161.
[49] Y. Ji, H. Zhou, Z. Jerzak, A. Nica, G. Hackenbroich, C. Fetzer, Quality-driven continuous query execution over out-of-order data streams, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 889–894.
[50] B. Lohrmann, D. Warneke, O. Kao, Massively-parallel stream processing under QoS constraints with Nephele, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2012, pp. 271–282.
[51] K.G.S. Madsen, Y. Zhou, Dynamic resource management in a massively parallel stream processing engine, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, in: CIKM'15, 2015, pp. 13–22.
[52] C. Balkesen, N. Tatbul, M.T. Özsu, Adaptive input admission and management for parallel stream processing, in: Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, in: DEBS'13, 2013, pp. 15–26.
[53] T. Das, Y. Zhong, I. Stoica, S. Shenker, Adaptive stream processing using dynamic batch sizing, in: Proceedings of the ACM Symposium on Cloud Computing, in: SOCC'14, 2014, pp. 16:1–16:13.
[54] Q. Lin, B.C. Ooi, Z. Wang, C. Yu, Scalable distributed stream join processing, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, in: SIGMOD'15, 2015, pp. 811–825.
[55] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, G. Antoniu, JetStream: Enabling high performance event streaming across cloud data-centers, in: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, 2014, pp. 23–34.
[56] C. Liang, X. Huang, SmartCell: an energy efficient coarse-grained reconfigurable architecture for stream-based applications, EURASIP J. Embedded Syst. 2009 (1) (2009) 518659.
[57] M. Staples, M. Niazi, Experiences using systematic review guidelines, J. Syst. Softw. 80 (9) (2007) 1425–1437.
[58] T. Dybå, T. Dingsøyr, Strength of evidence in systematic reviews in software engineering, in: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 178–187.
[59] N. Backman, K. Pattabiraman, R. Fonseca, U. Cetintemel, C-MR: continuously executing MapReduce workflows on multi-core processors, in: Proceedings of Third International Workshop on MapReduce and Its Applications Date, in: MapReduce'12, 2012, pp. 1–8.
[60] L. Lim, A. Misra, T. Mo, Adaptive data acquisition strategies for energy-efficient, smartphone-based, continuous processing of sensor streams, Distrib. Parallel Databases 31 (2) (2013) 321–351.
[61] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, U. Ramachandran, MCEP: a mobility-aware complex event processing system, ACM Trans. Internet Technol. (TOIT) 14 (1) (2014) 6.
[62] S. Ravindra, M. Dayarathna, S. Jayasena, Latency aware elastic switching-based stream processing over compressed data streams, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, in: ICPE '17, 2017, pp. 91–102.
[63] A. Hamza, M. Hefeeda, A dash-based free viewpoint video streaming system, in: Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, in: NOSSDAV'14, 2014, pp. 55:55–55:60.
[64] L. Li, G. Xing, L. Sun, Y. Liu, A quality-aware voice streaming system for wireless sensor networks, ACM Trans. Sen. Netw. 10 (4) (2014) 61:1–61:25.
[65] L. Guo, E. Tan, S. Chen, Z. Xiao, O. Spatscheck, X. Zhang, Delving into internet streaming media delivery: a quality and resource utilization perspective, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, in: IMC'06, 2006, pp. 217–230.
[66] S.K. Lee, S. Yoo, J. Jung, H. Kim, J. Ryoo, Link-aware reconfigurable point-to-point video streaming for mobile devices, ACM Trans. Multimedia Comput. Commun. Appl. 12 (1) (2015) 9:1–9:25.
[67] R.O. Vasconcelos, M. Endler, B.d.T.P. Gomes, F. José da Silva e Silva, Design and evaluation of an autonomous load balancing system for mobile data stream processing based on a data centric publish subscribe approach, Int. J. Adapt. Resilient Auton. Syst. 5 (3) (2014) 1–19.
[68] M. Babazadeh, A. Gallidabino, C. Pautasso, Liquid stream processing across web browsers and web servers, in: Proceedings of the 15th International Conference on Engineering the Web in the Big Data Era - Volume 9114, in: ICWE'15, 2015, pp. 24–33.
[69] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, E.A. Lee, AWStream: adaptive wide-area streaming analytics, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, in: SIGCOMM '18, 2018, pp. 236–252.
[70] Y. Simmhan, B. Cao, M. Giakkoupis, V.K. Prasanna, Adaptive rate stream processing for smart grid applications on clouds, in: Proceedings of the 2Nd International Workshop on Scientific Cloud Computing, in: ScienceCloud'11, 2011, pp. 33–38.
[71] L. Eskandari, Z. Huang, D. Eyers, P-Scheduler: adaptive hierarchical scheduling in apache storm, in: Proceedings of the Australasian Computer Science Week Multiconference, in: ACSW'16, 2016, pp. 26:1–26:10.
[72] R. Huang, D. Sun, Analysing and evaluating topology structure of online application in big data stream computing environment, Int. J. Wire. Mob. Comput. 10 (4) (2016) 317–324.
[73] A. Floratou, A. Agrawal, B. Graham, S. Rao, K. Ramasamy, Dhalion: self-regulating stream processing in heron, Proc. VLDB Endow. 10 (12) (2017) 1825–1836.
[74] K. Claypool, M. Claypool, Teddies: Trained eddies for reactive stream processing, in: Proceedings of the 13th International Conference on Database Systems for Advanced Applications, in: DASFAA'08, 2008, pp. 220–234.
[75] S. Chun, J. Jung, S. Seo, W. Ro, K.-H. Lee, An adaptive plan-based approach to integrating semantic streams with remote RDF data, J. Inf. Sci. 43 (6) (2017) 852–865.
[76] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M.J. Franklin, B. Recht, I. Stoica, Drizzle: fast and adaptable stream processing at scale, in: Proceedings of the 26th Symposium on Operating Systems Principles, in: SOSP '17, 2017, pp. 374–389.
[77] K. Sheykh Esmaili, T. Sanamrad, P.M. Fischer, N. Tatbul, Changing flights in mid-air: a model for safely modifying continuous queries, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, in: SIGMOD'11, 2011, pp. 613–624.
[78] S. Wang, E. Rundensteiner, Scalable stream join processing with expensive predicates: workload distribution and adaptation by time-slicing, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, in: EDBT'09, 2009, pp. 299–310.
[79] J. Wu, K.-L. Tan, Y. Zhou, Data-driven memory management for stream join, Inf. Syst. 34 (4–5) (2009) 454–467.
[80] G. Jacques-Silva, R. Lei, L. Cheng, G.J. Chen, K. Ching, T. Hu, Y. Mei, K. Wilfong, R. Shetty, S. Yilmaz, A. Banerjee, B. Heintz, S. Iyer, A. Jaiswal, Providing streaming joins as a service at facebook, Proc. VLDB Endow. 11 (12) (2018) 1809–1821.
[81] G. Mencagli, M. Torquati, M. Danelutto, Elastic-PPQ: a two-level autonomic system for spatial preference query processing over dynamic data streams, Future Gener. Comput. Syst. 79 (2018) 862–877.
[82] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: Proceedings of the 10th International Conference on The Semantic Web - Volume Part I, in: ISWC'11, 2011, pp. 370–388.
[83] G. Du, I. Gupta, New techniques to curtail the tail latency in stream processing systems, in: Proceedings of the 4th Workshop on Distributed Cloud Computing, in: DCC'16, 2016, pp. 7:1–7:6.
[84] L. Hu, K. Schwan, H. Amur, X. Chen, ELF: efficient lightweight fast stream processing at scale, in: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, in: USENIX ATC'14, 2014, pp. 25–36.
[85] G.T. Lakshmanan, R.E. Strom, Biologically-inspired distributed middleware management for stream processing systems, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, in: Middleware'08, 2008, pp. 223–242.
[86] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, Z. Zhang, TimeStream: reliable stream computation in the cloud, in: Proceedings of the 8th ACM European Conference on Computer Systems, in: EuroSys'13, 2013, pp. 1–14.

[87] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, B. Li, Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky, in: Proceedings of the 17th ACM International Conference on Multimedia, in: MM'09, 2009, pp. 25–34.
[88] N. Zacheilas, N. Zygouras, N. Panagiotou, V. Kalogeraki, D. Gunopulos, Dynamic load balancing techniques for distributed complex event processing systems, in: Proceedings of the 16th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems - Volume 9687, 2016, pp. 174–188.
[89] Y. Zhou, K. Aberer, K.-L. Tan, Toward massive query optimization in large-scale distributed stream systems, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, in: Middleware'08, 2008, pp. 326–345.
[90] T. De Matteis, G. Mencagli, Proactive elasticity and energy awareness in data stream processing, J. Syst. Softw. 127 (2017) 302–319.
[91] R. Tudoran, A. Costan, O. Nano, I. Santos, H. Soncu, G. Antoniu, JetStream: enabling high throughput live event streaming on multi-site clouds, Future Gener. Comput. Syst. 54 (2016) 274–291.
[92] M. Zhao, X. Gong, J. Liang, W. Wang, X. Que, Y. Guo, S. Cheng, QoE-driven optimization for cloud-assisted dash-based scalable interactive multiview video streaming over wireless network, Image Commun. 57 (C) (2017) 157–172.
[93] V. Cardellini, F.L. Presti, M. Nardelli, G.R. Russo, Decentralized self-adaptation for elastic data stream processing, Future Gener. Comput. Syst. 87 (2018) 171–185.
[94] J. Kuang, L. Bhuyan, H. Xie, D. Guo, E-AHRW: an energy-efficient adaptive hash scheduler for stream processing on multi-core servers, in: Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems, in: ANCS'11, 2011, pp. 45–56.
[95] F. Kalim, L. Xu, S. Bathey, R. Meherwal, I. Gupta, Henge: intent-driven multi-tenant stream processing, in: Proceedings of the ACM Symposium on Cloud Computing, in: SoCC '18, 2018, pp. 249–262.
[96] S. Budhkar, V. Tamarapalli, Delay management in mesh-based p2p live streaming using a three-stage peer selection strategy, J. Netw. Syst. Manage. 26 (2) (2018) 401–425.
[97] M. Luthra, B. Koldehofe, P. Weisenburger, G. Salvaneschi, R. Arif, TCEP: Adapting to dynamic user environments by enabling transitions between operator placement mechanisms, in: Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems, in: DEBS '18, 2018, pp. 136–147.
[98] A. Bhattacharya, Z. Yang, D. Pan, Query adaptation techniques in temporal-DHT for P2P media streaming applications, Int. J. Multimed. Data Eng. Manag. 3 (3) (2012) 45–65.
[99] M. Caneill, A. El Rheddane, V. Leroy, N. De Palma, Locality-aware routing in stateful streaming applications, in: Proceedings of the 17th International Middleware Conference, in: Middleware'16, 2016, pp. 4:1–4:13.
[100] S. Chen, X. Li, Input-adaptive parallel sparse fast fourier transform for stream processing, in: Proceedings of the 28th ACM International Conference on Supercomputing, in: ICS'14, 2014, pp. 93–102.
[101] H.C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, M. Dahlin, FlightPath: Obedience vs. choice in cooperative services, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, in: OSDI'08, 2008, pp. 355–368.
[102] G. Mencagli, A game-theoretic approach for elastic distributed data stream processing, ACM Trans. Auton. Adapt. Syst. 11 (2) (2016) 13:1–13:34.
[103] R.V. Nehme, E.A. Rundensteiner, E. Bertino, Self-tuning query mesh for adaptive multi-route query processing, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, in: EDBT'09, 2009, pp. 803–814.
[104] T.N. Pham, P.K. Chrysanthis, A. Labrinidis, Avoiding class warfare: managing continuous queries with differentiated classes of service, VLDB J. 25 (2) (2016) 197–221.
[105] S. Schneider, J. Wolf, K. Hildrum, R. Khandekar, K.-L. Wu, Dynamic load balancing for ordered data-parallel regions in distributed streaming systems, in: Proceedings of the 17th International Middleware Conference, in: Middleware'16, 2016, pp. 21:1–21:14.
[106] W. Zhang, P. Duan, X. Xie, F. Xia, Q. Lu, X. Liu, J. Zhou, QoS4IVSaaS: a QoS management framework for intelligent video surveillance as a service, Pers. Ubiquitous Comput. 20 (5) (2016) 795–808.
[107] R. Birke, C. Kiraly, E. Leonardi, M. Mellia, M. Meo, S. Traverso, A delay-based aggregate rate control for P2P streaming systems, Comput. Commun. 35 (18) (2012) 2237–2244.
[108] L. Atzori, A. Floris, G. Ginesu, D. Giusto, Streaming video over wireless channels: exploiting reduced-reference quality estimation at the user-side, Signal Process. Image Commun. 27 (10) (2012) 1049–1065.
[109] N. Hidalgo, D. Wladdimiro, E. Rosas, Self-adaptive processing graph with operator fission for elastic stream processing, J. Syst. Softw. 127 (2017) 205–216.
[110] Z. Li, J. Cao, G. Chen, Y. Liu, On the source switching problem of peer-to-peer streaming, J. Parallel Distrib. Comput. 70 (5) (2010) 537–546.
[111] Y. Liu, S. Yu, J. Zhou, Adaptive segment-based patching scheme for video streaming delivery system, Comput. Commun. 29 (11) (2006) 1889–1895.
[112] S. Moon, J. Yoo, S. Kim, Adaptive interface selection over cloud-based split-layer video streaming via multi-wireless networks, Future Gener. Comput. Syst. 56 (2016) 664–674.
[113] L. Pozueco, X.G. Pañeda, R. García, D. Melendi, S. Cabrero, Adaptable system based on scalable video coding for high-quality video service, Comput. Electr. Eng. 39 (3) (2013) 775–789.
[114] K. Works, E.A. Rundensteiner, Practical identification of dynamic precedence criteria to produce critical results from big data streams, Big Data Res. 2 (4) (2015) 127–144.
[115] X. Yuan, G. Min, Y. Ding, Q. Liu, J. Liu, H. Yin, Q. Fang, Adaptive resource management for P2P live streaming systems, Future Gener. Comput. Syst. 29 (6) (2013) 1573–1582.
[116] Q. Huang, P.P.C. Lee, Toward high-performance distributed stream processing via approximate fault tolerance, Proc. VLDB Endow. 10 (3) (2016) 73–84.
[117] J. Fang, R. Zhang, T.Z. Fu, Z. Zhang, A. Zhou, J. Zhu, Parallel stream processing against workload skewness and variance, in: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, in: HPDC '17, ACM, 2017, pp. 15–26.
[118] M. Hosseini, C. Timmerer, Dynamic adaptive point cloud streaming, in: Proceedings of the 23rd Packet Video Workshop, in: PV '18, 2018, pp. 25–30.
[119] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, in: Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium, 2010, pp. 1–10.
[120] S. Duan, S. Babu, Processing forecasting queries, in: Proceedings of the 33rd International Conference on Very Large Data Bases, in: VLDB'07, 2007, pp. 711–722.
[121] M. Kuhrmann, D.M. Fernández, M. Daneva, On the pragmatic design of literature studies in software engineering: an experience-based guideline, Empir. Softw. Eng. 22 (6) (2017) 2852–2891.
[122] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (4) (2007) 571–583.