

Institute of Parallel and Distributed Systems

University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Masterarbeit

# **Load Shedding in Complex Event Processing with Probabilistic Features**

Timo Grosskopf

<b>Course of Study:</b>	Softwaretechnik
<b>Examiner:</b>	Prof. Dr. Kurt Rothermel
<b>Supervisor:</b>	Henriette Röger, M.Sc. Sukanya Bhowmik, Dr.
<b>Commenced:</b>	April 24, 2021
<b>Completed:</b>	October 28, 2021



## Abstract

Complex Event Processing (CEP) is a stream processing paradigm primarily searching for event type patterns in continuous event streams. Furthermore, load shedding is a common practice in CEP applications when resources are limited and under heavy load. Operators, which become a bottleneck due to bursts in event streams and therefore message queuing, drop event messages with a low matching probability to comply with their latency bound. Existing CEP load shedding mechanisms show a need for improvement if application queries consider an arithmetic relation between event attribute values, which are then called dependent attributes. In this case, incoming events of the same type must be differentiable in their individual matching probability, which bases on their dependent attribute values.

This work introduces the Probabilistic Feature Shedding (PFS) approach, which leverages probability distributions of the dependent attribute values to derive thresholds as shed margins for incoming events. These thresholds categorize incoming events into probable and improbable events to fulfill the arithmetic relation. Improbable events are dispensable for their lower matching probability. The intention behind the PFS mechanism is, that existing shedding mechanisms are complemented with its functionality. In the course of this work, a random shedding mechanism and a distributed shedding approach with linear program solver are each extended with an implementation of the introduced PFS paradigm. Hence, extensive experiments on synthetic datasets as well as a real world dataset are executed with the different shedding paradigms and extensions. The results of all experiments confirm the expectation of a significant improvement in output quality, measured in the number of complex events. Furthermore, simulations show a linear dependency between probabilistic feature evaluation and processing time.



# Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
<b>2</b>	<b>Related work and background</b>	<b>12</b>
2.1	Related work . . . . .	12
2.2	Background . . . . .	14
<b>3</b>	<b>Problem statement and hypothesis</b>	<b>16</b>
3.1	Shortcomings of existing approaches . . . . .	16
3.2	Research questions and hypothesis . . . . .	17
<b>4</b>	<b>Approach</b>	<b>19</b>
4.1	Assumptions . . . . .	19
4.2	Preliminaries . . . . .	20
4.3	Probabilistic feature shedding mechanism . . . . .	22
<b>5</b>	<b>Data collection</b>	<b>30</b>
5.1	Shedding mechanisms . . . . .	30
5.2	Scenarios . . . . .	32
5.3	Datasets . . . . .	34
5.4	Experiment setup . . . . .	44
<b>6</b>	<b>Evaluation</b>	<b>46</b>
6.1	Results . . . . .	46
6.2	Discussion . . . . .	50
6.3	Limitations . . . . .	57
<b>7</b>	<b>Conclusion and Outlook</b>	<b>58</b>
	<b>Bibliography</b>	<b>59</b>
<b>A</b>	<b>Diagrams</b>	<b>61</b>

## List of Figures

4.1	Operator graph of CEP system used in experiments with synthetic data . . . . .	19
4.2	Class diagram of given CEP load shedding framework . . . . .	21
4.3	Message communication between components with linear program shedding active	21
4.4	Class diagram of given CEP load shedding framework extended with PFS . . . . .	24
4.5	Message communication between components with probabilistic feature shedding replacing linear program shedding . . . . .	25
4.6	Threshold determination in a sample cummulative distribution function and corresponding probability density function . . . . .	27
4.7	Message communication between components with probabilistic feature shedding extending linear program shedding . . . . .	28
5.1	Value distributions of datasets D1, D2, D3 . . . . .	35
5.2	Value distributions of dataset D4 . . . . .	35
5.3	Event type ratios of datasets D1 and D4 . . . . .	36
5.4	Event type ratios of datasets D2 and D3 . . . . .	36
5.5	Event type ratios of events 1-30,000 of dataset D5 . . . . .	37
5.6	Event type ratios of events 30,001-60,000 of dataset D5 . . . . .	37
5.7	Value distribution of segment 1 for events 1-15,000 in dataset D6 . . . . .	38
5.8	Value distribution of segment 1 for events 15,001-30,000 in dataset D6 . . . . .	38
5.9	Value distribution of segment 1 for events 30,001-45,000 in dataset D6 . . . . .	38
5.10	Value distribution of segment 1 for events 45,001-60,000 in dataset D6 . . . . .	39
5.11	Excerpt of the reduced, filtered and sorted NYC Taxi Trip dataset . . . . .	40
5.12	Daily taxi trip start times aggregated every 15 minutes . . . . .	41
5.13	Aggregated passenger counts within one week . . . . .	41
5.14	Aggregated longitudes of taxi trip start locations . . . . .	42
5.15	Aggregated latitudes of taxi trip start locations . . . . .	42
5.16	Aggregated longitudes of taxi trip destination locations . . . . .	42
5.17	Aggregated latitudes of taxi trip destination locations . . . . .	43
6.1	Total number of complex events for all experiments with dataset D1 . . . . .	47
6.2	Total number of complex events for all experiments with dataset D2 . . . . .	48
6.3	Total number of complex events for all experiments with dataset D3 . . . . .	48
6.4	Total number of complex events for all experiments with dataset D4 . . . . .	48
6.5	Total number of complex events for all experiments with dataset D5 . . . . .	49
6.6	Total number of complex events for all experiments with dataset D6 . . . . .	49
6.7	Total number of complex events for all experiments with dataset D7 . . . . .	49
6.8	Calculated output qualities for all experiments with dataset D1 . . . . .	51
6.9	Calculated output qualities for all experiments with dataset D2 . . . . .	52
6.10	Calculated output qualities for all experiments with dataset D3 . . . . .	52

6.11	Calculated output qualities for all experiments with dataset D4 . . . . .	52
6.12	Calculated output qualities for all experiments with dataset D5 . . . . .	53
6.13	Calculated output qualities for all experiments with dataset D6 . . . . .	53
6.14	Calculated output qualities for all experiments with dataset D7 . . . . .	53
6.15	Total number of false positives for all experiments with dataset D7 . . . . .	54
6.16	Ordered average fitting times for the chosen 13 distributions, fitting of 2 event types	56
6.17	Average fitting times for 2, 5, 20 and 100 different event types to be fitted . . . .	56
A.1	Total number of false negatives for experiments with dataset D1 . . . . .	61
A.2	Total number of false negatives for experiments with dataset D2 . . . . .	62
A.3	Total number of false negatives for experiments with dataset D3 . . . . .	62
A.4	Total number of false negatives for experiments with dataset D4 . . . . .	62
A.5	Total number of false negatives for experiments with dataset D5 . . . . .	63
A.6	Total number of false negatives for experiments with dataset D6 . . . . .	63
A.7	Total number of false negatives for experiments with dataset D7 . . . . .	63
A.8	Ratios of required to measured processing rates for experiments with dataset D1 .	64
A.9	Ratios of required to measured processing rates for experiments with dataset D2 .	64
A.10	Ratios of required to measured processing rates for experiments with dataset D3 .	64
A.11	Ratios of required to measured processing rates for experiments with dataset D4 .	65
A.12	Ratios of required to measured processing rates for experiments with dataset D5 .	65
A.13	Ratios of required to measured processing rates for experiments with dataset D6 .	65
A.14	Ratios of required to measured processing rates for experiments with dataset D7 .	66
A.15	Ratios of required to measured processing times for experiments with dataset D1	66
A.16	Ratios of required to measured processing times for experiments with dataset D2	66
A.17	Ratios of required to measured processing times for experiments with dataset D3	67
A.18	Ratios of required to measured processing times for experiments with dataset D4	67
A.19	Ratios of required to measured processing times for experiments with dataset D5	67
A.20	Ratios of required to measured processing times for experiments with dataset D6	68
A.21	Ratios of required to measured processing times for experiments with dataset D7	68
A.22	Measured processing times for experiments with dataset D1 . . . . .	68
A.23	Measured processing times for experiments with dataset D2 . . . . .	69
A.24	Measured processing times for experiments with dataset D3 . . . . .	69
A.25	Measured processing times for experiments with dataset D4 . . . . .	69
A.26	Measured processing times for experiments with dataset D5 . . . . .	70
A.27	Measured processing times for experiments with dataset D6 . . . . .	70
A.28	Measured processing times for experiments with dataset D7 . . . . .	70

List of Tables

5.1 Matrix for combinations of value and ratio characteristics to create synthetic datasets 34

5.2 Taxi trips in Manhattan (M), outside of Manhattan (O) and in between . . . . . 41

6.1 Total number of complex events for variable shedding types, processing rates und datasets . . . . . 47

6.2 Resulting output quality for variable shedding types, processing rates und datasets 51



# List of Algorithms

4.1	Probabilistic feature algorithm . . . . .	25
4.2	Algorithm for a best fit distribution to attribute values . . . . .	26
4.3	Algorithm to calculate thresholds from the lower end distribution of an event type attribute value . . . . .	27
4.4	LP_receive thread of the probabilistic feature algorithm when LP shedding is active	28
4.5	Simple excerpt of the dropping mechanism within the operator’s load shedder . .	29
5.1	Recursive refinement of event type drop rates . . . . .	32

# 1 Introduction

Complex event processing (CEP) [BK09; Rob10] is a state-of-the-art paradigm when it comes to monitoring and evaluating continuous streams of information in the form of events, also called a primitive event stream. Continuous event streams offer nearly limitless information insight, which can be derived through aggregation and composition by querying for patterns occurring in primitive event streams. In CEP systems, one or multiple operators implement continuous queries to detect patterns. On a query match, the respective operator emits a complex event for further processing. The complexity of queries varies strongly depending on the application. It has impact on the number of operators, whereas an operator either implements an individual query or is part of a composite query together with other operators. One single operator might suffice for a query of low complexity, but the number of operators is likely to increase with query complexity. Operators then form an acyclic operator graph.

The incoming event streams of an operator contain primitive events emitted by known or unknown event sources as well as complex events emitted by upstream operators in the operator graph. Emitted complex events can either be used as an input for downstream operators or flow into a sink, where the occurrence of the respective complex event gets further processed.

CEP systems are applicable in various environments, whereby event streams can originate from different kinds of sources such as sensor networks, edge and fog computing components or from applications, which are globally distributed over the internet. The volume of these event streams can be huge and often changes in unpredictable rates. Capacities of single machines are easily exceeded in case of incoming event bursts, which causes dissonance with given time constraints, also called latency bounds. A common approach to fix this circumstance is parallelism and distribution of operators on multiple processing nodes [JKD+15]. The corresponding paradigm is called distributed CEP. In case of operator overload the concept of parallelism is only applicable if there are infinite resources available, as it is in cloud computing. However, there are various reasons for limited resources, such as monetary limitations, geographical and installation space constraints or architectural causes to use a fixed number of nodes.

If scaling out is not applicable, operators become a bottleneck when under high load. This results in event message queuing, high latency (or processing times) and delayed pattern detection, which is often unacceptable in CEP applications. One of CEP's features is real time processing of incoming event streams. For example in safety relevant applications or for other time critical pattern detections such as stock exchange applications or fraud detection where short response times are key to efficacy.

Load shedding [HBN13; SBFR19; SBR19; SBR20; TÇZ07; ZVW20] is an established method to resolve bottleneck operators when resources are limited as in edge or fog computing environments, where parallelization [JKD+15; RM19] is not possible. The main goal of the shedding mechanism is to reduce load on the bottleneck operator and lower its resulting latency. This is achieved by dropping a part of incoming events and thereby losing them for all further analysis. Dropping a primitive event prior to processing it will potentially lead to miss a complex event, if the primitive

---

event is part of a match. This would directly reduce the output quality, which is measured in the number of complex events. However, this is acceptable for numerous CEP applications if it allows to maintain real time processing. Shedding mechanisms put up with a loss of output quality as long as the operator latency stays below the application's maximum tolerable boundary. Efforts to keep the output quality high emphasise dropping events of types, which arrive in excessive rates in relation to their portion in the search pattern [HBN13; SBFR19; SBR19; SBR20; ZVW20].

The previously mentioned CEP load shedding efforts all share restrictions in decision making. Single events are dropped according to event types and operator metrics, but not considering other attributes of the event payload. A hybrid CEP and stream processing (SP) query, where a match depends on both event type as well as event attribute values, raises issues which can not be tackled by present shedding approaches. Optimizing output quality is limited while solely taking the event type into account, which gives space for this work's contribution to fill this gap in research. The probabilistic feature shedding (PFS) approach established in this thesis utilizes the natural presence of a probabilistic distribution in attribute values. This allows to predict the probability of incoming event attribute values within their corresponding distributions. Further on, it enables the approach's mechanism to tag incoming events with matching probabilities based on their attribute values and define thresholds, which classify events into high and low matching probability. In this context, the shedding mechanism enables the bottleneck operator to stay below a latency bound by shedding according to the calculated thresholds. It is anticipated that the output quality raises for such attribute dependent query applications in comparison to given CEP shedding mechanisms.

This thesis provides a detailed insight into contributions made in the course of this work, starting with a short state-of-the-art recap in section 2.1 and a necessary clarification of terms and concepts in section 2.2.

Following the related work and background chapter 2, chapter 3 examines existing CEP shedding approaches in section 3.1 and emphasizes a flaw which they have in common. These CEP approaches perform unsatisfactory when applied on widely established CEP applications querying for event attribute value relations. In section 3.2 the driving research questions are formulated, which were established in the course of this thesis and dictated most of the actions. The section closes out with the hypothesis, consequently resulting out of these questions.

Chapter 4 states the rationale behind this work, such as in section 4.1 general assumptions and a given CEP environment where improvements are implemented. The following section 4.3 documents detailed intentions of how to solve the stated issues and prove the formulated hypothesis.

All data for evaluation is raised from experiments further illustrated in chapter 5. First, section 5.1 provides a listing of examined shedding strategies for a throughout evaluation of the introduced PFS mechanism. Section 5.2 illuminates different scenarios regarding data distribution followed by the illustration of datasets in section 5.3, where these scenarios led to. The chapter closes with a description of the experiment setup in 5.4.

The following chapter 6 presents the results of experiments in 6.1 and closes up this thesis' contributions to load shedding in CEP systems with probabilistic features in section 6.2 before chapter 7 provides an outlook to future work.

## 2 Related work and background

There is a broad spectrum of work published by established researchers in the area of SP and CEP. This chapter gives a brief summary of existing load shedding mechanisms in section 2.1, their shedding approach as well as their suitability for queries including attribute value relations. In the course of this work, a wide range of terminologies is used, which are not yet established unambiguously and therefore need extra clarification. The section 2.2 covers all terms and concepts to follow the proposed improvements and results presented in this work.

### 2.1 Related work

Tatbul et al. follow a blackbox vision on the operators with cost and selectivity values. They define drop locations along the data flow arcs to reduce the respective incoming data streams. The shedding mechanism is designed for SP systems, but applies a linear program approach on distributed operators to optimize the output quality and determine the drop probability of input tuples. Applying a linear optimization program to maximize the global output aligns well with the optimization algorithm of the shedding framework this work is based on, which is the reason it obtained special attention upon searching for related work. Within the approach, load shedding plans are previously generated according to drop locations and maximum feasible rates and then stored for future usage in case a bottleneck occurs. The total weighted query throughput is used as a quality metric. The individual queries, which are implemented in the operators, are not considered. Therefore the approach is not fit for CEP load shedding. [TÇZ07]

Slo et al. propose shedding of single incoming events as well as shedding of partial matches. The incoming event shedding mechanism named eSpice assigns an utility value to events, depending on their occurrence in a window of events. This utility value reflects the probability of a primitive event, located at a specific location within a window, to be part of a complex event. A utility function uses statistics from past complex events to determine the importance of single primitive events in relation to their position in an event window. [SBR19] pSpice, which is a partial match shedding framework published by Slo et al., assigns utility values to the state machines tracking the progress of partial matches to full matches. The pSpice utility values are calculated from the probability of a partial match to become a full match, from the estimated processing time a partial match needs to be completed as well as a weight assigned to the pattern. Partial matches with a low utility value are dropped from the operator in order to reduce the latency for every incoming event to check if they align with existing partial matches. eSpice and pSpice improve established SP shedding mechanisms to be capable of CEP load shedding by considering the query pattern and other incoming event types instead of independently assessing utility values to single events. [SBFR19] In 2020, Slo et al. published a shedding mechanism combining both, eSpice and pSpice. The resulting approach called hSpice assigns utility values to event types considering the importance

of the single event, as well as the importance of existing partial matches. The research group adapt a probabilistic model, which uses the type and position of an event in a window and the states of partial matches to assign a utility to events corresponding to different partial matches. [SBR20]

Zhao et al. propose a hybrid shedding mechanism to combine partial match shedding with incoming event shedding. The rationale behind the mechanism is a cost model classifying a partial match right after its creation. This classification of partial matches reflects a matching probability with incoming events to maximize the number of complete matches. Additionally, it serves as a foundation for decisions which partial matches are dropped when a latency bound is exceeded. Dropping partial matches is a fine granular optimization of the CEP system and has a small effect on the latency. If the latency violation is high, the hybrid mechanism switches to input-based shedding. The same partial match classification is utilized to identify incoming events which have the lowest influence on the output quality when shed. Since shed events are not processed in the first place, the influence of input-based shedding on the latency is higher and stays active as long as the latency bound is violated. [ZVW20]

He et al. investigate two shedding mechanisms called integral load shedding and fractional load shedding. The integral approach focuses on dropping types of events or partial query matches. In fractional load shedding a proportion of event types or partial query matches is dropped randomly. They examine both, memory bound and CPU bound shedding, and base on assigning utility values for queries. The algorithm minimizes utility loss, which in turn maximizes utility gain, and categorizes the queries into promising and unpromising queries. The set of event types in promising queries will be accepted, while the unpromising ones are all rejected, which means they are dropped by the load shedder. [HBN13]

These existing shedding mechanisms all are restricted in their functionality to the same characteristics. These are event input rates, the implemented event type patterns and the global output rate. Even though queries potentially consider logical or arithmetic relations between event attribute values, they are not incorporated into shedding decisions. This work starts off on the basis of a given CEP load shedding framework further outlined in section 4.2. The framework fixes latency bound violations by optimizing local and global output quality at runtime with a linear program solver. It utilizes incoming rates of different event types as well as the operator patterns to calculate a processing rate given a required maximum operator latency. The processing rate is determined as percentage of each event type's input rate. The requested maximum latency for a single event at the bottleneck operator can be maintained when the calculated processing rates are applied. This work targets to improve the described CEP shedding mechanism.

### 2.2 Background

This section introduces terminologies and concepts, which must be defined or clarified to follow the narrative of this thesis.

**Complex Event Processing (CEP)** is the core paradigm of this thesis, which uses queries to search continuous streams of primitive events for pattern occurrences.

**Primitive event** refers to an atomic event, which is not further decomposable and therefore the lowest level event in the CEP system. All primitive events in this work are tuples of the structure  $e = \langle t, p, id, ts \rangle$ . Thereby  $t$  refers to the event type,  $p$  is the payload containing all attributes and their respective values embodied in the event,  $id$  is a unique message identification and  $ts$  a unique timestamp containing information when the event was issued.

**Event type** is a distinction of events, which allows the operator to identify primitive events without accessing their payload. In this work, event types are integer values.

**Event attributes** (also called **event features**) are contained in the payload and accessible in a key-value fashion.

A **query** is the center piece of CEP applications containing an event pattern and dependencies between attributes, which will lead to a complex event if all constraints are satisfied. An application query can be divided and distributed to multiple operators, potentially running on different nodes.

**Patterns** contain a sequence of event types to search for in the event stream. A pattern can be a complete application query or a sub-query of a composed query. In this work, a pattern also contains arithmetic relations between values of event attributes, which further specify the query.

A **Composed query** is formed when a sophisticated application query is divided into multiple operators, each implementing a simplified sub-pattern. Participating operators emit complex events to downstream operators in the operator graph in order to communicate when a sub-pattern match is found.

**Operator** is the architectural component implementing a pattern or sub-pattern of the application query. An operator has primitive events as input from event sources or complex events as input from upstream operators. It emits output events, which are always complex.

An **operator graph** is composed of multiple operators, each implementing a sub-query of the application query. When the application query is fulfilled, the furthest downstream operator will emit a complex event, which serves as application output.

**Downstream and upstream** operators are the subsequent and preceding operators from the view of an operator within the operator graph.

**Load shedders** are implemented in operators and responsible for decisions of processing individual events. Load shedders contain a shedder configuration, which defines the rationale to drop single incoming events.

**Bottleneck operator** refers to an operator, which is overloaded by incoming events and violates its latency bound.

**Latency** of an operator refers to the average processing time the operator needs to process a single event.

**Latency bound** of an operator is its maximum possible latency to guarantee proper performance. In case the latency bound is violated during runtime, the load shedding mechanism reduces the incoming event stream to reduce the latency.

**Optimal processing rate**  $\lambda_{\omega,opt}$  is the proportion of all incoming events, which can be processed without violating the latency bound.

**Dependent attribute** values are part of an arithmetic relation in an application query with other events' attribute values.

**Arithmetic relations** are applied in search queries and specify the relationship, dependent event attribute values must exhibit to each other in order to generate a complex event.

**Complex events** result from a successful match of a query or sub-query in an operator. Operator queries are previously defined by the application query and operator graph.

A **partial match** is present in an operator from the point on, when the initial event for an operator pattern is detected. It is available while subsequent events of the pattern arrive, but not yet complete. It gets discarded as soon as the match is complete.

**Matching probability** indicates the likelihood of an incoming event to participate in a match.

**Thresholds** are used as margins to decide if incoming events have a high or low matching probability. The event attribute values are used as reference for classification.

**Output quality** is for this work defined by the ratio of the number of complex events arriving at the sink to the number of all possible output events. Output quality is a floating point number, typically in the range [0,1], and is decreased when complex events stay undetected. These undetected events are referred to as false negatives. Output quality values higher than 1.0 imply false positives.

**False negatives** are the difference of total emitted complex events with shedding activated to the benchmark number of total emitted complex events when shedding is inactive. False negatives occur when events are shed, which would have led to a match.

**False positives** are the contrary to false negatives and refer to complex events which would not occur in the benchmark run, but appear due to shedding activity.

**Linear program** refers to the optimization function of the local output of single operators or the global output of the whole application. This work's contributions build upon a framework with a linear program already implemented and seen as given.

**Probabilistic feature shedding (PFS)** is the improvement this work suggests as extension for existing shedding mechanisms.

**Probabilistic feature component (PFC)** depicts the actual implementation of this work's PFS mechanism as an extension to the given framework.

**Apache Kafka** [Apa] is a distributed event streaming platform, providing a robust publish-subscribe functionality. Events are published on channels, called topics. Kafka offers an interface where publishers emit events to topics and Kafka consumers subscribe to these topics to subsequently process events in the order they are published.

### 3 Problem statement and hypothesis

Previous works of related research communities pose a comprehensive and thorough analysis of load shedding mechanisms in CEP applications. They also show that shedding mechanisms of SP systems [BDM04; TÇZ+03; TÇZ07] do not suffice the needs of CEP systems [HBN13; TÇZ07; ZVW20]. This section provides an overview of the approaches and spotlight of different CEP load shedding strategies. It points out their limitations where this work begins to provide an improvement. Existing shedding mechanisms assign utility scores to event types [SBR19; ZVW20] and partial matches [SBFR19] in single operator CEP systems. They also use matching probability predictions [TÇZ07] for single events or event windows in distributed CEP systems. These mechanisms focus on the event patterns implemented in operators and the input rate events arrive at these operators. Solutions for single operator applications [SBR19; ZVW20] either consider the operator patterns and incoming event windows to assign utility values to single event types or the internal operator state to assign utility values to the state machines. In the first case, an incoming event can be dropped without risk of much quality reduction if its type has been assigned a low utility value. This mechanism reduces the incoming event stream for a bottleneck operator to lower its latency. In the second case the operator's internal state is considered to assign utility values to its state machines, which keep track of partial matches. The state machine is dropped completely if the probability to form a match in the near future is low. This mechanism reduces the computational effort when checking for incoming events to complete any partial matches. Both approaches result in a lower strain on the respective operator.

The solutions provided for distributed SP [TÇZ07] maximize the local operator output, the global application output or both by implementing a linear program. The linear program uses current or possible violation of the maximum tolerable latency, the rates event types arrive in and knowledge about operator patterns to calculate an optimal processing rate for each event type. The goal is to reduce the latency at the bottleneck operator. These processing rates are either pre-processed or calculated at runtime, as in the CEP shedding framework used in this work to implement an improvement. Resulting processing rates are in this context forwarded to the bottleneck operator's load shedder, which applies the rates on incoming events by dropping the exceeding percentage of each incoming event type.

#### 3.1 Shortcomings of existing approaches

All mentioned approaches analyze incoming events only in regard of their event type, in order to make a processing or dropping decision to maximize output quality *OQ*. This decision is made nondeterministically by shedding the appropriate ratio of incoming events of the respective type. This indeterminism limits the number of feasible reasonable applications strongly.

An existing distinction between classic stream processing, also called data stream management system (DSMS), from CEP blurred within the last decades [Bas07]. Per definition, SP applications



are used to continuously compute the informational content of event streams, whereas CEP applications emphasise the detection of patterns in event streams. This definition does not limit CEP to pattern detection, but enables it to perform further event stream analysis in order to produce complex events [Rob10]. CEP systems provide additional meaningful information when extended with SP functionality by considering attribute values of events. A pattern of primitive events could only lead to a match, if their respective attribute values in the payload are in a specific relation to each other. For example suppose a stock exchange application searches for a pattern of share price alterations in different stocks. The alteration of a share price is signaled by a primitive event with the stock as type. The queried pattern of stock alteration events is only of interest, if the alterations have a specified relation to each other. To be more precise, a simple query sequence is a sequence of  $seq(A, B)$ , whereas an event of type  $A$  signals a price alteration of stock  $A$ , and an event of type  $B$  an alteration of stock  $B$ . Additionally, the CEP system only produces a complex event if the price alteration of stock  $A$  is greater than the alteration of stock  $B$ . Alternatively, one could only be interested in this sequence  $seq(A, B)$  if the alteration of  $A$  is positive and the alteration of  $B$  is negative. These examples are only two of the possible queries when combining patterns in event streams with the information contained in these events.

Even though such overlap of CEP and SP is fairly common in existing CEP applications [BGAH06; GWC+06] and does not pose any novelty, there is no shedding approach present to consider any event attributes next to the event type. This work attempts to establish a relation between event types and attribute values to assess the importance of single events.

## 3.2 Research questions and hypothesis

The missing functionalities in existing CEP load shedding solutions stated in section 3.1 raise attention to several research questions. This section provides an aggregation of the questions towards a hypothesis, which will further be analyzed in the course of this work.

The following list contains the research questions which received the most attention during the analysis and development

- Q1 : Is there a way to assess a matching probability of events based on dependent attribute values?
- Q2 : How to predict relevant attribute values?
- Q3 : How can knowledge about event type, event attribute values and query requirements lead to a shedding decisions?
- Q4 : Where can PFS be applied to improve existing shedding mechanisms?
- Q5 : Which metrics justify the efficacy of a PFS extension?

Shedding mechanisms further explained in section 2.1 consider the type of incoming events as well as the ratios these events arrive in at the operators. Therefore the final shedding decision for any single event is made in a nondeterministic fashion. For example, assume in a CEP application the latency bound is violated and an existing shedding mechanisms examined, that events of a certain type should be processed with a rate of 50 %. There are various ways to reach this goal such as dropping every second event. Another way is to assign a random number  $x$  in the range  $[0,1]$  to every event and drop the event if  $x > 0.5$  resolves to *True*. Both methods achieve the correct processing

rate, but the final decision is not reproducible. There is no deterministic approach developed, which takes other event attributes into consideration to distinguish events of the same type.

The existing approaches offer a high output quality for complex queries based on event types, but all of them are flawed when operator queries browse for relations between values of event's dependent attributes. The first step to tackle given shortcomings is to extend an existing load shedding framework with an analytic component to evaluate event attribute values and make predictions based on the probability distribution of these past attribute values. Consecutive steps collect application results of the implemented shedding algorithm when executed on a simple event query with a synthetic dataset of events. The results are then compared to existing solutions running on the same query and dataset. Finally the proposed shedding solution will be tested on a real world dataset and again compared to an existing shedding solution.

At this point the running example and subject to experiments on synthetic data is introduced and further elucidated in section 4.1. The example considers an operator  $\omega$  receiving events of types 0 and 1, whereby every event contains an attribute value as payload  $p$ . While a sequence of event  $e_0$  of type 0 followed by event  $e_1$  of type 1 is well known in CEP systems, an additional relation between attribute values of  $e_0[p]$  and  $e_1[p]$  introduces higher complexity when load shedding is applied. In this running example for synthetic data, the sequence  $seq(e_0, e_1)$  and relation  $e_0[p] > e_1[p]$  both need to resolve to *True* for the operator to emit a complex event.

Because the importance of an event depends on its type and also its attribute values, shedding mechanisms which act nondeterministically within a single event type will inevitably decrease the output quality. The PFS mechanism approaches to track values of dependent attributes and determines their probabilistic distributions. This leads to the capability of deriving threshold values for required processing rates. According to these thresholds, incoming events are classified into either having a high or a low probability to participate in a match. Consequently the derived thresholds are used as drop margins to reach specific processing rates in order to comply a latency bound. This deterministic categorization into important and unimportant events will eventually increase the output quality when extending a nondeterministic shedding mechanism. The improvement is achieved through minimizing false negatives in the total number of complex events when shedding is applied. False negatives will decrease because the load shedder will first drop events with the lowest probability to form a match, considering the event attribute values. This additional consideration is new to shedding mechanisms in CEP.

The effectiveness of the introduced PFS extension can be assessed by the total number of complex events at the application output. The output quality  $OQ_{p\_rate, s\_type, data}$  is assessed for all experiments introduced in section 5.4 for different processing rates  $p\_rate$ , shedding types  $s\_type$  and datasets  $data$ . It is defined in section 2.2 as the quotient between the number of emitted complex events  $n_{CE, p\_rate, s\_type, data}$  when shedding is active, and the complex events  $n_{CE, bm, s\_type, data}$  from a benchmark run when shedding is inactive. A quality value close to 1 denotes a low number of false negatives and is most desirable.

$$OQ_{p\_rate, s\_type, data} = \frac{n_{CE, p\_rate, s\_type, data}}{n_{CE, bm, s\_type, data}}$$

This chapter presented a listing of substantial research questions, which lead to the hypothesis that PFS improves application output when appointed as extension of existing CEP load shedding mechanisms. The implementation, examination and validation of the research questions and hypothesis is further outlined in chapters 4, 5 and 6.

## 4 Approach

This chapter represents the main part of the contribution provided to the CEP load shedding paradigm. Section 4.1 states assumptions about the environment and framework conditions for the approach to be applicable and verifiable. Section 4.2 gives an introduction to the CEP framework this work builds upon. Section 4.3 specifies the implemented concepts to achieve the desired improvements.

### 4.1 Assumptions

In order to reduce the complexity of shedding experiments described in chapter 5 to the necessary minimum, a number of assumptions regarding the environment and synthetic dataset are made in this section. This work's experiments base on a CEP system receiving primitive events of types  $\{0, 1\}$ , all having the same structure  $\{type, payload, msg\_id, timestamp\}$  where the payload  $p$  is a continuous floating point number following a normal distribution. Section 5.3 further specifies the defined distributions and attribute values of applied datasets. Primitive events are emitted by a source *SOURCE* in a constant emission rate  $\lambda_{SOURCE}$ . The source emits events to a Kafka topic, where an operator  $\omega$  subscribes to. The operator implements the application query and emits a complex event CE to an output topic in case of a pattern match, where a sink *SINK* subscribes to the output topic and further evaluates emitted complex events. Figure 4.1 displays the described setup as an operator graph.



**Figure 4.1:** Operator graph of CEP system used in experiments with synthetic data

Operator  $\omega$  implements the application query  $q$  as a combination of 2 conditions  $q = C_1 \wedge C_2$ . The condition  $C_1$  is a sequence of event types  $C_1 = seq(e_0, e_1)$  with  $e_0[t] = 0, e_1[t] = 1$ , condition  $C_2$  is an arithmetic relation between event payloads  $C_2 = e_0[p] > e_1[p]$ . Incoming events can only participate in one match and are consequently unavailable for any further partial matches once they are used. Partial matches are removed after a specified timeout  $to$  from the operator storage to keep actuality high and latency low.

The arrival rate of incoming events at the operator  $\omega$  is traced as  $\lambda_\omega$ . Additionally, the ratios of all incoming event types  $\gamma_{\omega, type}$  are tracked as a fraction of their respective arrival rates  $\lambda_{\omega, type}$  to the operator's incoming rate  $\lambda_\omega$ .

$$\gamma_{\omega, type} = \frac{\lambda_{\omega, type}}{\lambda_\omega}$$

There exists a given optimal processing rate  $\lambda_{\omega,opt}$  of events, which can be processed by operator  $\omega$  without violating its maximum latency. The fraction of the optimal processing rate and the event arrival rate is denoted as optimal event processing ratio  $\gamma_{\omega,opt}$ . It is specified as a fraction of the arrival rate  $\lambda_{\omega}$  and serves as indicator, which proportion of all incoming events the operator can process, while staying below its maximum latency.

$$\gamma_{\omega,opt} = \frac{\lambda_{\omega,opt}}{\lambda_{\omega}}$$

Optimal processing rates  $\lambda_{\omega,type,opt}$  for each event type can be derived from the operator query  $q$ , arrival rates  $\lambda_{\omega,type}$  and the latency violation. As soon as shedding reduces incoming event rates at the bottleneck operator to the optimal processing rates, the system will achieve the total optimal processing rate  $\lambda_{\omega,opt}$ . This means, the shedding mechanism uses past incoming rates  $\lambda_{\omega,type}$  to calculate corresponding optimal processing rates  $\lambda_{\omega,type,opt}$  for each event type. It also provides the most beneficial incoming event type ratios  $\gamma_{\omega,type,opt}$  to reach the highest potential number of complex events.

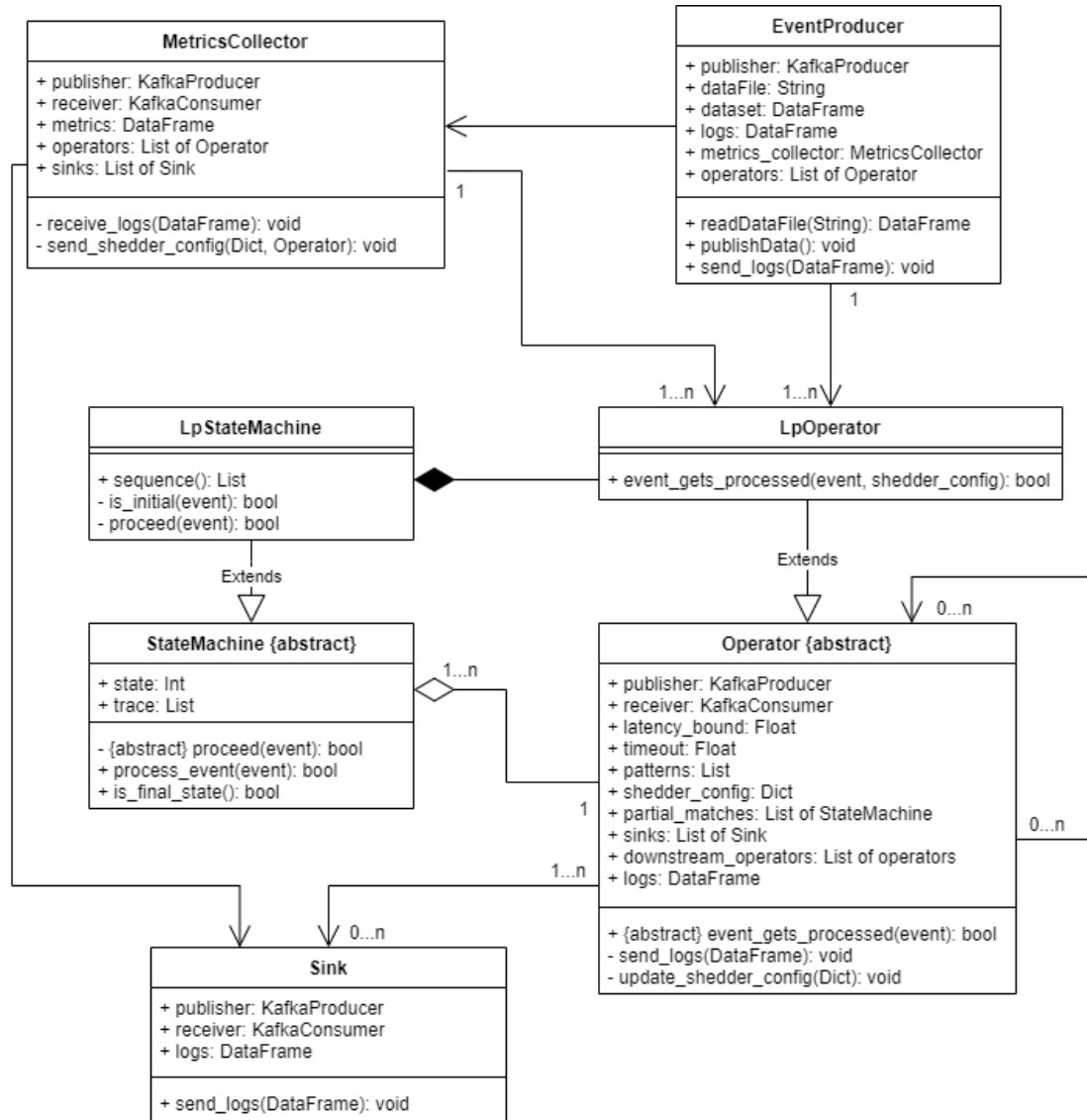
On receiving an initial event for a query pattern, operators instantiate state machines as partial matches to wait for completion through consecutive events. These partial matches exist until a specific timeout  $t_o$  after their creation is reached or until completion. The PFS approach does not utilize partial matches, but discards them nevertheless after their timeout to prevent the operator to be spammed. Operators are capable of accessing attributes in the payload of events before starting to actually process the event. This assumption is essential for the approach to be beneficial.

## 4.2 Preliminaries

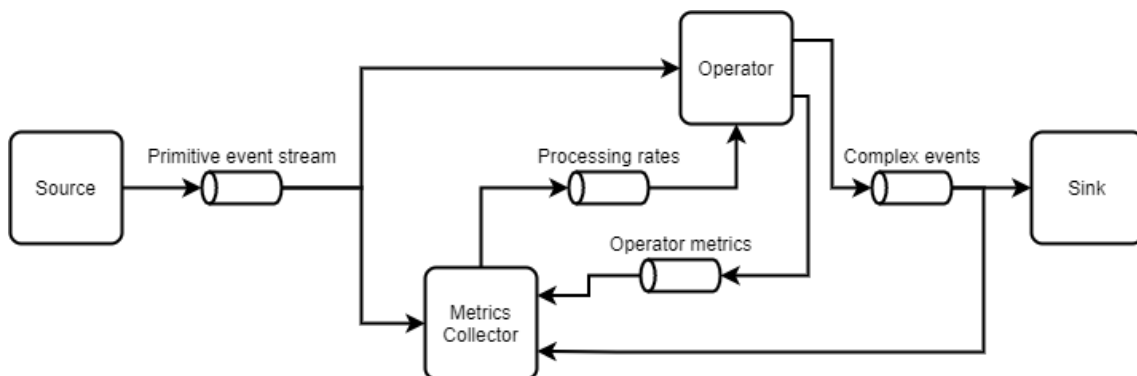
Figure 4.2 shows the simplified architecture of the given CEP framework, which serves as basis for this work's contribution and as reference for load shedding improvements.

The framework starts a producer process as *SOURCE*, which emits events in the rate  $\lambda_{\omega}$ . Events are published as a primitive event stream to a Kafka topic  $t_{\omega,in}$  with the targeted operator as topic name, where operators and monitoring components can subscribe to. Complex events are equally published to Kafka topics to be available for operators, monitoring components and sinks. The communication setup is displayed in figure 4.3.

A metrics collector component task, hosting a shedding controller, runs in parallel to monitor the system's statistics. The shedding controller receives frequent updates of incoming event rates  $\lambda_{\omega}$  and latency  $latency_{\omega}$  at the monitored operator. It uses the latest statistics as well as knowledge about the operator's search patterns and the required maximum latency  $latency_{\omega,max}$  to execute a linear program (LP). This LP returns optimized processing ratios  $\gamma_{\omega,type,opt}$  by either maximizing the potential local operator output or global application output considering the operator patterns and latency bounds  $latency_{\omega,max}$ . The processing ratios  $\gamma_{\omega,type,opt}$  contain one ratio value for each corresponding event type. The responsible LP is given and subject to another work by researchers of University of Stuttgart. The metrics collector sends latest optimal processing ratios  $\gamma_{\omega,type,opt}$  as shedder configurations to the operators. The operators update their load shedders with the new shedder configurations. In case no metrics collector is started, operators can apply random shedding mechanism with a fixed processing rate. Random processing rates are set on operator startup and are not updated during runtime.



**Figure 4.2:** Class diagram of given CEP load shedding framework



**Figure 4.3:** Message communication between components with linear program shedding active

Operators subscribe to their respective event input topics  $t_{\omega,in}$  and process incoming events as soon as they arrive. First the operator's load shedder checks whether the incoming event gets processed. This check is a nondeterministic action and bases on optimal processing ratios  $\gamma_{\omega,type,opt}$  provided by the metrics collector. In case the event gets discarded, the operator takes the next incoming event from its input topic. In case it gets processed, the operator starts its procedure of examining existing state machines and initial state machine conditions. Every operator implements at least one state machine which represents the search pattern. For every incoming event taken from the operator input topic  $t_{\omega,in}$ , the operator sequentially checks against all existing partial matches whether it can proceed or complete a match. If a partial match can proceed, the operator executes all necessary steps and eventually continues by processing the next incoming event from the topic  $t_{\omega,in}$ . Provided there exists no partial match for the event to complement, the operator checks whether the event corresponds to the initial state of any state machine. In this case, the operator creates a new partial match by instantiating a state machine. The new partial match is added to the list of existing partial matches for future incoming events to check against. Events which do neither complement partial matches nor initiate the state machine get discarded. The operator has no usage for events, which do not participate in the currently available states of its search query. Instead it continues with the next incoming event from the topic  $t_{\omega,in}$ .

Operators implement a partial match timeout  $to_{\omega}$ . Partial matches which exist for longer than the operator's timeout are discarded no matter how complete they are. This sanction gives a guarantee of actuality in potential complete matches. Timeouts need individual evaluation and verification for each use case.

As mentioned before, a metrics collector runs centrally to monitor operator inputs as well as latency. Monitored operators aggregate their processing times and send the statistics to the Kafka topic  $t_{logs}$ , where the metrics collector subscribes to. It is the metrics collector's task to keep track of operator latency and detect violations. Moreover, bottleneck operators are started as remote processes on separate machines to guarantee measured latency to be independent from any other CPU straining processes like the metrics collector. Upon completion of partial matches before they reach their timeout  $to_{\omega}$ , the respective operator publishes a complex event to its output topic  $t_{\omega,out}$ . Depending on the application query and operator graph, downstream operators or sinks subscribe to the output topic. Downstream operators further process the emitted complex events whereby sinks provide them as application output.

### 4.3 Probabilistic feature shedding mechanism

From the given assumptions and preliminaries of section 4.1, this work's contribution considers the following case. There is a bottleneck at the operator  $\omega$  because its predefined latency bound is violated. Therefore, the incoming event stream needs to be reduced by shedding single incoming events. From the latency bound violation and current incoming rates, a given algorithm calculates an optimal processing rate  $\lambda_{\omega,opt}$  for operator  $\omega$  to stay below the latency bound. To achieve a reduction of the incoming event stream to  $\lambda_{\omega,opt}$ , the PFS mechanism will determine a set of threshold values to the load shedder, which act as drop margins for different incoming event types. In previous solutions, the load shedder was provided processing rates for event types instead of actual threshold values. This section shows how the PFS approach derives the correct threshold values for the operator to achieve the calculated optimal processing rate  $\lambda_{\omega,opt}$ . Thresholds guarantee, that events with irrelevant attribute values are shed, whereas events with relevant attribute values,

which receive a high matching probability, are kept for further processing. The correct thresholds are calculated in a new probabilistic feature component (PFC), containing the main contribution to enable PFS in CEP applications. Figure 4.4 shows how the PFC fits into the given framework architecture.

To keep operator output quality high (i.e. minimize false negatives when shedding is active), the PFC determines thresholds according to matching probabilities. The matching probability is dependent on the event's type and its payload, as explained in section 3.2. Every event type gets assigned one or two thresholds, depending on the application. One threshold suffices if the application query searches for attribute values being lower or higher than another dependent attribute value. The PFC will return two threshold values if the query seeks for attribute values within a window. These thresholds classify every incoming event for the load shedder to decide which events to drop. The load shedding contribution consists of two steps. It will first replace the metrics collector and work with fixed processing rates. The appropriate communication between components is shown in figure 4.5. The metrics collector will be reintroduced when the aptitude of PFS is clarified.

First, the PFC is introduced in algorithm 4.1, which fits common probability distributions to the incoming payload values, selects the best distribution and calculates thresholds for load shedding. It gets initialized with an application dependent update interval and an update timeout. They both initiate the same update mechanism, depending on which one arises first. The update interval starts the update mechanism after a specific number of received events, whereas the timeout starts it after a specific time. Additionally, the PFC receives initial processing rates and thresholds in case it acts as standalone shedding mechanism and does not get its processing rates dictated by an LP shedding mechanism. In the main loop, all events are continuously tracked and stored in memory. At stated intervals of either incoming events or a fixed timeout, the probabilistic feature algorithm described in algorithm 4.1 calculates the cumulative distribution function (CDF) of each event type for previously defined relevant event attributes. The algorithm to determine the CDFs is described in algorithm 4.2. Algorithm 4.1 uses the calculated CDFs to derive thresholds for each event type according to the initial processing rates, if there is no LP shedding mechanism activated. In case a linear program shedding mechanism provides processing rates, the PFC starts a separate thread to receive these updated processing rates. Whenever the PFC receives new processing rates, it initiates the calculations and update mechanism. Algorithm 4.4 shows the LP\_receive thread and describes how these updates are received and further processed.

In order to derive a CDF for each event type attribute, past event attribute values are used as data to fit common probability distributions using the `scipy.stats` package as shown in algorithm 4.2. The PFC finds the best distribution using smallest squared error (sse) comparison and additionally provides corresponding parameters to determine the probability density function (PDF). From the PDF, the PFC calculates the CDFs  $CDF_{type}$  for each event type. All CDFs accept floating point values as input and return a continuous floating point value in the range (0,1). The returned value is the statistical probability of any data point in the same distribution to have a smaller value than the input value. Values close to 0 mean that the requested data point is on the lower end in the distribution. Values close to 1 mean that the data point value is greater than the majority of data points in the distribution.

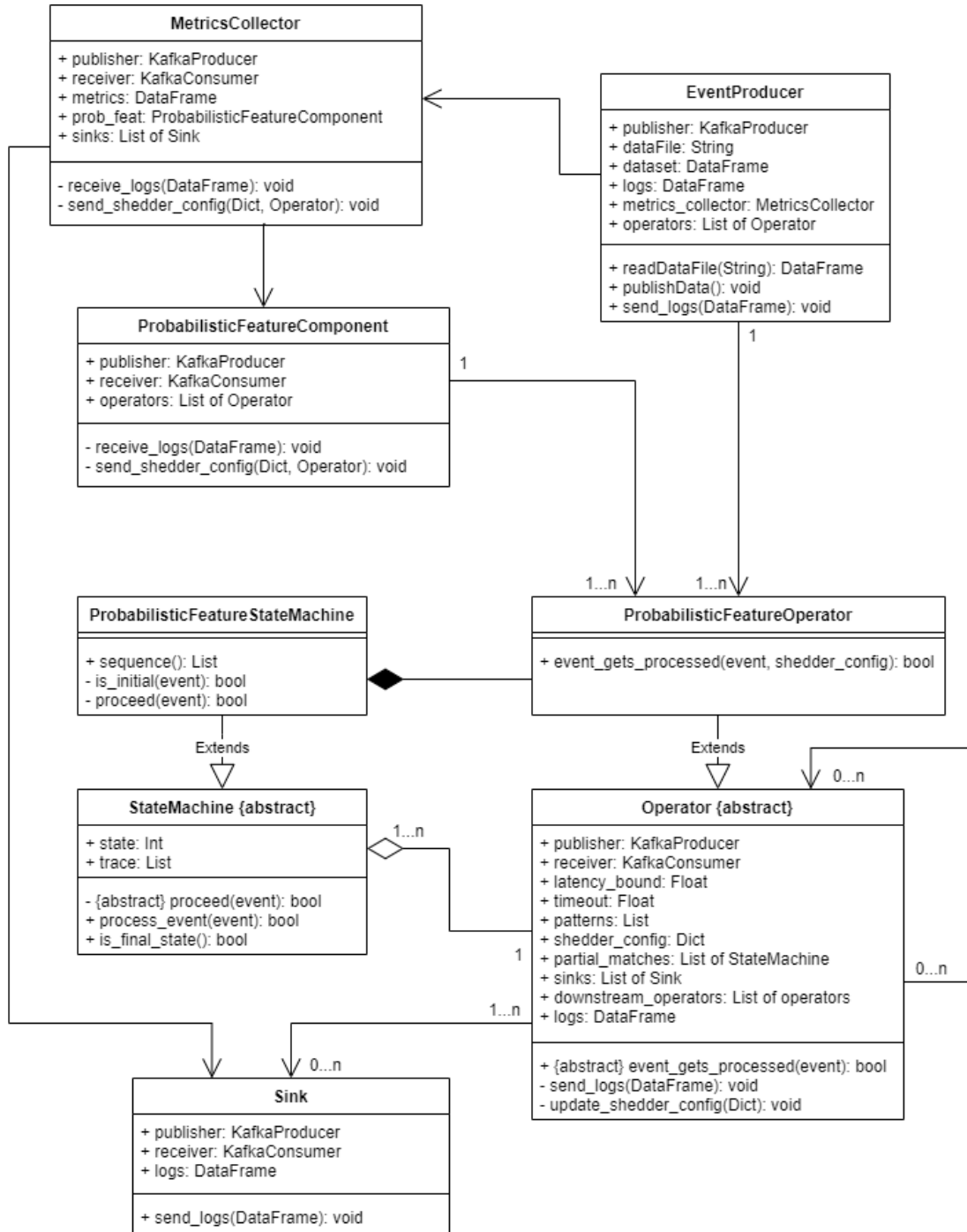
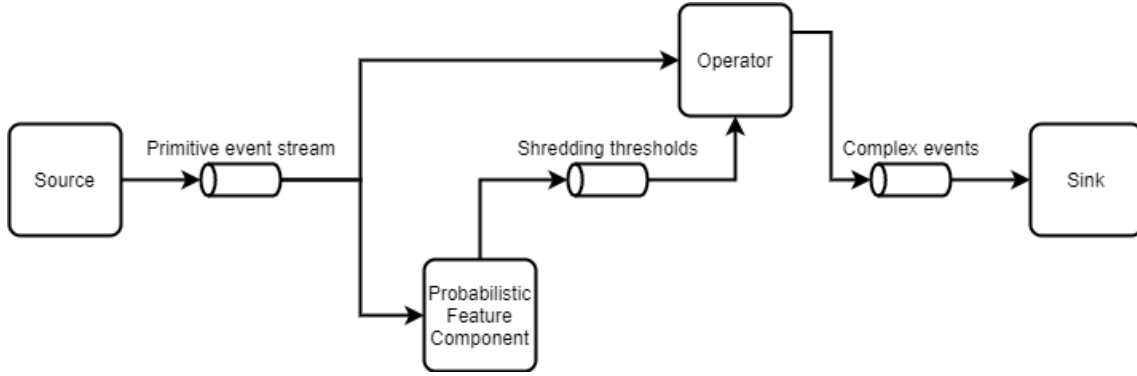


Figure 4.4: Class diagram of given CEP load shedding framework extended with PFS





**Figure 4.5:** Message communication between components with probabilistic feature shedding replacing linear program shedding

---

**Algorithm 4.1** Probabilistic feature algorithm

---

```

INPUT update_interval, update_timeout, processing_rates, thresholds,
      LP_shedding_active, relevant_distribution_types
INIT events, event_counters, CDFs ← empty dictionary
SET distributions ← relevant_distribution_types
START update_timer
for each event type do
  INIT events[type] ← empty list
  SET event_counters[type] ← 0
end for
if LP_shedding_active then START LP_receive_thread
end if
repeat
  RECEIVE event e
  APPEND e to list of events[e.type]
  INCREMENT integer event_counters[e.type]
  if event_counters[e.type] == update_interval OR update_timer reached update_timeout then
    for each event type do
      CALL CALC_CDF with events[type] and distributions, RETURNS cdf
      SET CDFs[type] ← cdf
    end for
    if not LP_shedding_active then
      CALL CALC_THRESHOLDS with CDFs and processing_rates,
        RETURNS new_thresholds
      if new_thresholds ≠ thresholds then
        SET thresholds ← new_thresholds
        SEND thresholds to operator
      end if
    end if
    SET event_counters[e.type] ← 0
    RESET update_timer
  end if
until RECEIVE final flag
  
```

---

**Algorithm 4.2** Algorithm for a best fit distribution to attribute values

---

```

function CALC_CDF(data, distributions)
  INIT best_distribution
  SET best_sse  $\leftarrow$  INF
  for each distribution in distributions do
    SET params, sse  $\leftarrow$  CALL scipy.stats.fit(data, distribution)
    if sse < best_sse then
      SET best_distribution  $\leftarrow$  params
      SET best_sse  $\leftarrow$  sse
    end if
  end for
  RETURN best_distribution
end function

```

---

The PFC uses the CDF of each event type as shown in algorithm 4.3 to reversely find a threshold value  $T_{type}$ , which corresponds to a given processing rate  $\lambda_{\omega, type}$ . This means, depending on the arithmetic relation between event type attributes, the threshold value separates the range of values in the distribution  $CDF_{type}$  into two sections. The lower section corresponds to  $\lambda_{\omega, type} * 100\%$  of the event attribute values and serves as the proportion to process if the arithmetic relation of event attribute values prioritizes the lower part of values. The upper section corresponds to  $(1 - \lambda_{\omega, type}) * 100\%$  of event attribute values, which can be dropped. In case the arithmetic relation in the query requires the upper part of event attribute values, then the lower part can be dropped. Figure 4.6 displays the plots of a CDF and PDF taken from a sample distribution with the mean  $\mu = 0.0$  and standard deviation  $\sigma = 1.0$ . An exemplary processing rate of  $\lambda_{\omega, type} = 0.85$  and the displayed sample distribution in figure 4.6 as attribute values of incoming events illustrate the procedure to derive a threshold  $T_{type}$  for this exemplary event type. Assuming the overload operator requires an optimal processing rate  $\lambda_{\omega, 1} = 0.85$  for type 1 events and the application query states the following. It searches for an event sequence  $seq(0, 1)$ , whereas it only leads to a match if the attribute value of the type 1 event is smaller than the value of the type 0 event. Type 1 events receive the highest matching probability if they have low attribute values. With the CDF, the PFC finds a threshold value  $T_1 = 1.035$  for type 1 events, which separates the probability density function displayed in the upper diagram in figure 4.6 into proportions of 85 % and 15 %. The blue area corresponds to 85 % of all incoming events of type 1. When shedding all events with an attribute value higher than  $T_1 = 1.035$ , the operator reduced its processing rate for type 1 events to  $\lambda_{\omega, 1} = 0.85$ , all the while having low risk of losing events with a high matching probability.

The previously introduced algorithms show, the PFC can calculate thresholds as load shedder configuration depending on a fixed processing rate of each event type. At this point the metrics collector gets reintroduced, which calculates optimal event type processing rates  $\lambda_{\omega, type, opt}$  depending on the incoming event ratios  $\gamma_{\omega, type}$  and potential latency bound violations at operator  $\omega$ . Therefore, the processing rates emitted by the metrics consumer are redirected to the PFC instead of the operator. The PFC calculates appropriate thresholds to the processing rates and sends them to the operator's load shedder. Figure 4.7 shows the communication between components after implementing the PFS extension.

---

**Algorithm 4.3** Algorithm to calculate thresholds from the lower end distribution of an event type attribute value

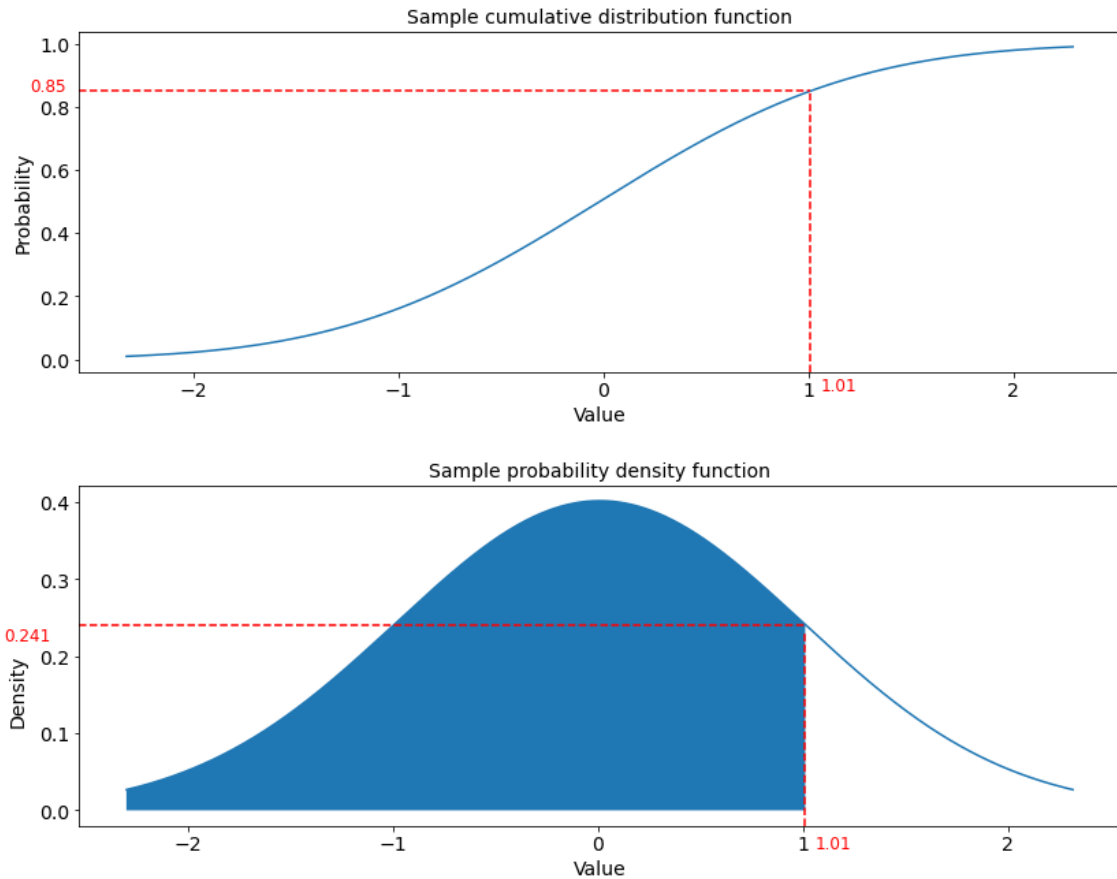
---

```

function CALC_THRESHOLDS(CDFs, processing_rates)
    SET thresholds  $\leftarrow$  empty dictionary
    for each event type do
        for each value in CDFs[type] do
            if CDFs[type](value) > processing_rate[type] then
                SET thresholds[type]  $\leftarrow$  value
            end if
        end for
    end for
    RETURN thresholds
end function

```

---



**Figure 4.6:** Threshold determination in a sample cumulative distribution function and corresponding probability density function

**Algorithm 4.4** LP\_receive thread of the probabilistic feature algorithm when LP shedding is active

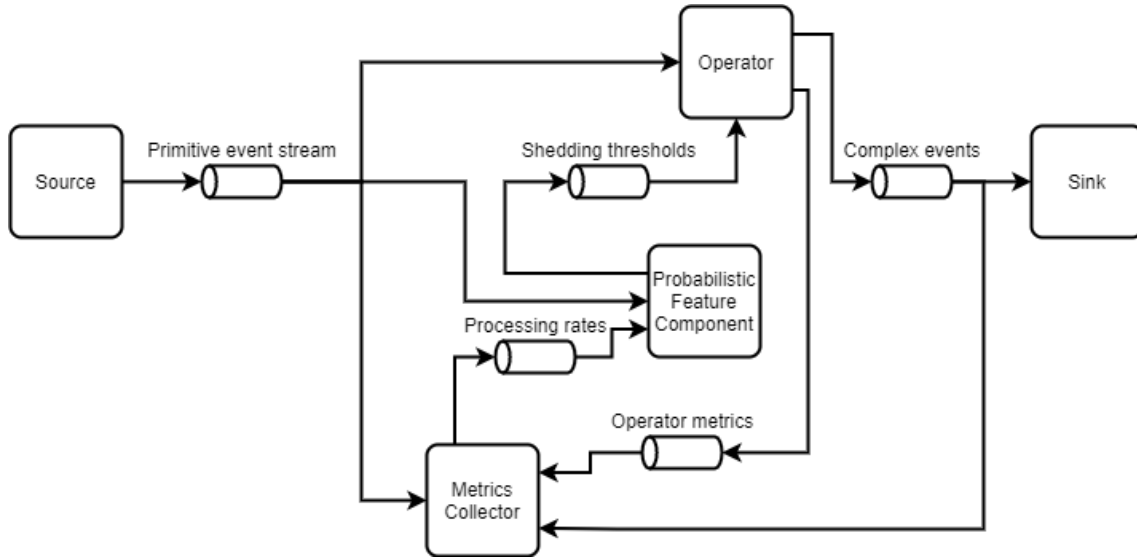
---

```

INIT thresholds
SET processing_rate  $\leftarrow$  1.0
loop
  RECEIVE new_processing_rate
  if new_processing_rate  $\neq$  processing_rate then
    SET new_thresholds  $\leftarrow$  CALL calculate_thresholds(CDFs, processing_rates)
    if new_thresholds  $\neq$  thresholds then
      SET thresholds  $\leftarrow$  new_thresholds
      SEND thresholds to operator
    end if
  end if
end loop

```

---



**Figure 4.7:** Message communication between components with probabilistic feature shedding extending linear program shedding

Algorithm 4.4 shows how the PFS extension is used to improve the already existing linear program shedding mechanism. The PFS extension algorithm runs in a separate thread, which receives processing rates through a Kafka topic  $t_{PFS}$  from the metrics collector and its shedding controller. It can access the CDFs, which are frequently reevaluated and updated by the PFC's main thread shown in 4.1 to calculate threshold values and send them to the bottleneck operator  $\omega$ .

The bottleneck operator receives and stores threshold values as shedder configuration in a separate message receiving thread. Algorithm 4.5 displays in short, how the operator checks every incoming event attribute value with an individually implemented function if the event gets processed. The processing of the event is not part of this work, which is why the algorithm 4.5 simply depicts it with 'process(e)'.

---

**Algorithm 4.5** Simple excerpt of the dropping mechanism within the operator's load shedder

---

```
SET shedder_configuration ← initial_shedder_configuration
repeat
  RECEIVE event e
  SET thresholds ← shedder_configuration[e.type]
  if event_gets_processed(e.payload, thresholds) then
    CALL process(e)
  end if
until RECEIVE final flag
```

---

This chapter introduced an environment of assumptions and given technologies in section 4.1 in order to solve the problems stated in chapter 3.1. Two approaches were proposed to either shed with probabilistic features and a fixed processing rate or by extending a given shedding mechanism, which provides processing rates to the PFC.

The next step is to evaluate the efficacy of a PFS extension first in comparison to a rudimentary random shedding mechanism and then to the introduced LP load shedding mechanism. The load shedding mechanism uses an LP to determine suitable processing rates, aiming to reduce the latency back to a predefined latency bound in case of a violation. In the following chapter the means to assess efficacy of the PFS extension are introduced. Different scenarios are explained in section 5.2 on how the structure and variable parameters of synthetic test data is determined in section 5.3. The experiments executed on the datasets are specified in section 5.4. The produced data will be discussed and evaluated in chapter 6.

## 5 Data collection

One of the core tasks in this work is to raise expressive data in order to support the hypothesis formulated in section 3.2. This chapter designates and exemplifies all parts and steps of data collection. This includes a brief summary of shedding types applied on the data, the different scenarios targeted by the datasets and experiments, the applied synthetic and real world datasets as well as the setup of experiments. The target of data collection is mainly to prove efficacy of the PFS extension, which is measured in output quality. Changes in output quality can be assessed in the difference of emitted complex events, whereas benchmark experiments are executed without load shedding to determine the total possible amount of complex events. The resulting complex events of an experiment with active load shedding is set into relation with the result of its corresponding benchmark run to reveal both false negatives and false positives occurring due to load shedding activity. Conclusions about the results are discussed in chapter 7.

### 5.1 Shedding mechanisms

As already stated in section 4.3, the PFS extension is introduced and examined in 2 stages. First random shedding is improved before stepping forward to extend the more complex LP load shedding mechanism. In the course of experiments, the results are examined to proof efficacy in the random shedding experiments, only then the behavior of the PFS extension is executed in combination with a linear program, which determines and updates processing rates at runtime. For structural reasons, the four shedding mechanisms are all presented together in this section before the results of experiments are summarized in chapter 6.

#### 5.1.1 Random shedding

A random shedding mechanism is already implemented. On operator instantiation, each event type receives a processing rate in the range  $(0, 1)$  within the load shedder configuration. The processing rate tells the operator's load shedder which proportion of incoming event types must be processed, consequently it also tells which proportion must be dropped. A value of 0.0 means no event will be processed and a value of 1.0 makes the operator process every event of the respective type. A nondeterministic mechanism decides at processing time, which events to process eventually. This shedding mechanism has the lowest expectations with regards to output quality.

### 5.1.2 Probabilistic feature shedding

The first step to improve random shedding is to dynamically calculate threshold values during runtime for the bottleneck operator  $\omega$ . For the probabilistic feature shedding approach, a previously determined required processing rate  $\lambda_{\omega,opt}$  is defined to force the PFC to reduce the rate of incoming events  $\lambda_{\omega}$ . This required processing rate and the incoming event type ratios  $\gamma_{\omega,type}$  are used to determine optimal processing rates for each event type  $\lambda_{\omega,type,opt}$ . The PFC leverages these optimal processing rates to determine threshold values for each event type  $T_{type}$ . The calculated thresholds serve the operator's load shedder to decide which events to process and which to shed according to their attribute values. Shedding each event type at its threshold will result in processing its respective processing rate and eventually lead to the total required processing rate  $\lambda_{\omega,opt}$  at the bottleneck operator. The PFC calculates optimal processing rates for each type from the respective drop rate  $dr_{type}$  as  $\lambda_{\omega,type,opt} = 1 - dr_{type}$ , whereas the drop rate is calculated for all types as defined in equation 5.1. The event type drop rates depend on incoming ratios of event types  $\gamma_{\omega,type}$ , total required processing rate  $\lambda_{\omega,opt}$ , the number of occurrences of the respective type  $n_{type}$  in the operator's patterns and the total number of events  $n$  participating in the operator's search patterns. Drop rates are finally recursively refined as shown in algorithm 5.1. Refinement is needed if the drop rate for an event type is negative  $dr_{type} < 0$ . This is the case if its incoming ratio  $\gamma_{\omega,type}$  is too low for the required processing rate  $\lambda_{\omega,opt}$ . A negative drop rate  $dr_{type}$  states, that for an optimal event type ratio there is need of adding events of this certain event type. This is not possible, so the drop rates of other event types need to be adapted in order to eventually achieve the required processing rate  $\lambda_{\omega,opt}$ . Since processing rates are updated frequently for each event type, the resulting thresholds are equally updated and published to the operators as described in chapter 4.

$$(5.1) \quad dr_{\omega,type} = \frac{\gamma_{\omega,type} - \frac{\lambda_{\omega,opt}}{n_{types}}}{\gamma_{\omega,type}}$$

### 5.1.3 Linear program shedding

A linear program shedding mechanism has already been implemented and provided as subject to improvement and comparison for the contribution and experiments of this thesis. The details of its functionality is part of another research work and not further discussed in this thesis. For the purpose of running shedding experiments and to use their results as assessment of improvement, this section gives a coarse outline of linear program shedding, whereas the correct performance of the shedding mechanism is seen as given. As described in section 4.2, an existing metrics collector provides processing rates  $\lambda_{\omega,type}$  for each event type to the bottleneck operator. Therefore, the metrics collector receives frequent updates on the operator latency to detects latency bound violations as well as the ratios of incoming event types  $\gamma_{\omega,type}$ . The severity of a latency bound violation and incoming rates of event types serve as input values for a linear program solver, which provides optimal processing rates  $\lambda_{\omega,type,opt}$  for each event type in return. Main task of the solver is a latency reduction through reduced processing rates while considering the search query pattern and corresponding incoming rates of event types participating in the pattern. Resulting processing rates are applied by the load shedder and decrease the load on the bottleneck operator to achieve the optimal processing rate  $\lambda_{\omega,opt}$ , which leads to processing times below the latency bound.

**Algorithm 5.1** Recursive refinement of event type drop rates

---

```
function CALC_DROP_RATES(incoming_rates, processing_rate)
  INIT processing_rates, drop_rates  $\leftarrow$  empty dictionary
  for each event type do
    CALCULATE drop_rate as in equation 5.1
    SET drop_rates[type]  $\leftarrow$  drop_rate
  end for
  for each event type do
    if drop_rates[type] < 0 then
      SET reduced_incoming_rates  $\leftarrow$  incoming_rates REDUCED BY type
      SET processing_rate  $\leftarrow$  processing_rate - incoming_rates[type]
      CALL CALC_DROP_RATES with reduced_incoming_rates and processing_rate,
        RETURNS drop_rates
      SET drop_rates[type]  $\leftarrow$  0.0
    end if
  end for
  RETURN drop_rates
end function
```

---

**5.1.4 Linear program shedding extended with probabilistic feature shedding**

A practical application of probabilistic feature shedding is the extension of the already existing LP load shedding mechanism, which optimizes processing rates at runtime using a linear program. The LP load shedding is the same as described in section 5.1.3, providing optimized processing rates for each event type. The processing rates base on incoming event rates, a latency bound and a violation of that bound. However, the resulting processing rates are not directly sent to the bottleneck operator, to decide which proportion of the incoming events to process, but are forwarded to the PFC running in a separate process. At the PFC, optimized processing rates are used as described in section 4.3 to determine thresholds from the frequently updated CDFs of query relevant event attribute values. These thresholds are sent to the bottleneck operator, as in the PFS mechanism in section 5.1.2. In contrary to the given linear program shedding mechanism in section 5.1.3, threshold values enable the load shedder to deterministically drop events in order to achieve the requested processing rates. However, the requested optimized processing rates result from the required maximum latency and its violation, as explained in section 5.1.3. It has to be noticed, that output quality is expected to improve because events with lowest matching probability are preferably shed.

**5.2 Scenarios**

Continuous event streams, which are input to a CEP application where relations of attribute values are part of the search query, do typically not follow any rules regarding incoming rates, value distribution or maximum operator latency. In order to prove the functionality of the PFS mechanism against a broad variety of possible event stream properties, this section provides potential scenarios. These scenarios reflect possible properties in the event streams which need special consideration. All these scenarios have to be covered the synthetic datasets to prove efficacy of the PFS mechanism.



A Scenario is built from combinations of different characteristics of the two dominant event stream properties in this work: value distribution and event type ratios. Value distributions themselves can either be static with parameters staying the same during the whole experiment or they change over the course of an experiment which makes them dynamic. Changing distribution parameters during one single experiment entail an adjustment of threshold values at runtime. Simultaneously, the value distributions of different event types can either have a large or a small overlap, depending on their mean  $\mu_{type}$  and variance  $\sigma_{type}$ . In case of dynamic distributions, there are two cases. First is, the distribution means  $\mu_{type}$  change similarly but the overlap remains constant as seen in figures 5.7 through 5.9 in section 5.3. Second is, their means  $\mu_{type}$  change unequally, which causes a variable overlap as seen in figures 5.9 and 5.10 in section 5.3. There are 3 prominent cases of characteristics within the event type ratio. The ratios can be the same for all event types, which means they are balanced. The ratios can be imbalanced, meaning the ratios of incoming event types  $\gamma_{\omega, type}$  differ. In the imbalanced case the ratios can be steadily or unsteadily imbalanced. Steady imbalance shows a constant difference between incoming event type rates and unsteady imbalance displays a variance in the difference of the respective incoming rates.

Value distribution characteristics are

- V1 static distributions with a large overlap
- V2 static distributions with a small overlap
- V3 dynamic distributions with a constant overlap
- V4 dynamic distributions with a variable overlap

Event type ratio characteristics are

- R1 balanced and steady ratios
- R2 imbalanced and steady ratios
- R3 imbalanced and unsteady ratios

The combinations of these characteristics lead to a matrix, displayed in table 5.1. During the conceptual process of synthetic datasets, all fields in the matrix have to be covered by either one dataset or a combination of datasets for the experiments to be wholesome. All synthetic datasets for the validation of this work, which are introduced in section 5.3, are chosen to leave no blanks in the matrix of characteristic combinations. Additionally, the optimal processing rate in the PFS mechanism or the latency bound in linear program shedding mechanisms acts as a third characteristic which also has influence on certain scenarios. As an example scenario, one event type has a very low incoming event type ratio  $\gamma_{\omega, type}$ , but is relevant for the search query. There are scenarios where load shedding is active, but the output quality suffers if any of the low rate event type events is shed. This behavior needs to be considered and processing rates of the experiments need to be chosen over a broad spectrum to cover all potential scenarios. Therefore, the experiments defined in section 5.4 implement a variance of processing rates. This last scenario is an example for the necessity of refining drop rates as described in section 4.3. Dataset D7 is not present in table 5.1 for the simple reason that it is not part of the synthetic datasets and most likely covers every cell of the matrix with characteristics.

		Value characteristics			
Ratio characteristics		V1	V2	V3	V4
	R1	D1	D4	D6	D6
	R2	D2 & D3	D5	D5 & D6	D5 & D6
	R3	D5	D5	D5 & D6	D5 & D6

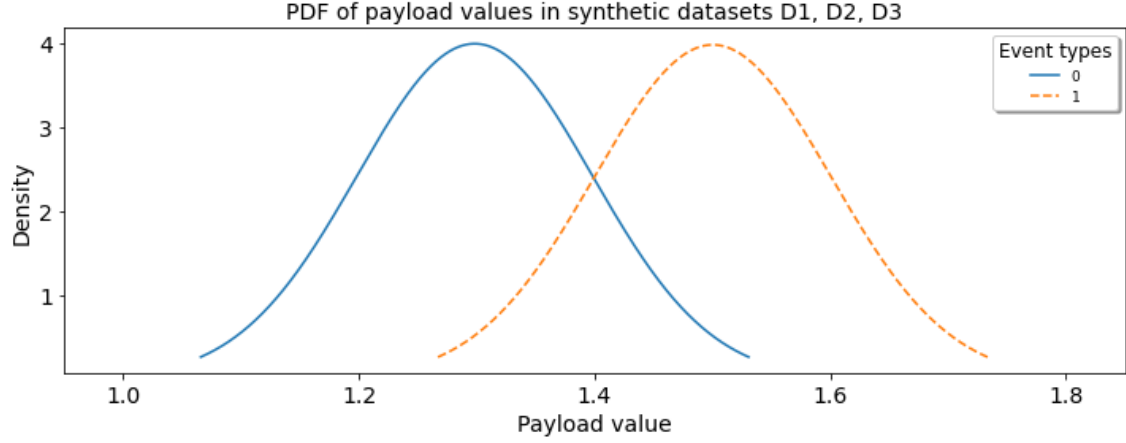
**Table 5.1:** Matrix for combinations of value and ratio characteristics to create synthetic datasets

### 5.3 Datasets

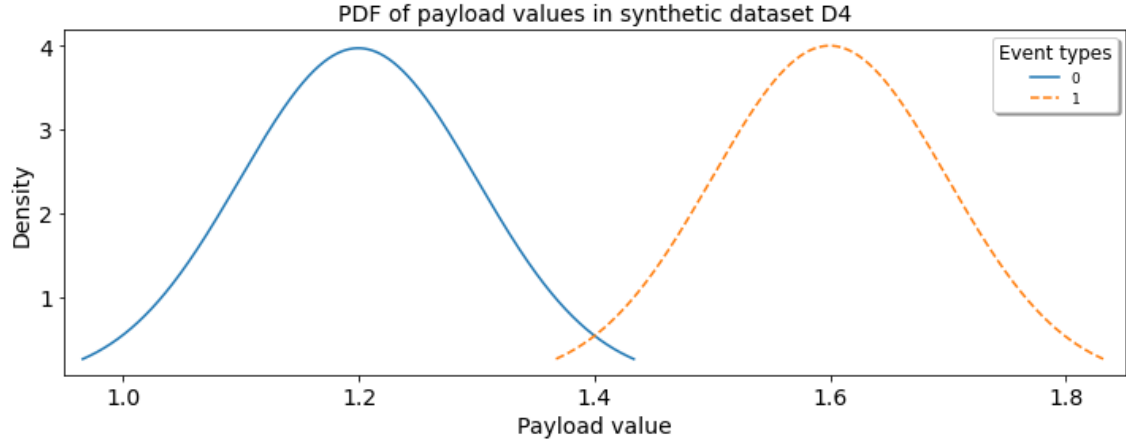
The previous sections 5.1 and 5.2 introduced different shedding mechanism and scenarios for primitive event streams. This section takes all possible scenarios into consideration and gives a summary about the datasets, which have been used to prove the efficacy of the PFS mechanism subject to this thesis. Additionally, this section gives an insight into the decisive properties of the final datasets. For the purpose of covering all scenarios in table 5.1 six synthetic datasets {D1, D2, D3, D4, D5, D6} of equal size were created to provide all properties requested. These synthetic datasets are further explained in the following section 5.3.1. To proof the improvement through a PFS extension of existing shedding mechanisms, all experiments for shedding types introduced in 5.1 have additionally been executed on a real world dataset D7. Section 5.3.2 gives further insight into the structure and content of the real world dataset as well as arrangements in the application search query.

#### 5.3.1 Synthetic data

All synthetic datasets consist of only two event types {0,1} to keep complexity at a minimum, because the search query for these datasets is defined for two event types only. These datasets each consist of 60,000 primitive events. This high number of primitive events allows for changes in event stream properties while still having a sufficient amount of events in between the changes. The payload values of events are for the datasets {D1, D2, D3, D4} chosen to be distributed in a normal fashion around a mean  $\mu_0 = 1.3$  for type 0 events and around a mean  $\mu_1 = 1.5$  for type 1 events. The standard deviation is chosen to be  $\sigma = 0.1$  for all synthetic datasets and does not change over the course of an experiment. Dataset D1 provides a static normal distribution of attribute values and has a large overlap between type 0 and type 1 events as shown in figure 5.1. Incoming ratios of each event type are at  $\gamma_{\omega, type} = 50\%$  and do not change over the course of an experiment. Dataset D4 is similar to D1 regarding statics of the attribute value distributions and constant ratios of  $\gamma_{\omega, type} = 50\%$  for the whole experiment as seen in figure 5.3. However, it differs in the overlap of the attribute value distributions as seen in figure 5.2. A shift in the mean values to  $\mu_0 = 1.2$  and  $\mu_1 = 1.6$  causes the PDFs to have less overlap. With a smaller overlap, the probability for a match decreases strongly for the defined search query in section 4.1. This shift in means of probability distribution consequently reduces the number of matches and emphasizes the importance of attribute value dependent shedding in order not to miss complex events when they occur. As seen in figure 5.4 datasets D2 and D3 show a steady imbalance in the incoming event type ratios. 70 % of all events in dataset D2 are of type 0 and 30 % are type 1 events. Dataset D3



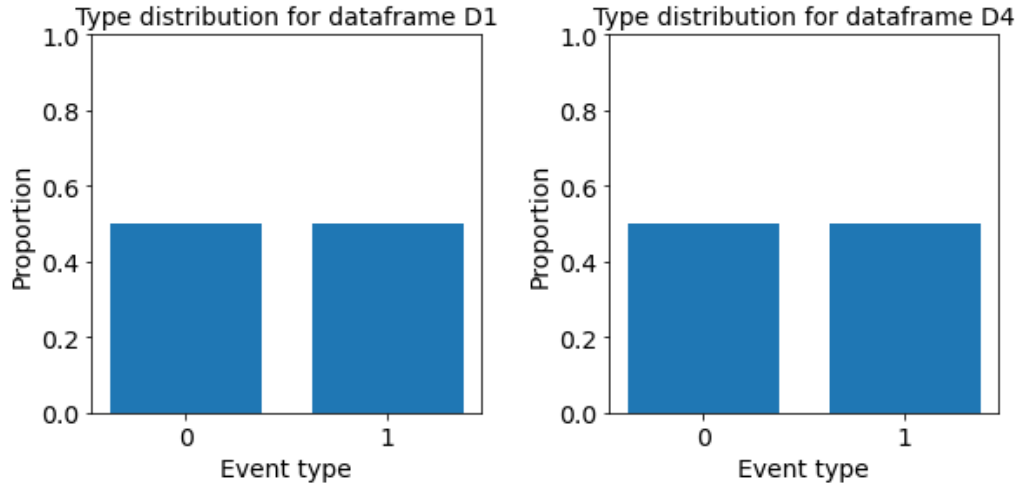
**Figure 5.1:** Value distributions of datasets D1, D2, D3



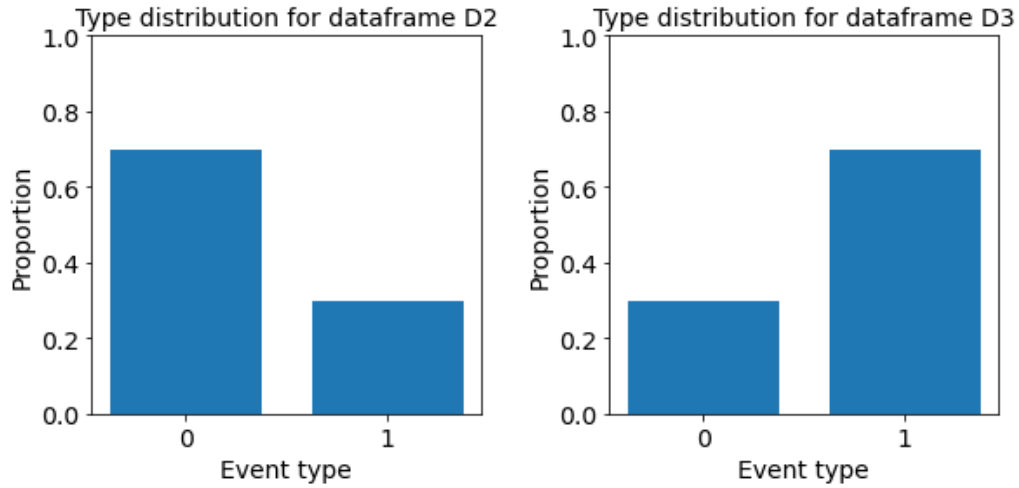
**Figure 5.2:** Value distributions of dataset D4

flipped this relation to 30 % type 0 events and 70 % type 1 events. With this strong imbalance and the assumed query from section 4.1, the load shedder is forced to prioritize event types with the lower incoming ratio  $\gamma_{\omega, type}$  over higher represented event types.

Datasets D5 and D6 introduce a dynamic behavior of in section 5.2 mentioned characteristics in event streams. In dataset D5 the event type ratios change after 15,000 events, which makes up for 25 % of all events. The first change of incoming event type ratios is from 60 % type 0 events and 40 % type 1 events to 40 % type 0 events and 60 % type 1 events. After 30,000 events, which is 50 % of all events, the ratios change to 80 % type 0 events and 20 % type 1 events. After 45,000 events, which is 75 % of all events, the ratios flip to 20 % type 0 events and 80 % type 1 events. Figures 5.5 and 5.6 show the various incoming ratios in D5. This dataset examines the ability of the shedding mechanism to adapt to unsteadiness of incoming event ratios and to update the thresholds accordingly. The PFS mechanism illustrated in section 5.1 as well as the LP shedding approach take care of suboptimal incoming rates for the search pattern. Dataset D6 has a dynamic behavior in the attribute value distributions. After 15,000 events, which is 25 % of all events, the initial attribute value distributions change simultaneously. The mean of type 0 events changes from  $\mu_{0,1} = 1.3$

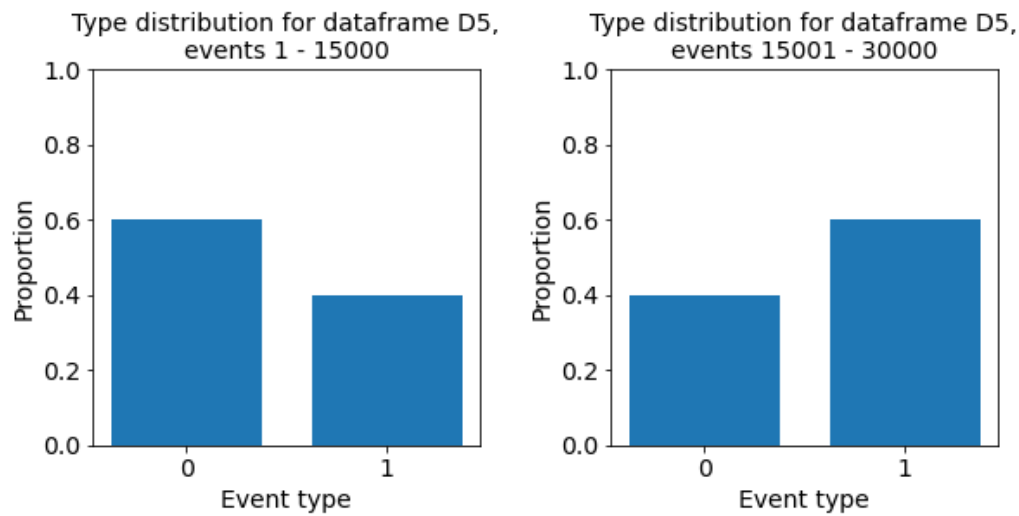


**Figure 5.3:** Event type ratios of datasets D1 and D4

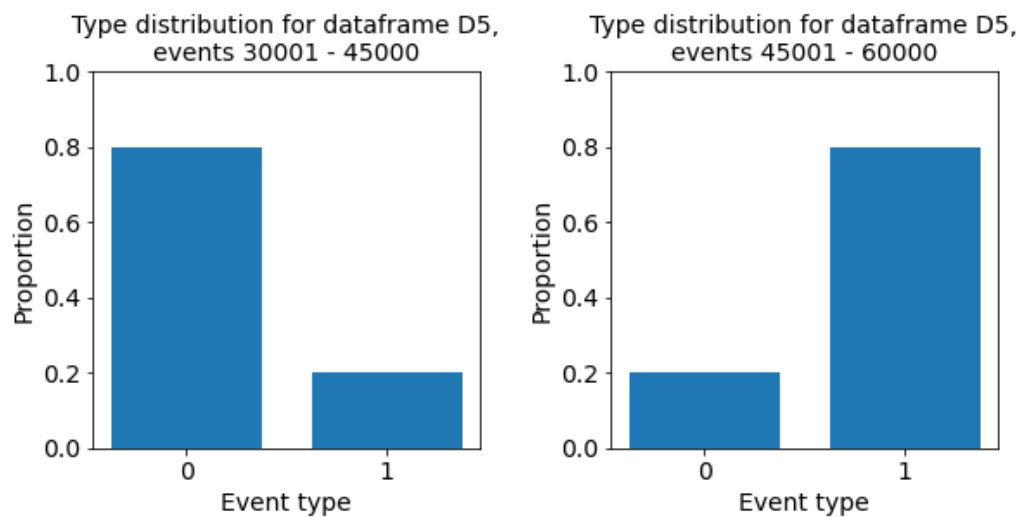


**Figure 5.4:** Event type ratios of datasets D2 and D3

to  $\mu_{0,2} = 1.4$  and the mean of type 1 events changes from  $\mu_{1,1} = 1.5$  to  $\mu_{1,2} = 1.6$ . After 30,000 events, which is 50 % of all events, the distributions again change simultaneously to  $\mu_{0,3} = 1.5$  and  $\mu_{1,3} = 1.7$ . The last change in attribute value distributions appears at 75 % of all events, which is 45,000 events, but falls out of alignment with a change to  $\mu_{0,4} = 1.4$  and  $\mu_{1,4} = 1.8$ . This last change in value distributions decreases the overlap of PDFs for type 0 and type 1 events. Figures 5.7 through 5.10 display the variety of type 0 and type 1 event attribute distributions in the dataset. The first two distribution alterations represent a dynamic attribute value distribution with constant overlap, whereas the last alteration makes up for the dynamic distribution with a variable overlap.



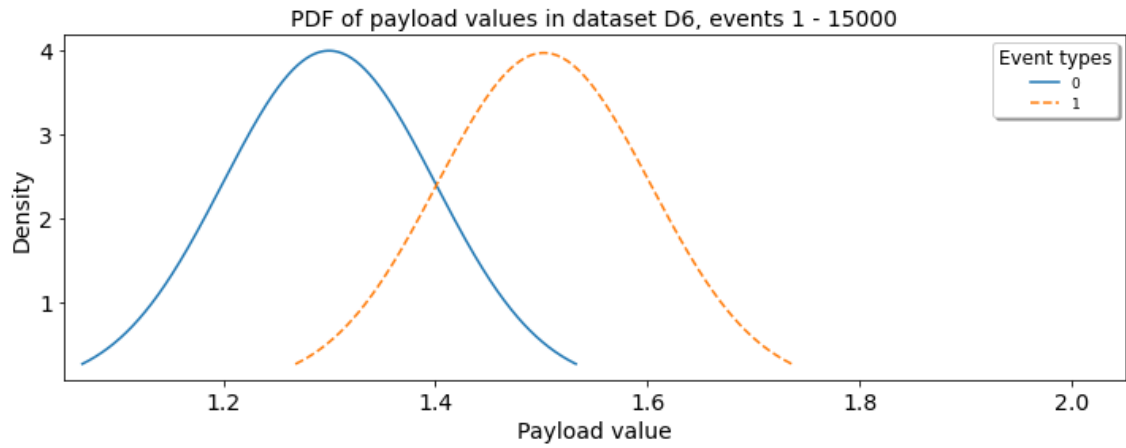
**Figure 5.5:** Event type ratios of events 1-30,000 of dataset D5



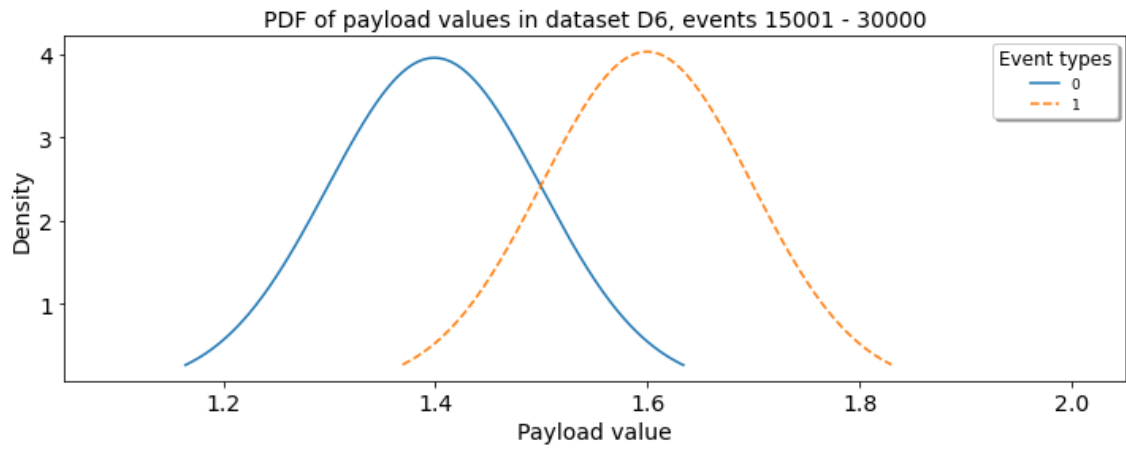
**Figure 5.6:** Event type ratios of events 30,001-60,000 of dataset D5

### 5.3.2 Real world data

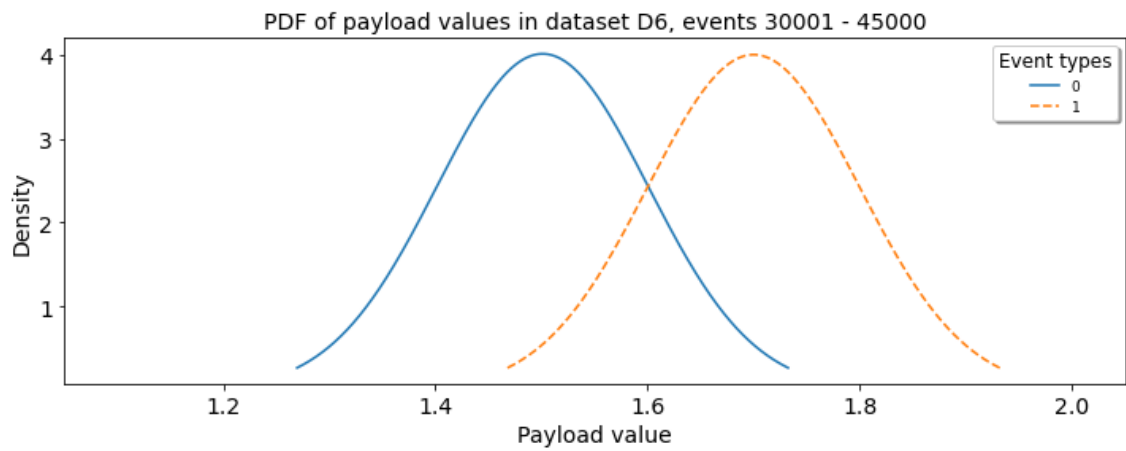
Dataset D7 is an excerpt of the New York City taxi trip data provided by Chris Whong [Chr]. The complete dataset consists of 12 files formatted in csv, each representing one month of the year 2013. Each file contains information about taxi trips published by NYC's Taxi and Limousine Commission. The original dataset is reduced by removing unneeded entries, but the sheer amount of taxi trips is still too much for reasonable experiments. Therefore, only one week is chosen as an expressive excerpt of taxi trips. Even though the further reduced dataset depicts one week, the total amount of primitive events still consists of 3,381,158 taxi trips. To pick a representative time frame of taxi trips, the biggest desire was to find a common week outside of vacation periods and which does not contain any holidays. Therefore the final data only consists of taxi trips in the week from 07th of October 2013 to 13th of October 2013, which are chronologically ordered by pickup



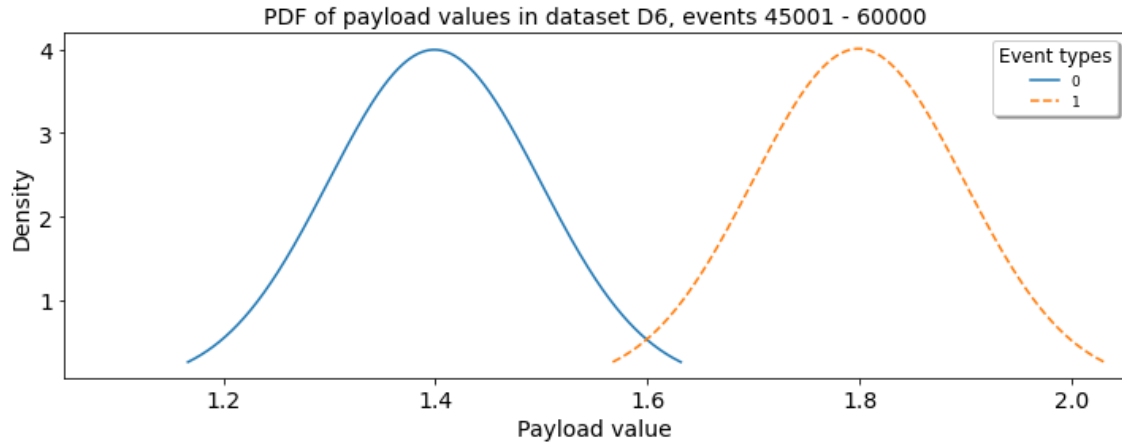
**Figure 5.7:** Value distribution of segment 1 for events 1-15,000 in dataset D6



**Figure 5.8:** Value distribution of segment 1 for events 15,001-30,000 in dataset D6



**Figure 5.9:** Value distribution of segment 1 for events 30,001-45,000 in dataset D6



**Figure 5.10:** Value distribution of segment 1 for events 45,001-60,000 in dataset D6

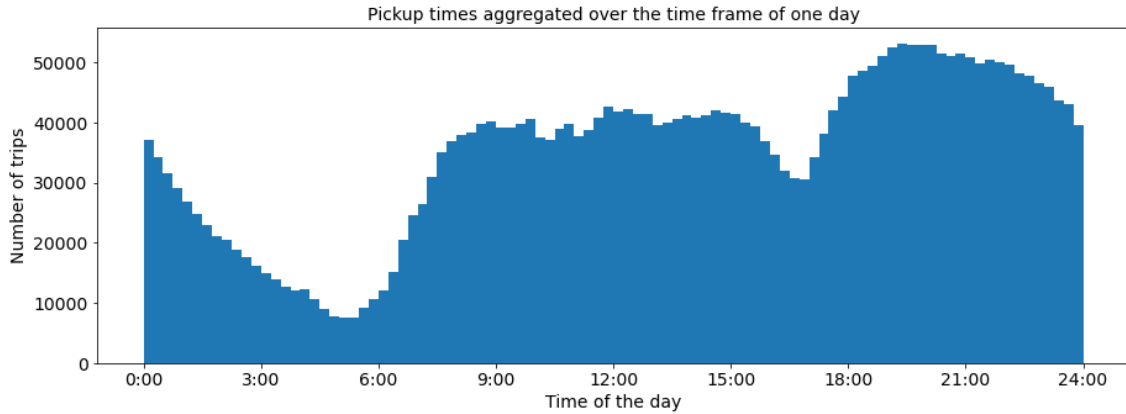
timestamps to emit events in the exact order they occurred. Figure 5.11 shows an excerpt of the finalized taxi trip data with 7 columns containing potentially relevant data for future processing tasks. In the final dataset, each line contains one trip entry while every trip entry has information in its 7 columns about the start and stop times, passenger count, pickup and drop-off coordinates as well as the resulting trip duration and distance. Figure 5.12 visualizes the start times of taxi trips throughout the day, aggregating all 7 days of the chosen week. It is easy to see, that the frequency of trips increases at rush hours and at night. Passenger counts and the number of their occurrences is displayed in figure 5.13, which shows that most trips are with one passenger only. Figures 5.14 through 5.17 show the probability distributions of pickup and dropoff coordinates in degrees longitude and latitude. Pickup and dropoff coordinates with the highest frequencies are right in the center of New York City. The red vertical dashed line marks the coordinates of the Times Square, which means most taxi trips start and / or end there. Further computationally intensive investigation of the dataset shows in table 5.2 that 441,425 taxi trips started from outside Manhattan and ended in Manhattan. 592,360 taxi trips started in Manhattan and ended outside of Manhattan and 623,147 trips started and ended outside of Manhattan. As expected, the majority of rides with 1,724,226 taxi trips started and ended in Manhattan. These data explorations could further be utilized by taxi companies to analyze customer behavior or most profitable car placements. For the purpose of this work a potentially real scenario has been constructed, which searches for car pooling possibilities. Since most rides are with only one customer, half empty cars are clogging the streets. Numerous research findings confirm a malicious impact of traffic jams on the environment, on the experience of any driver as well as on the profit of taxi companies [DG16; JVV04; KD95]. Including these motivations but not limited to them, it is in the interest of both the taxi companies and the customers to share cars.

	pickup_datetime	dropoff_datetime	passenger_count	pickup_latitude	pickup_longitude	dropoff_latitude	dropoff_longitude
0	2013-10-07 00:00:00	2013-10-07 00:18:00	1	40.726711	-74.007347	40.771179	-73.906792
1	2013-10-07 00:00:00	2013-10-07 00:06:00	5	40.732964	-73.998367	40.742638	-74.003563
2	2013-10-07 00:00:00	2013-10-07 00:16:00	5	40.725288	-73.981026	40.702248	-73.928360
3	2013-10-07 00:00:00	2013-10-07 00:07:13	3	40.729404	-73.977974	40.722229	-73.991844
4	2013-10-07 00:00:00	2013-10-07 00:04:00	2	40.783501	-73.950264	40.758293	-73.966110
...	...	...	...	...	...	...	...
3381153	2013-10-13 23:59:58	2013-10-14 00:03:11	2	40.751678	-74.004555	40.749580	-73.999092
3381154	2013-10-13 23:59:58	2013-10-14 00:04:16	1	40.763706	-73.985680	40.758675	-73.992683
3381155	2013-10-13 23:59:59	2013-10-14 00:17:51	3	40.730022	-73.954918	40.748863	-73.991730
3381156	2013-10-13 23:59:59	2013-10-14 00:02:41	1	40.733665	-73.976654	40.733665	-73.976654
3381157	2013-10-13 23:59:59	2013-10-14 00:21:29	1	40.708630	-74.011055	40.687073	-73.976089

3381158 rows × 7 columns

Figure 5.11: Excerpt of the reduced, filtered and sorted NYC Taxi Trip dataset



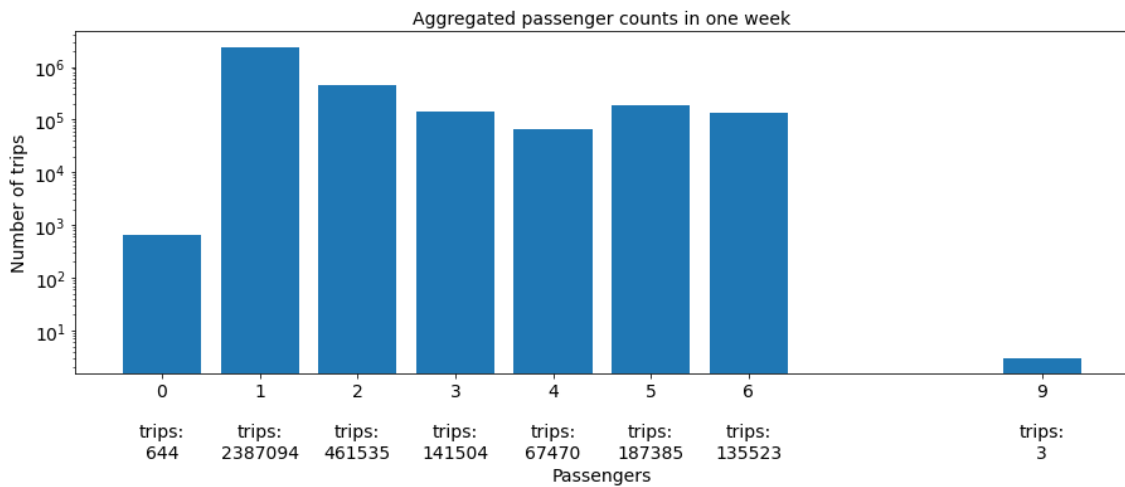


**Figure 5.12:** Daily taxi trip start times aggregated every 15 minutes

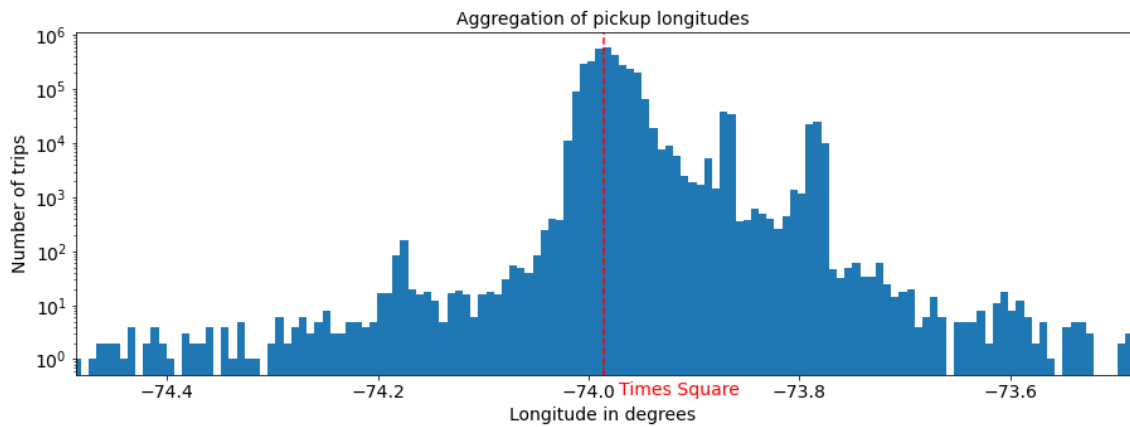
trip origin and destination	O $\rightarrow$ M	M $\rightarrow$ O	O $\rightarrow$ O	M $\rightarrow$ M
number of taxi trips	441,425	592,360	623,147	1,724,226

**Table 5.2:** Taxi trips in Manhattan (M), outside of Manhattan (O) and in between

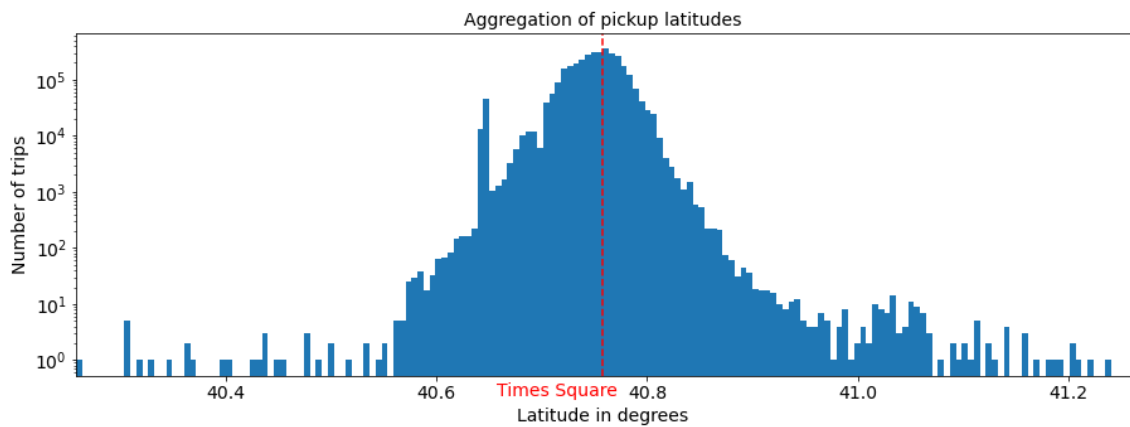
A simple approach to find potential carpooling trips is, to define a taxi trip with one passenger as initial event. Naturally, the next event in the query pattern is another taxi trip with one passenger in close geographic proximity to the initial event. A third taxi trip with one passenger in close proximity to the initial event is optional to refine the search query. The geographic relation of close proximity is assessed with a 2-stage arithmetic operation. First it checks if the difference of the longitudes of pickup location do not exceed a specified limit  $lim_{long}$ . If this resolves to *True*, it checks for the difference of the latitudes of pickup locations to be below a specified limit  $lim_{lat}$ . If this check also resolves to *True*, then the two trips both start within a cuboid with the edge lengths  $2 * lim_{long}$  and  $2 * lim_{lat}$  with the initial event being at the center. If any of the operations resolves to *False*, then the second event in the sequence will further be checked against other



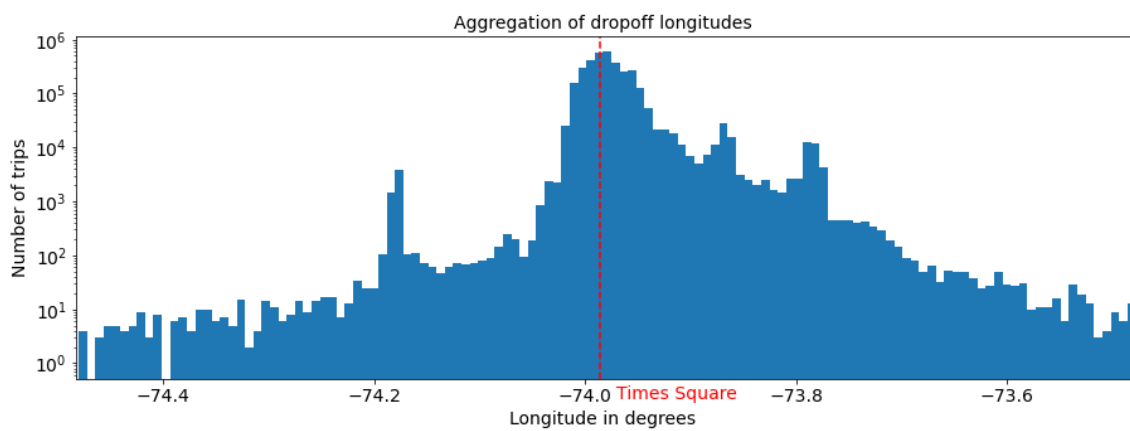
**Figure 5.13:** Aggregated passenger counts within one week



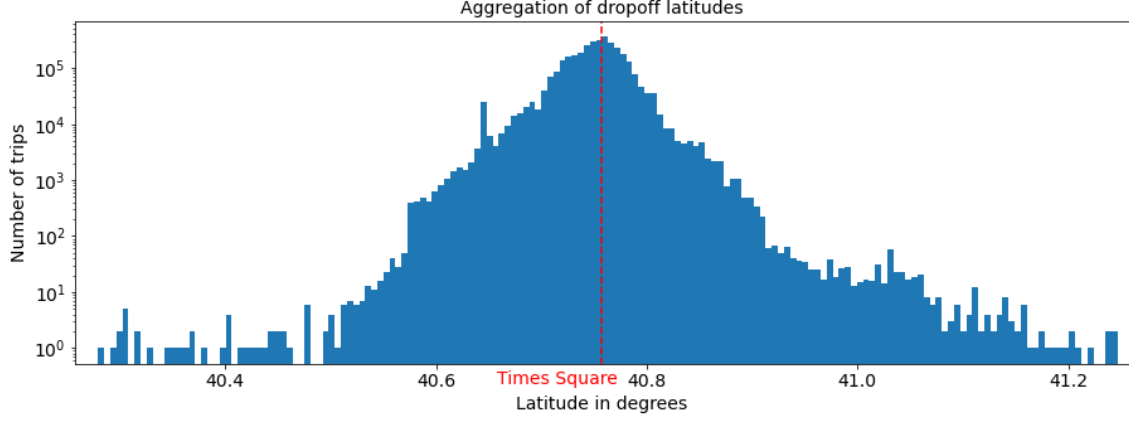
**Figure 5.14:** Aggregated longitudes of taxi trip start locations



**Figure 5.15:** Aggregated latitudes of taxi trip start locations



**Figure 5.16:** Aggregated longitudes of taxi trip destination locations



**Figure 5.17:** Aggregated latitudes of taxi trip destination locations

existing partial matches. In case of a third event getting checked against the start coordinates of the initial event, whereas both checks resolve to *True*, then all three taxi trips start within the specified cuboid with the initial trip at the center. This example is of practical use for customers to find potential carpooling trips in their area, or for taxi companies to act more economically. However, the application query for the NYC taxi dataset used in this thesis differs slightly from this suggested one for experimental reasons. To task and examine the PFS approach under more complex query and event attribute relations, the search query to form a complex event has been adapted for the experiments with the real world taxi dataset. In contrast to the suggested real world application, the search pattern sequence of event types includes multiple event types for the real world data experiments. The arithmetic operation of the PFC to derive thresholds from event attribute values also needs alteration to a window function and produces two threshold values. For the new query pattern and threshold assessment, the assumptions about the CEP application made in section 4.1 change as follows. The event producer has access to the trip entries and assigns the passenger count as the event type  $e[t]$  when forming primitive events  $e$  to be published. This extends the possible event types to integers  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  and allows to implement a search query pattern containing more than 2 event types. Consequently, the query  $q = C_1 \wedge C_2$  has been extended and searches as first condition  $C_1$  for a sequence of event types  $seq(e_1, e_2, e_3, e_4)$  with  $e_1[t] = 1$ ,  $e_2[t] = 2$ ,  $e_3[t] = 3$ ,  $e_4[t] = 4$ , within a time frame of one minute. Partial matches get discarded after a timeout  $to = 60$  s to free the operator from old partial matches. The extension from 10 s to 60 s increases the total number of complex events getting emitted by the operator. The condition  $C_2$  referring to the event attribute value relation is set to  $C_2 = c_{12} \wedge c_{13} \wedge c_{14}$  with

$$c_{12} = |e_1[p][long] - e_2[p][long]| < lim_{long} \wedge |e_1[p][lat] - e_2[p][lat]| < lim_{lat}$$

$$c_{13} = |e_1[p][long] - e_3[p][long]| < lim_{long} \wedge |e_1[p][lat] - e_3[p][lat]| < lim_{lat}$$

$$c_{14} = |e_1[p][long] - e_4[p][long]| < lim_{long} \wedge |e_1[p][lat] - e_4[p][lat]| < lim_{lat}$$

This change in the query makes the CEP application search for taxi trips with one, two, three and four passengers occurring in this exact order. State machines of partial matches only proceed if subsequent event attribute values of pickup longitude and latitude respectively are within a specific proximity to the initiating taxi trip event with one passenger.

## 5.4 Experiment setup

Running actual experiments requires to define previously introduced theoretical parameters like emission rate  $\lambda_{SOURCE}$ , required processing rates  $\lambda_{\omega,opt}$ , timeouts  $to_{\omega}$ , update interval  $freq_{update}$  and the distributions, which are used to fit incoming data within the PFS mechanism. The values assigned to these parameters are discussed in this section as well as all variables which are adjusted between experiment runs.

The different experiments of shedding types all use the same datasets, data producer and operator for complex events processing. The experiment series with random shedding as well as with sole probabilistic feature shedding run on a single machine because the resulting output quality is independent of processing times. Therefore no processing times are recorded for the evaluation in chapter 6. For all experiments with linear program shedding involved, a single machine does not suffice. The calculated load shedder configurations depend on measured operator processing times, whereas calculations for CDFs and corresponding thresholds use extensive computing power on the machine where they are executed. These calculations influence measurements of operator processing times if the bottleneck operator executes on the same machine. To be able to compare experiments depending on processing times, the bottleneck operator runs on another machine. This ensures a decoupling of event processing times on the bottleneck operator from any other computational expenses used for distribution fitting and threshold calculation.

The given CEP shedding framework provides an event producer, which has been adapted to fit the needs of the stated research questions. The event producer publishes events of synthetic datasets with their respective payload in intervals of 7 ms, which results in an emission frequency of 142.857 Hz. For events of the real world NYC taxi dataset the emission intervals change to 5ms, which corresponds to a frequency of 200 Hz. The reason for this adaption is simply to reduce experiment runtimes. One real world data experiment runs for 6.57 hours if executed with emission intervals of 7 ms. With the mentioned adaption, the experiment runtime is reduced to 4.69 hours. The published events are received by the operator's receiving thread and queued for further processing in a working thread. Operators are ultimately responsible for event shedding and therefore receive processing rates as load shedder configurations in case of random shedding and LP shedding, or event attribute thresholds as load shedder configurations in case of PFS. The operator drops the necessary amount of incoming events to either achieve the required processing time, or to process a proportion of all events, which is specified on experiment initiation.

Each shedding type described in section 5.1 results in 8 experiment runs, each with a specified required processing rate  $\lambda_{\omega,opt}$ . These are chosen to be  $\{1.0, 1.0, 1.0, 0.85, 0.65, 0.45, 0.25, 0.05\}$ . The first three runs are the mentioned benchmark runs to receive the number of all possible complex output events in order to assess the output quality. Additionally, they provide an operator benchmark latency  $latency_{\omega,bm}$ , which is used to determine a respective latency bound in LP shedding experiments with and without PFS extension when shedding is applied. The subsequent runs return results presented in chapter 6. Required processing rates are treated differently for the four shedding types. In case of random shedding, they are sent directly to the bottleneck operator. In case of PFS, the application instantiates a PFC with the required processing rate and frequently calculates threshold values from the fitted CDF. These thresholds are sent to the bottleneck operator as shedder configurations. In case of LP shedding with and without PFS extension, the latency measured in the benchmark runs is multiplied by the respective required processing rate. These resulting latency bounds  $latency_{\omega,opt} = latency_{\omega,bm} * \lambda_{\omega,opt}$  serve the LP to calculate processing rates for the

bottleneck operator.

Section 4.1 elucidated the function of a partial match timeout  $to$  for operators. The timeout is chosen as  $to_{\omega,s} = 10$  s for experiments with synthetic datasets. For real world datasets it is increased to  $to_{\omega,rw} = 60$  s to guarantee a higher number of matches and complex events for the final evaluation. A timeout of one minute for a carpooling application described in 5.3 is more reasonable than a 10 second timeout to provide both customers and taxi companies a longer time frame to find matching trips, which optimizes the number of shared cars and the time to wait for customers.

When the PFS extension is active, updated shedder configurations are sent to the bottleneck operator in intervals of  $interval_{update} = 100$  received events or after a timeout  $to_{update} = 5$  s. After  $interval_{update}$  incoming events or after the timeout  $to_{update}$ , the PFC starts its fitting process before calculating new threshold values and sending them to  $\omega$ . The fitting process uses the latest  $n_{fit} = 100$  received event attribute values of each event type to fit its respective distribution and calculate the corresponding CDF. This leads to an update every 700 ms or 500 ms, which can be prolonged if the application allows for slower reaction time and decreased output quality in order to economize computing resource consumption. This update interval has to be prolonged if the amount of event types participating in a query increases.

All experiments with synthetic datasets only fit incoming event attribute values to a normal distribution as their best fit is already known to be a normal distribution. For the real world dataset of New York City taxi trips, the relation between fitting operations and the resulting processing time for fitting, which is further explained in chapter 6, has been used to declare three common probability distributions to fit the incoming event attribute values. The chosen probability distributions are a normal distribution, a gamma distribution and a chi-squared distribution. The reason for this is to cover a spectrum of different probability distribution types. As mentioned before, this selection can be supplemented in case longer update intervals are applicable or they can be exchanged if the data should be checked against other probability distributions.

## 6 Evaluation

This chapter summarizes the outcome of shedding experiment in section 6.1. It assesses output quality and emphasizes the suitability of the PFS mechanism to improve existing shedding mechanisms in section 6.2. In order to assess the output quality of different shedding type experiments, their resulting number of complex events are set into relation to the total number of possible complex events. The amount of possible complex events is determined in benchmark runs when shedding is not activated whereas the output quality is defined in section 4.1 as follows.

$$OQ_{p\_rate,s\_type,data} = \frac{n_{CE,p\_rate,s\_type,data}}{n_{CE,bm,s\_type,data}}$$

The final evaluation is achieved by comparing the output qualities  $OQ_{p\_rate,s\_type,data}$  of the experiments with PFS extensions with the output qualities of corresponding extended shedding mechanisms random shedding and LP shedding. In addition, this chapter compares secondary experiment outcomes like the ratios of required processing rates and measured processing rates as well as the ratios of required processing times to the measured processing times. Further of interest are concerns regarding to scalability of the fitting process, targeting the question how raising numbers of event types and probability distributions impact resulting fitting times. Experiments with real world taxi trip data also resulted in a substantial number of false positive complex events. All mentioned additional results are further clarified in section 6.2.

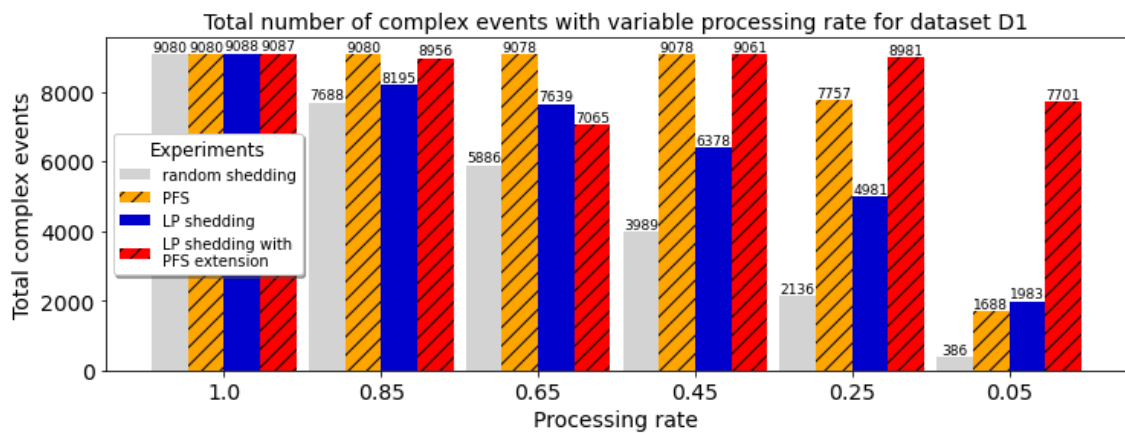
### 6.1 Results

Total complex events received by the sink with active load shedding is the main indicator for output quality, as already stated in section 2.2. The table 6.1 presents the results of shedding experiments introduced in chapter 5. Each row represents the results of all seven datasets executed with one of the determined processing rates and in one of the four shedding type experiments from section 5.1. Columns are sorted from top to bottom in a combination of processing times and shedding types. Overreaching rows are the processing rates of experiments in descending order and within each of the six processing times all four shedding types are listed one below the other. Grey rows highlight the experiment results with active PFS.

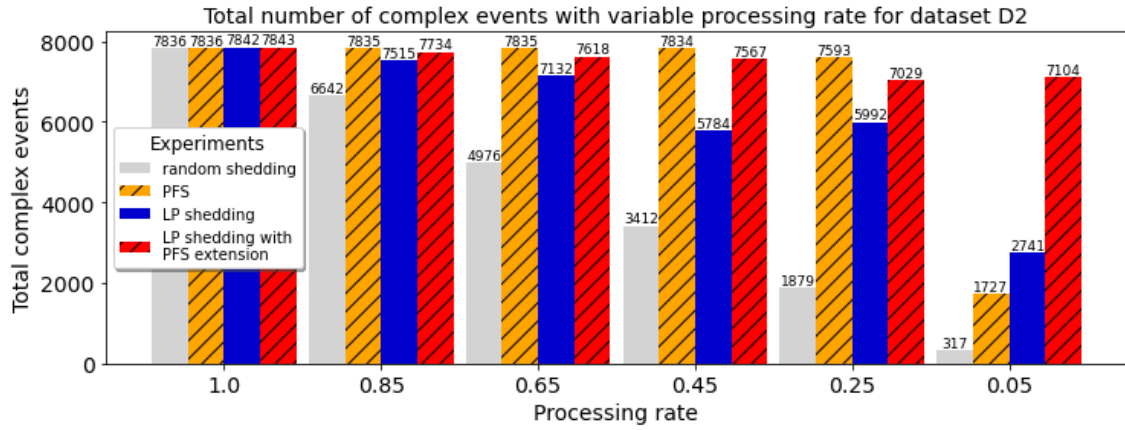
Figures 6.1 through 6.7 show plots of the seven columns of table 6.1. The four different shedding types are aligned next to each other to allow direct comparison of the resulting complex events. Gray and blue shaded bars depict existing shedding mechanisms random shedding and LP shedding, whereas the striped bars display the respective improvement through PFS extension.

processing rate	shedding type	dataset						
		D1	D2	D3	D4	D5	D6	D7
<b>Benchmark 1.0</b>	random	9080	7836	8012	1224	7886	7193	1965
	PFS	9080	7836	8012	1224	7886	7193	1995
	linear program	9088	7842	8017	1226	7897	7195	1980
	LP with PFS ext.	9087	7843	8017	1226	7897	7196	2077
0.85	random	7688	6642	6846	1040	6623	6042	2121
	PFS	9080	7835	8012	1224	7888	7192	26974
	linear program	8195	7515	6924	1147	7351	7044	4849
	LP with PFS ext.	8956	7734	8016	1226	7637	5351	16574
0.65	random	5886	4976	5080	774	5073	4603	1832
	PFS	9078	7835	8012	1224	7887	7192	31861
	linear program	7639	7132	5870	1065	6992	6271	9974
	LP with PFS ext.	7065	7618	7751	1223	7576	6122	19950
0.45	random	3989	3412	3464	519	3441	3220	1240
	PFS	9078	7834	8011	1223	7882	7189	23679
	linear program	6378	5784	4690	953	5902	4278	20048
	LP with PFS ext.	9061	7567	8015	1223	7228	7182	24097
0.25	random	2136	1879	1908	245	1856	1602	475
	PFS	7757	7593	7695	1223	7136	6177	28107
	linear program	4981	5992	3080	696	5161	3581	35191
	LP with PFS ext.	8981	7029	8001	1226	7325	7184	40260
0.05	random	386	317	286	31	306	316	83
	PFS	1688	1727	1740	1190	1760	1624	3936
	linear program	1983	2741	1261	290	1945	1613	62685
	LP with PFS ext.	7701	7104	6864	1226	6766	6130	40816

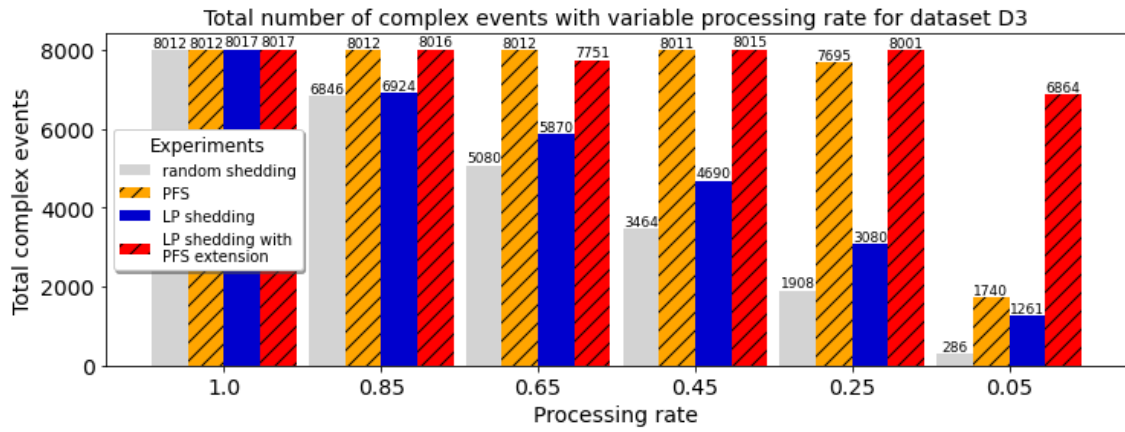
**Table 6.1:** Total number of complex events for variable shedding types, processing rates und datasets



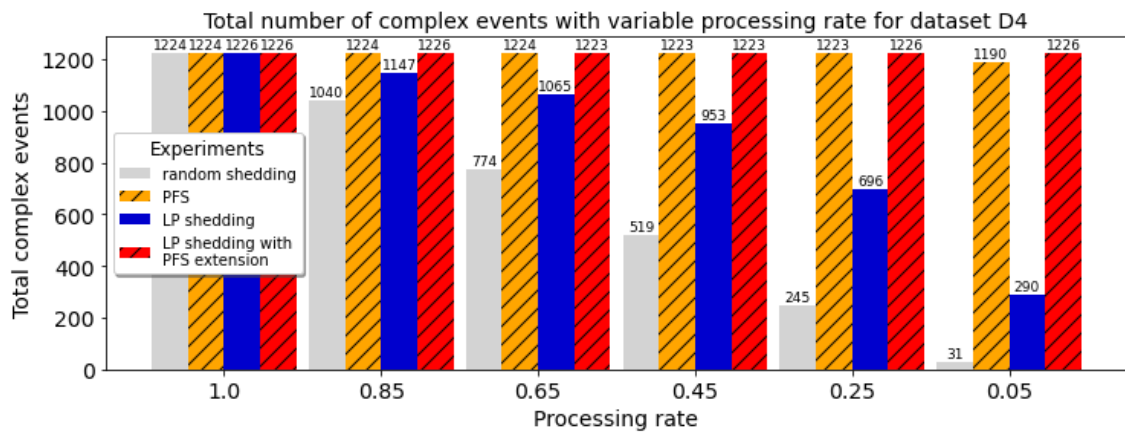
**Figure 6.1:** Total number of complex events for all experiments with dataset D1



**Figure 6.2:** Total number of complex events for all experiments with dataset D2

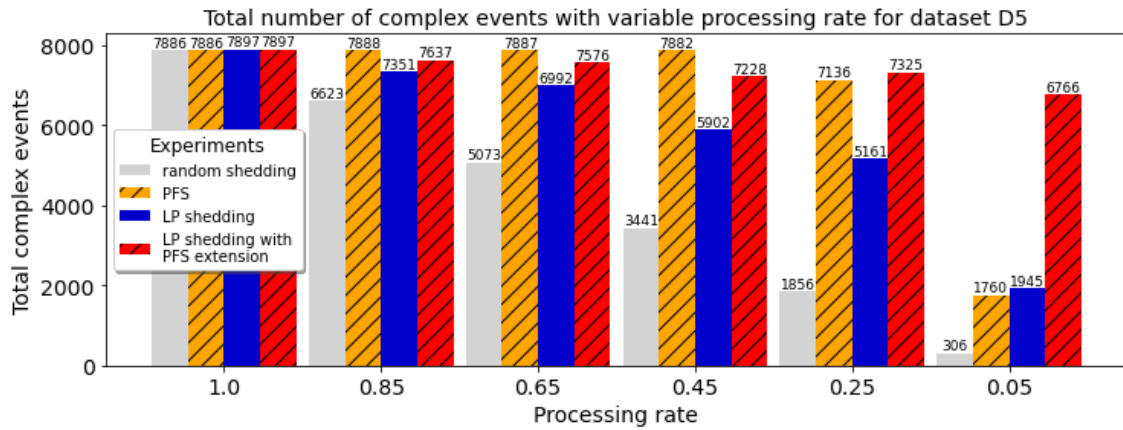


**Figure 6.3:** Total number of complex events for all experiments with dataset D3

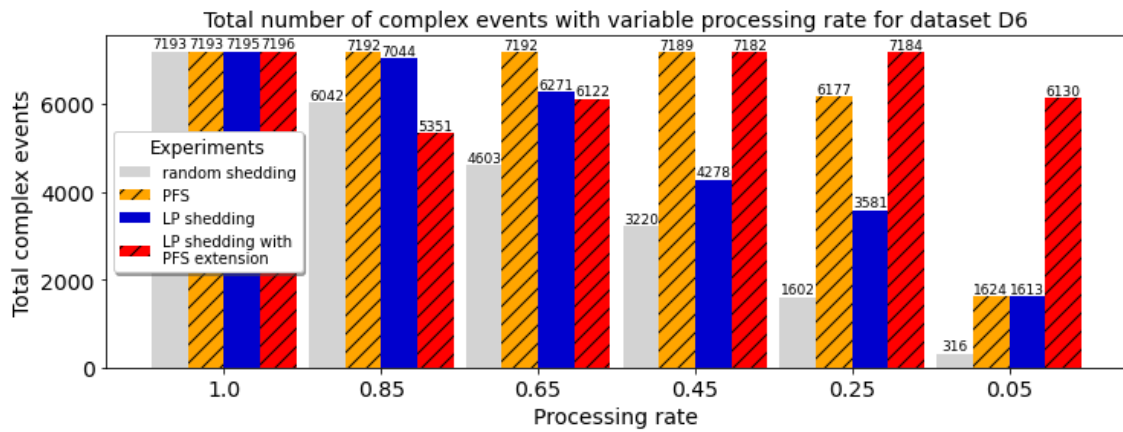


**Figure 6.4:** Total number of complex events for all experiments with dataset D4

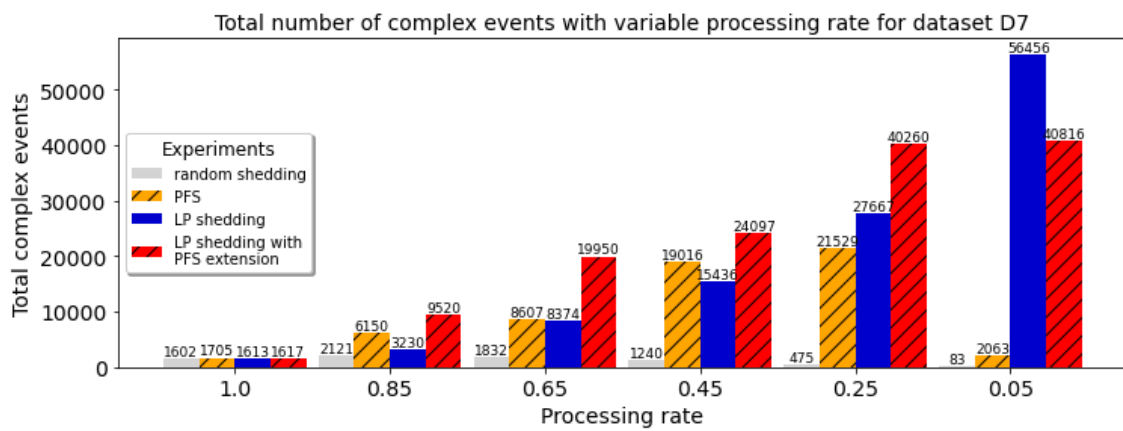




**Figure 6.5:** Total number of complex events for all experiments with dataset D5



**Figure 6.6:** Total number of complex events for all experiments with dataset D6



**Figure 6.7:** Total number of complex events for all experiments with dataset D7

## 6.2 Discussion

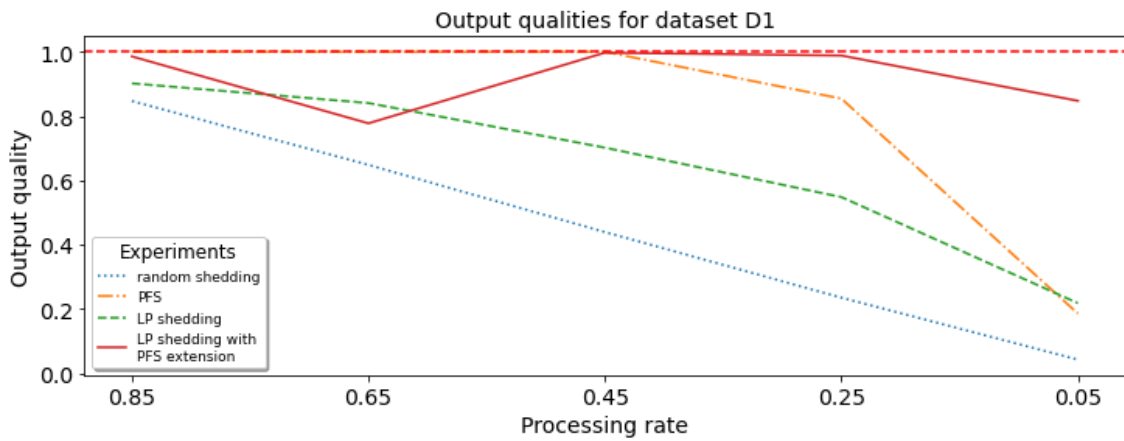
In all experiments but one, the number of complex events from experiments with the PFS extension exceed the produced complex events of each corresponding basic shedding mechanism, which are random shedding and LP shedding. Especially when processing rates decrease, the benefit gained through PFS becomes even more obvious. This outcome confirms the expected behavior of the PFS mechanism, which selectively drops events with less relevant payload. Especially with low processing rates, the few remaining events to be processed need to be relevant for the search query. This can be achieved through the PFS extension. Interestingly, dataset D4 seen in 6.4 shows a very high number of complex events throughout all processing rates for the PFS extension. This behavior can be explained in the same way as the efficacy of PFS for low processing rates. If there is a low number of complex events to start with, the importance of processing the few relevant primitive events becomes even more visible.

The discussed plots of experiment results show on a first glimpse, that a PFS extension improves output quality significantly if the query is structured to relate attribute values of incoming events, as defined for this thesis. Table 6.2 shows the calculated output qualities for all resulting complex events presented in table 6.1. Values below 1 depict a decrease in output quality compared to benchmark experiments, whereas values higher than 1 insicate false positives, which are further discussed later in this section. Figures 6.8 through 6.14 show the corresponding diagrams for better visualization. Unexpected results for LP shedding with PFS extension are seen in figure 6.8 as a drop in output quality for a processing rate of  $\lambda_{\omega,opt} = 0.65$  as well as in figure 6.13 as a low start of output quality for processing rates 0.85 and 0.65. Explanations for this behavior are faulty experiment runs, since these are the only occurrences of low output qualities for high processing rates. The output quality of the real world dataset D7 with a processing rate of 0.05 displays another unexpected result. Figure 6.14 shows a spike in output quality, even in comparison to the PFS extension. The reason for this observation is obscure, but necessary investigations to find an explanation exceeded the time limit of this thesis. A possible reason is a bug in the PFS implementation, which stayed undetected until the end of all experiments. Another possible reason is, that indeterministic shedding decisions of LP shedding combined with beneficial event type ratios in some way facilitate the occurrence of pattern matches. This behavior is a candidate for future research in the topic of this project.

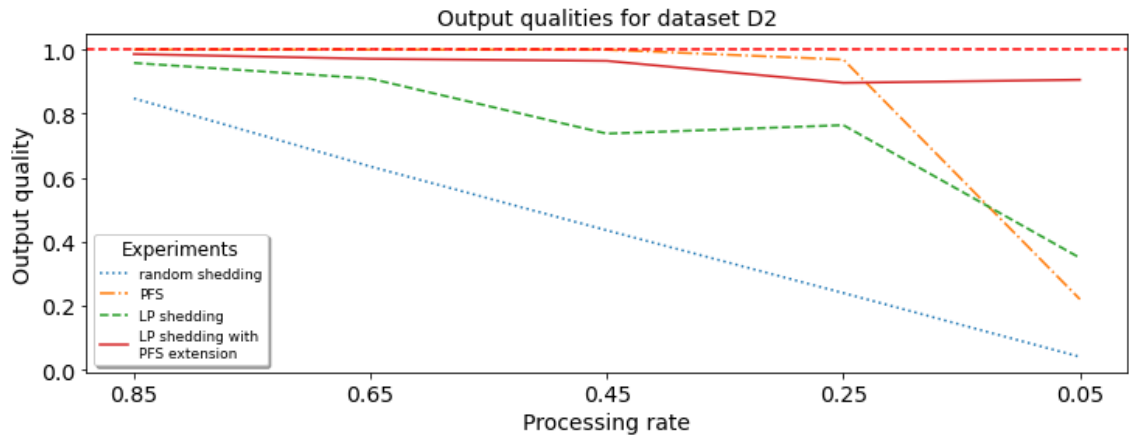
Figures A.1 through A.7, located in the appendix, visualize the number of false negatives for each experiment with decreasing processing rate. As described in section 2.2, false negatives emerge from missing complex events when shedding is active in comparison to the benchmark run. High numbers of false negatives imply a low output quality. The opposite to false negatives are false positives, which are additional complex events not present in the benchmark run, but occurring due to shedding activity. Only in the real world dataset D7 a significant number of false positives emerged when running the experiments. Figure 6.15 shows the numbers of resulting false positive complex events for different shedding types and processing rates in experiments with dataset D7. A reason for more false positive complex events in this experiment than in the other experiments is, that the search query is more sophisticated by involving four instead of two event types. The relation  $\zeta$  of total primitive events  $n_{total}$  as operator input to the total number of complex events in the benchmark runs  $n_{CE,bm}$  as operator output gives further insight. The ratio  $\zeta = \frac{n_{total}}{n_{CE}}$  is much lower in experiments with datasets D1-D6 than it is with dataset D7.

processing rate	shedding type	dataset						
		D1	D2	D3	D4	D5	D6	D7
0.85	random	0.847	0.848	0.854	0.85	0.84	0.84	1.324
	PFS	1.000	1.000	1.000	1.000	1.000	1.000	3.607
	linear program	0.902	0.958	0.864	0.936	0.931	0.979	2.002
	LP with PFS ext.	0.986	0.986	1.000	1.000	0.967	0.744	5.887
0.65	random	0.648	0.635	0.634	0.632	0.643	0.640	1.144
	PFS	1.000	1.000	1.000	1.000	1.000	1.000	5.048
	linear program	0.841	0.909	0.732	0.869	0.885	0.872	5.192
	LP with PFS ext.	0.777	0.971	0.967	0.998	0.959	0.851	12.338
0.45	random	0.439	0.435	0.432	0.424	0.436	0.448	0.774
	PFS	1.000	1.000	1.000	0.999	0.999	0.999	11.153
	linear program	0.702	0.738	0.585	0.777	0.747	0.595	9.570
	LP with PFS ext.	0.997	0.965	1.000	0.998	0.915	0.998	14.902
0.25	random	0.235	0.240	0.238	0.200	0.235	0.223	0.297
	PFS	0.854	0.969	0.960	0.999	0.905	0.859	12.627
	linear program	0.548	0.764	0.384	0.568	0.654	0.498	17.153
	LP with PFS ext.	0.988	0.896	0.998	1.000	0.928	0.998	24.898
0.05	random	0.043	0.040	0.036	0.025	0.039	0.044	0.052
	PFS	0.186	0.220	0.217	0.972	0.223	0.226	1.210
	linear program	0.218	0.350	0.157	0.237	0.246	0.224	35.001
	LP with PFS ext.	0.847	0.906	0.856	1.000	0.857	0.852	25.242

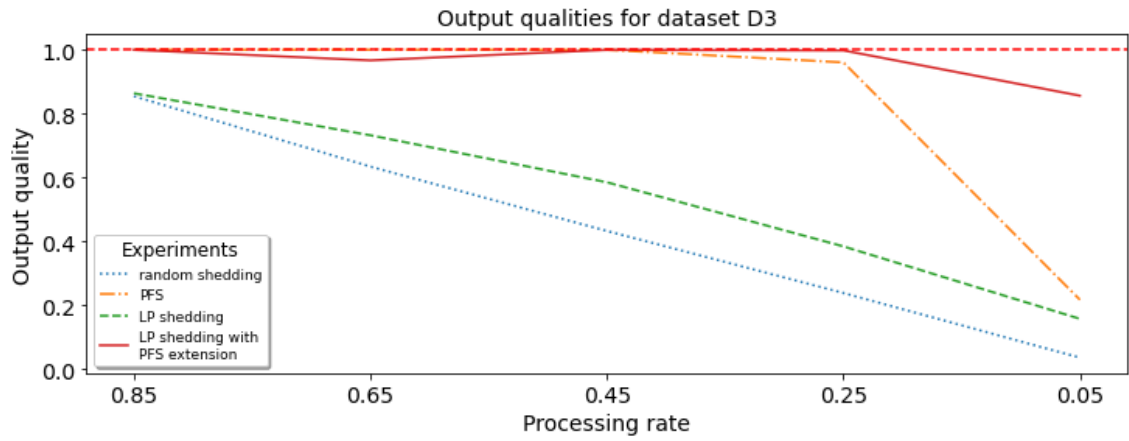
**Table 6.2:** Resulting output quality for variable shedding types, processing rates und datasets



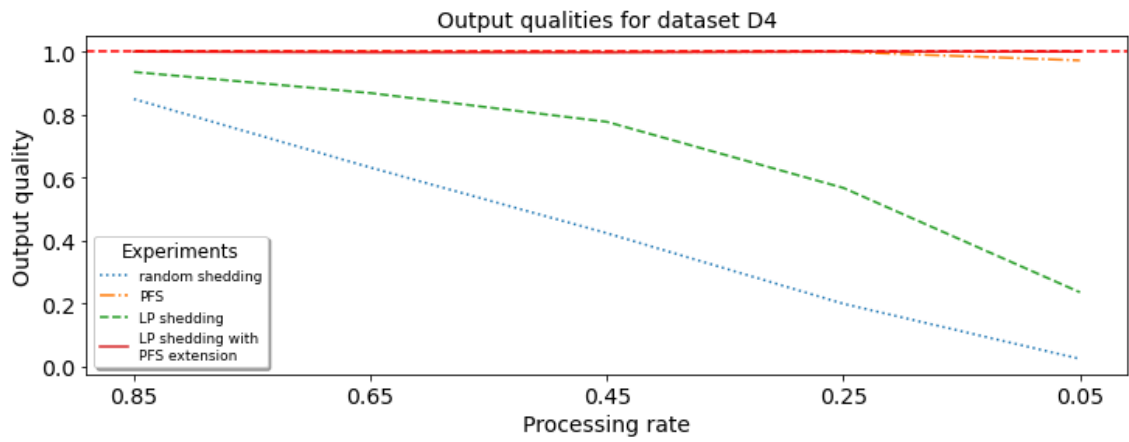
**Figure 6.8:** Calculated output qualities for all experiments with dataset D1



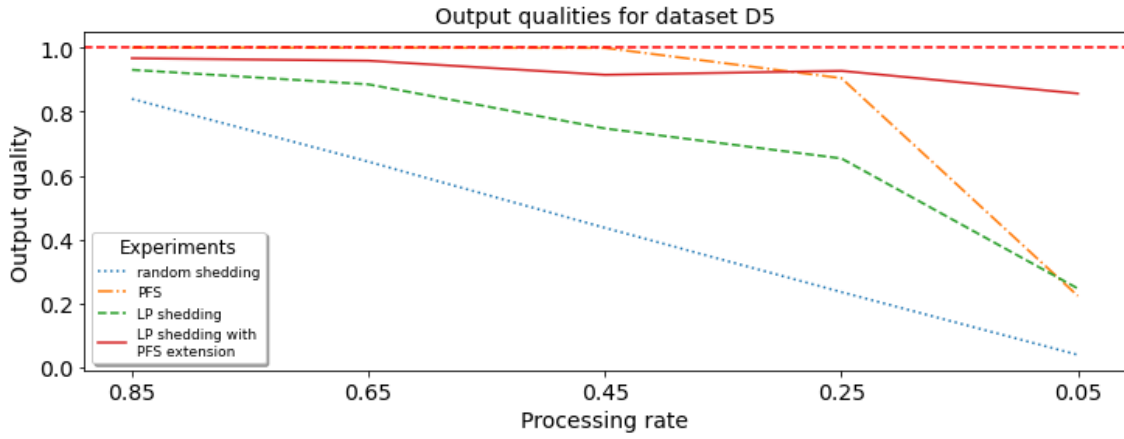
**Figure 6.9:** Calculated output qualities for all experiments with dataset D2



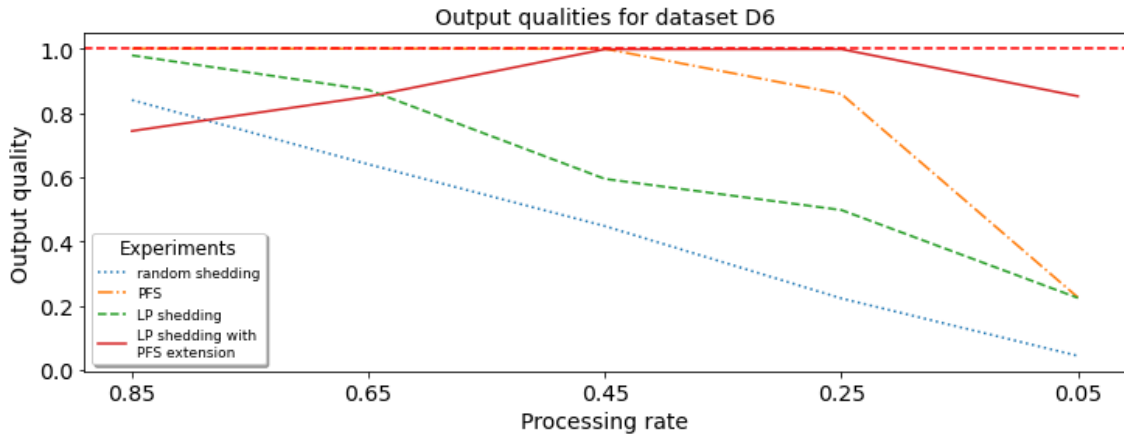
**Figure 6.10:** Calculated output qualities for all experiments with dataset D3



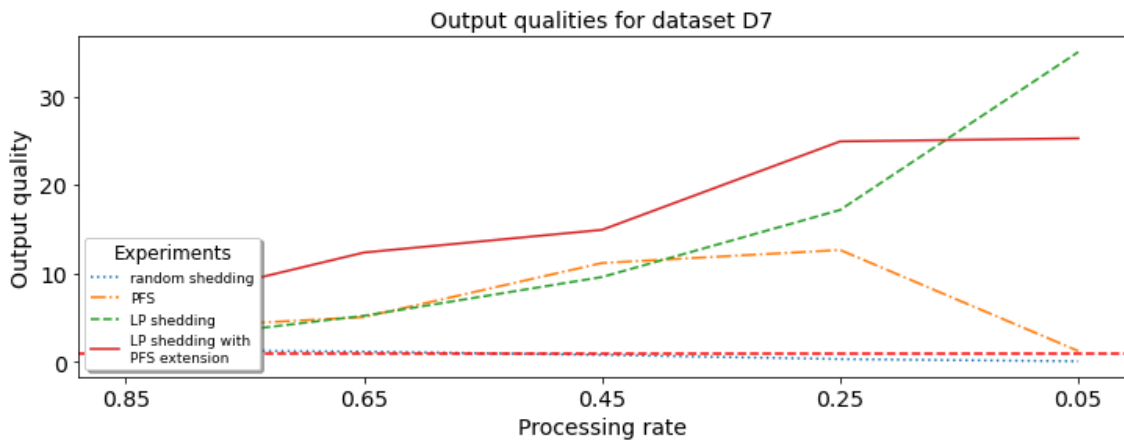
**Figure 6.11:** Calculated output qualities for all experiments with dataset D4



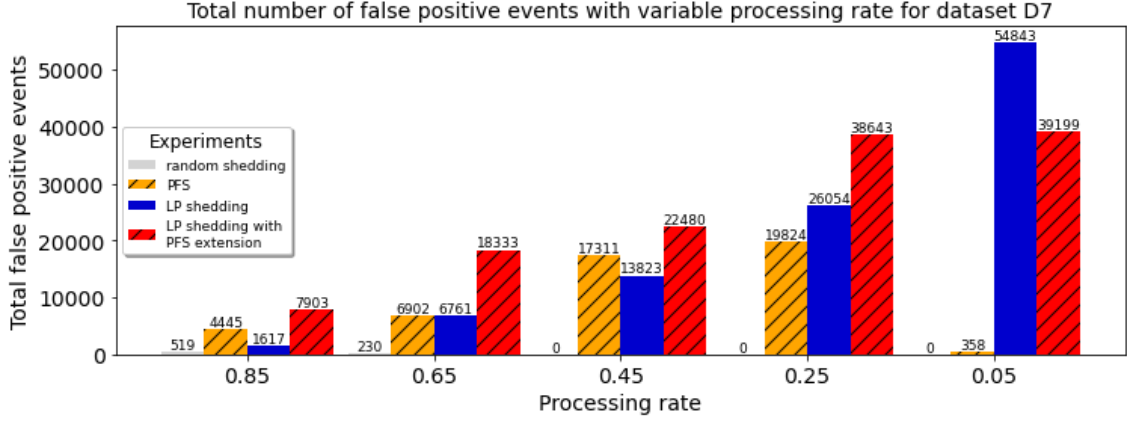
**Figure 6.12:** Calculated output qualities for all experiments with dataset D5



**Figure 6.13:** Calculated output qualities for all experiments with dataset D6



**Figure 6.14:** Calculated output qualities for all experiments with dataset D7



**Figure 6.15:** Total number of false positives for all experiments with dataset D7

For example compare  $\zeta_{D1} = \frac{60,000}{9088} = 6.6$  and  $\zeta_{D7} = \frac{3,381,158}{2077} = 1,527.9$ . This ratio implies, that in average every 7th primitive event in dataset D1 leads to a complex event, whereas averagely every 1,528th primitive event leads to a complex event when running the car sharing query on the real world taxi dataset D7. This implies many partial matches with 1,2 or 3 events already waiting for completion will expire eventually, but clutter the operator until getting discarded. In the case that events, which are irrelevant due to their event type or payload values, are shed before being checked against all existing partial matches and unnecessarily consuming computing resources, higher numbers of complex events can occur. The reason is, other events with higher matching probability do get the chance to complete a partial match before it expires and gets discarded. Another reason for false positives is a longer timeout before partial matches are discarded. The explanation is the same as already stated. Irrelevant events are shed instead of consuming computing resources and clogging the operator, therefore relevant events get processed earlier and complete partial matches which expired in the benchmark runs.

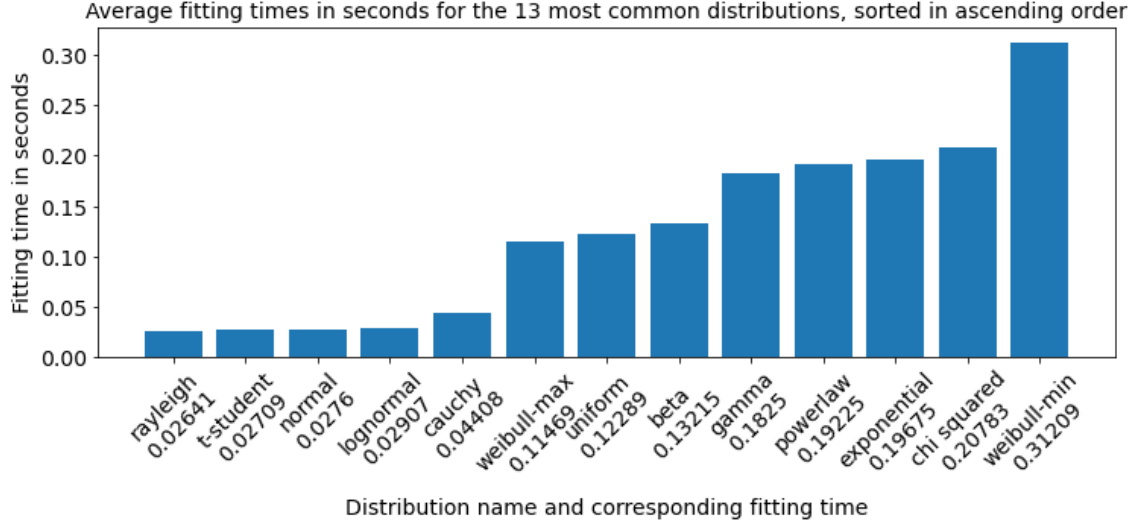
There are secondary factors of influence to be considered next to the number of complex events in order to evaluate experiment results. A high output quality is only valuable if it is not at cost of the original intention, which is a reduction of processing time per incoming event. This is the motivation to start load shedding in the first place and must not be omitted. Therefore, processing rates of all four shedding type experiments and processing times of LP shedding with and without PFS extension have been measured. Figures A.8 through A.14 located in the appendix show the actually achieved processing rate in relation to the required processing rates  $\lambda_{\omega,opt} \{0.85, 0.65, 0.45, 0.25, 0.05\}$  for all seven datasets. The red dashed horizontal line displays the optimal relation of 1.0, where values above this line shed too many events and values below reveal insufficient shedding in order to reach the respective required processing rate. The diagrams show a decreasing ratio for LP shedding with and without PFS extension towards low processing rates, but not for random shedding with and without PFS extension. It is reasonable for random shedding to meet the processing rate requirements, for it can just process the correct proportion of incoming events, not considering any attributes. The extension with PFS has a slightly worse performance, since processing rates are reached through calculated probability distributions and payload value predictions based on these probabilities. Nonetheless, the resulting processing rate ratios show that the PFS mechanism works as desired and behaves closely to the base shedding mechanism. Low ratios for LP shedding as well as its PFS extension depict the difficulty of load shedding with an LP mechanism. The

similarly behaving PFS extension shows the efficacy of PFS once again, because even for the same inability to reach the required rates, the output quality of the PFS extension is higher than for plain LP shedding. Figures A.15 through A.21 in the appendix show the equivalent plots of measured processing times in relation to the required processing times to reach the anticipated processing rate  $\lambda_{\omega, opt}$ . Required processing times represent the latency bound  $latency_{\omega, opt}$  of the bottleneck operator. Values above the red dashed horizontal line point out, that the measured processing time is lower than the required processing time, which is a desirable relation. Values below the line reveal an average processing time higher than the defined latency bound. The results show a decreasing ratio for LP shedding with and without PFS extension towards low processing rates. This means that the given latency bounds are not reached for high shedding ratios in both cases. Nonetheless, the diagrams show a slightly better performance of LP shedding in reaching the latency bound. It has to be decided individually for each application case, if this behavior invalidates the improvement in output quality.

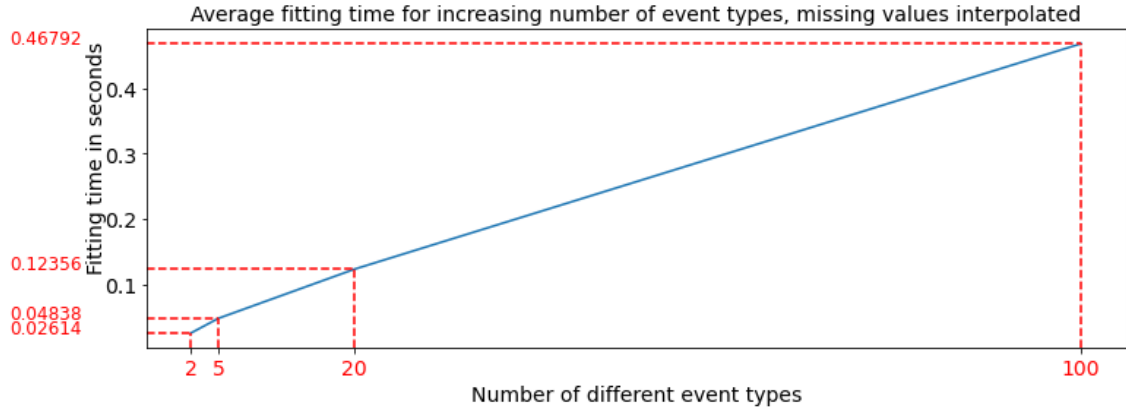
The PFC, introduced in section 4.3, runs in a separate process to execute calculations in short intervals and will inevitably lead to higher computational workload on the central machine. In order to make estimations about scalability, a series of simulation experiments collect data of the actual fitting process. These experiments consider on one side variable numbers of predefined distributions to fit and on the other side a variable number of event types to execute the fitting process. For the first case, each distribution is analyzed on itself, then in combination with other. It is expected that different distribution fitting processes have different computational requirements and it is desirable that they build upon each other in a linear manner. For the second case, a linearly increasing distribution fitting processing time for increasing numbers of event types is desired. The results of these simulations show the fitting process for multiple distribution types linearly increasing with the number of possible distributions. They demonstrate a slight deviation in computing time of fitting operations with different distribution types, some fitting processes seem to have a higher consumption of computational resources. For example, fitting an exponential distribution takes longer than fitting a gamma distribution, which again takes longer than fitting a normal distribution. Figure 6.16 aligns the fitting times for 2 event attribute values in 13 different distributions, which are considered in the course of this work for scalability.

Simulations with variable numbers of event types in the fitting process within the PFC confirm, that fitting times are linear in the number of event types with dependent attributes. Figure 6.17 shows the fitting times for 2, 5, 20 and 100 different event types with interpolated values in between, which illustrates their linearity.

Messaging overhead is an influential parameter which also has to be considered, when discussing scalability. The PFC sends frequent threshold update messages to all bottleneck operators. Each update message contains one or two threshold values for every event type and pattern, which is searched for by the operator. This means the overhead produced through message size is in linear dependency to the number of patterns implemented in the bottleneck operator as well as to the number of event types being part of the implemented patterns. Regarding to the number of messages the messaging system has to cope with, each bottleneck operator receives one update message in defined intervals. This puts the overhead produced by message quantity in a linear relation to the number of bottleneck operators. For these reasons, message overhead as an influential factor on efficacy of a PFS mechanism was not further considered within this work.



**Figure 6.16:** Ordered average fitting times for the chosen 13 distributions, fitting of 2 event types



**Figure 6.17:** Average fitting times for 2, 5, 20 and 100 different event types to be fitted

When executing the LP shedding experiments, the average processing times varied strongly between the different datasets. This variance could be traced back to the event type ratios  $\gamma_{\omega, type}$  of the datasets, with D2 and D3 having the biggest differences between event type ratios. These two datasets consequently show the highest deviation in average processing times with 0.002 s for D2 and 0.000,5 s for D3 in benchmark runs, compared to the benchmark runs on the other synthetic datasets D1, D4, D5 and D6. As shown in figures A.23 and A.24, the average processing time of a single event in experiments with dataset D2 is two times longer than in dataset D1 or D4 (see figures A.22 and A.25 as reference). This observation can be explained with a much higher number of type 0 events in dataset D2. Since every incoming type 0 event starts a state machine, the operator creates a substantial higher number of partial matches. In turn, each incoming type 1 event gets considered for every existing partial match as a potential match. This higher number of partial matches increases the average processing times for dataset D2. Similarly, the average



processing times for dataset D3 are accordingly lower because less partial matches are present from type 0 events to check against. These variations need to be taken into account when setting required processing times for datasets D2 and D3 with active shedding.

## 6.3 Limitations

This chapter's considerations of secondary factors introduce some of the limitations of this work's contribution. One limitation is the already discussed computing overhead and processing time of fitting data in distributions. There are limits regarding to the number of event types or distribution types to fit, depending on the update frequency and the necessity of actuality of shedding configurations. The application can run into timing issues if each fitting process takes up considerable high computing resources. This can happen because the query searches for a high number of event types or many different distribution types. When an operator becomes a bottleneck, response times need to be shortened for the application to behave properly. In this case, the PFS approach needs further research and adaptations. To guarantee short response times, there has to be a limit of event types and distributions, depending on the maximum tolerable response time. The limits of event types and distributions depend on each other and can be solved by linear equations.

Another limitation, which has not been addressed within this work, is the actual number of false negatives and false positives within experiments. The presented results in section 6.1 do simply state the difference of complex events registered in benchmark runs and the complex events when shedding is active. Assumptions made in section 4.1 equally lead to limitations. The cost of reading event attribute values is assumed to be small enough to be ignored. In the case that payload values are not accessible as easy and cheap as assumed, this extra computational effort needs to be considered when evaluating experiment results and efficacy of the shedding mechanism. The assumption that only one operator exists in the implemented operator graph makes this work leave out scenarios with multiple operators. In theory, the extension is able to improve existing shedding mechanism for multiple operators just as efficiently as it does with a single operator. This thesis' scope does not provide proof for that hypothesis.

The limitations close up this chapter, which evaluates the PFS extension by the results taken from numerous experiments. The following chapter 7 concludes the findings of this work and presents possible future topics in this field of research.

## 7 Conclusion and Outlook

In chapter 6, the results of shedding experiments with and without the PFS extension are presented and discussed in detail. This chapter concludes the findings in this work. Furthermore, the assumptions from 4.1 lead to a number of limitations in this work, which can be used as basis for future work and further research.

In chapter 3, the analysis of existing CEP shedding mechanisms led to questions, which defined the course of this work significantly. Chapter 4 describes an approach to assess the importance of single events on the basis of dependent attribute values. It also introduces an algorithm to trace the values of dependent attributes and determine probability density function parameters, which enables the application to assign probabilities to event attribute values. The PFS mechanism links cumulative distribution functions of event attribute values with processing rates to find threshold values, which serve as a margin for shedding decisions. An architecture proposal in section 4.3 places the PFC in a given LP shedding architecture to extend its functionality and improve overall application output quality for an existing shedding mechanisms. Finally, section 6.2 introduces metrics like output quality and ratios between processing times and rates to testify efficacy of the PFS extension addressed in this work. The findings in chapters 5 and 6 are direct resolutions of the stated research questions in section 3.2 and confirm the hypothesis, that existing CEP load shedding mechanisms will receive an output quality improvement when extended with the PFS mechanism from section 4.3. However, this improvement is linked to a relationship of dependent event attribute values in the search query of the CEP application. The effectiveness is most prominent when complex events are scarce or the intended processing rate  $\lambda_{\omega, opt}$  is low.

Future research in probabilistic feature shedding mechanisms may pick up at the stated limitations in section 6.3. An implementation of the algorithm for multi-operator applications and assessment of its efficacy is a valuable advancement. Furthermore, it would be of interest to investigate mechanisms to calculate an optimal number of distributions the algorithm uses for fitting processes depending on the search query, current shedding performance and response time requirements of the application.

Tracing event identifications to determine the correct numbers of both, false negatives and false positives in complex events, will lead to a more transparent evaluation of experiment results.

The algorithm itself could be improved by determining a number of threshold values for each event type in order to categorize incoming events into multiple matching probability areas. With this improvement, the application might not need the algorithm to determine a new set of thresholds for every change in workload. Changing processing rates due to sinking or raising latency bound violations could be handled by using different previously calculated matching probability areas to make shedding decisions.

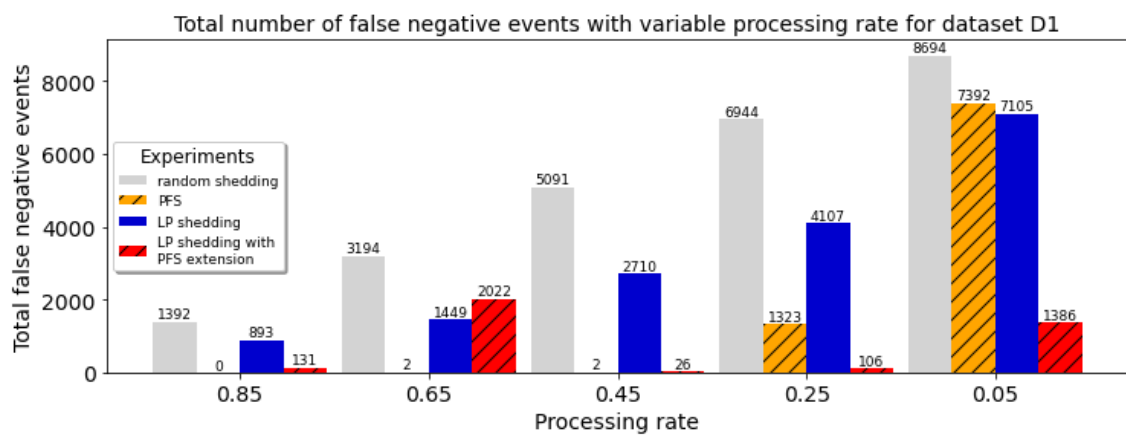
## Bibliography

- [Apa] Apache Software Foundation. *Apache Kafka*. <https://kafka.apache.org/>. Accessed: 2021-10-01 (cit. on p. 15).
- [Bas07] T. Bass. “Mythbusters: Event Stream Processing versus Complex Event Processing”. In: *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*. DEBS ’07. Toronto, Ontario, Canada: Association for Computing Machinery, 2007, p. 1. ISBN: 9781595936653. DOI: [10.1145/1266894.1266896](https://doi.org/10.1145/1266894.1266896). URL: <https://doi.org/10.1145/1266894.1266896> (cit. on p. 16).
- [BDM04] B. Babcock, M. Datar, R. Motwani. “Load shedding for aggregation queries over data streams”. In: *Proceedings. 20th International Conference on Data Engineering*. 2004, pp. 350–361. DOI: [10.1109/ICDE.2004.1320010](https://doi.org/10.1109/ICDE.2004.1320010) (cit. on p. 16).
- [BGH06] R. S. Barga, J. Goldstein, M. Ali, M. Hong. “Consistent streaming through time: A vision for event stream processing”. In: *arXiv preprint cs/0612115* (2006) (cit. on p. 17).
- [BK09] A. Buchmann, B. Koldehofe. “Complex Event Processing:” in: 51.5 (2009), pp. 241–242. DOI: [doi:10.1524/itit.2009.9058](https://doi.org/10.1524/itit.2009.9058). URL: <https://doi.org/10.1524/itit.2009.9058> (cit. on p. 10).
- [Chr] Chris Whong. *FOILing NYC’s Taxi Trip Data*. [https://chriswhong.com/open-data/foil\\_nyc\\_taxi/](https://chriswhong.com/open-data/foil_nyc_taxi/). Accessed: 2021-08-12 (cit. on p. 37).
- [DG16] P. Delhomme, A. Gheorghiu. “Comparing French carpoolers and non-carpoolers: Which factors contribute the most to carpooling?” In: *Transportation Research Part D: Transport and Environment* 42 (2016), pp. 1–15 (cit. on p. 39).
- [GWC+06] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, G. Anderson. “SASE: Complex event processing over streams”. In: *arXiv preprint cs/0612128* (2006) (cit. on p. 17).
- [HBN13] Y. He, S. Barman, J. F. Naughton. *On Load Shedding in Complex Event Processing*. 2013. arXiv: [1312.4283](https://arxiv.org/abs/1312.4283) [cs.DB] (cit. on pp. 10, 11, 13, 16).
- [JKD+15] S. Jayasekara, S. Kannangara, T. Dahanayakage, I. Ranawaka, S. Perera, V. Nanayakkara. “Wihidum: Distributed complex event processing”. In: *Journal of Parallel and Distributed Computing* 79-80 (2015). Special Issue on Scalable Systems for Big Data Management and Analytics, pp. 42–51. ISSN: 0743-7315. DOI: <https://doi.org/10.1016/j.jpdc.2015.03.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0743731515000519> (cit. on p. 10).
- [JVV04] J. A. Joireman, P. A. Van Lange, M. Van Vugt. “Who cares about the environmental impact of cars? Those with an eye toward the future”. In: *Environment and behavior* 36.2 (2004), pp. 187–206 (cit. on p. 39).
- [KD95] A. R. Kearney, R. De Young. “A knowledge-based intervention for promoting carpooling”. In: *Environment and Behavior* 27.5 (1995), pp. 650–678 (cit. on p. 39).

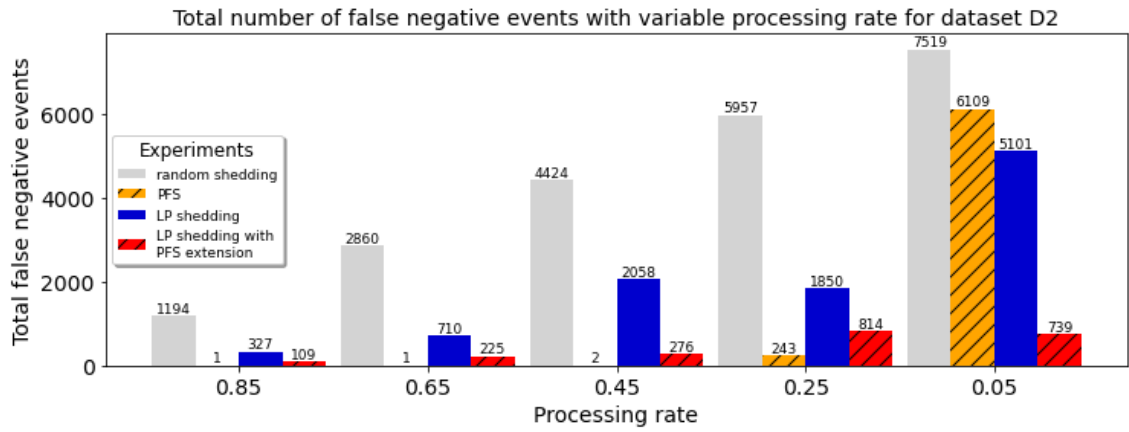
- [RM19] H. Röger, R. Mayer. “A comprehensive survey on parallelization and elasticity in stream processing”. In: *ACM Computing Surveys (CSUR)* 52.2 (2019), pp. 1–37 (cit. on p. 10).
- [Rob10] D. Robins. “Complex event processing”. In: *Second International Workshop on Education Technology and Computer Science. Wuhan*. Citeseer. 2010, pp. 1–10 (cit. on pp. 10, 17).
- [SBFR19] A. Slo, S. Bhowmik, A. Flaig, K. Rothermel. “pSPICE: partial match shedding for complex event processing”. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 372–382 (cit. on pp. 10–12, 16).
- [SBR19] A. Slo, S. Bhowmik, K. Rothermel. “espice: Probabilistic load shedding from input event streams in complex event processing”. In: *Proceedings of the 20th International Middleware Conference*. 2019, pp. 215–227 (cit. on pp. 10–12, 16).
- [SBR20] A. Slo, S. Bhowmik, K. Rothermel. “hSPICE: state-aware event shedding in complex event processing”. In: *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. 2020, pp. 109–120 (cit. on pp. 10, 11, 13).
- [TÇZ+03] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, M. Stonebraker. “- Load Shedding in a Data Stream Manager\*”. In: *Proceedings 2003 VLDB Conference*. Ed. by J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, A. Heuer. San Francisco: Morgan Kaufmann, 2003, pp. 309–320. ISBN: 978-0-12-722442-8. DOI: <https://doi.org/10.1016/B978-012722442-8/50035-5>. URL: <https://www.sciencedirect.com/science/article/pii/B9780127224428500355> (cit. on p. 16).
- [TÇZ07] N. Tatbul, U. Çetintemel, S. Zdonik. “Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB '07*. Vienna, Austria: VLDB Endowment, 2007, pp. 159–170. ISBN: 9781595936493 (cit. on pp. 10, 12, 16).
- [ZVW20] B. Zhao, N.Q. Viet Hung, M. Weidlich. “Load Shedding for Complex Event Processing: Input-based and State-based Techniques”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1093–1104. DOI: [10.1109/ICDE48307.2020.00099](https://doi.org/10.1109/ICDE48307.2020.00099) (cit. on pp. 10, 11, 13, 16).

All links were last followed on October 25th 2021.

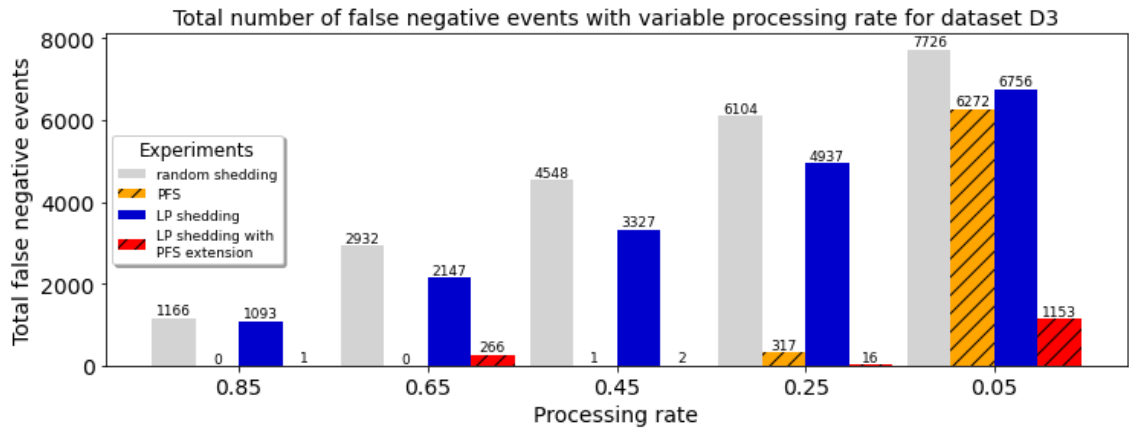
## A Diagrams



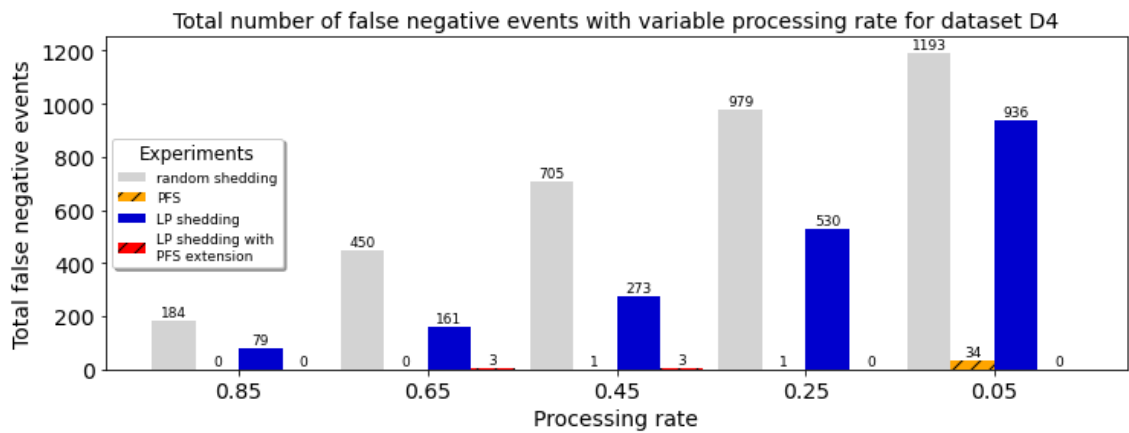
**Figure A.1:** Total number of false negatives for experiments with dataset D1



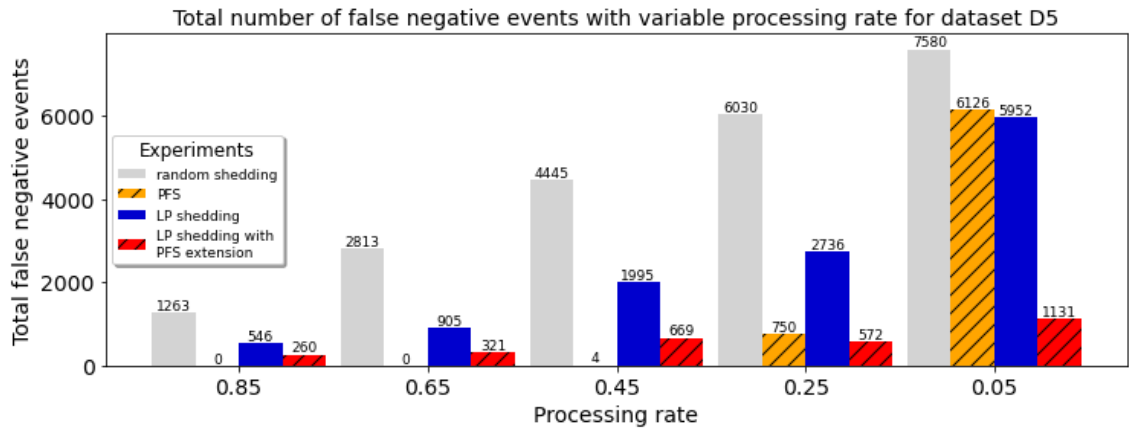
**Figure A.2:** Total number of false negatives for experiments with dataset D2



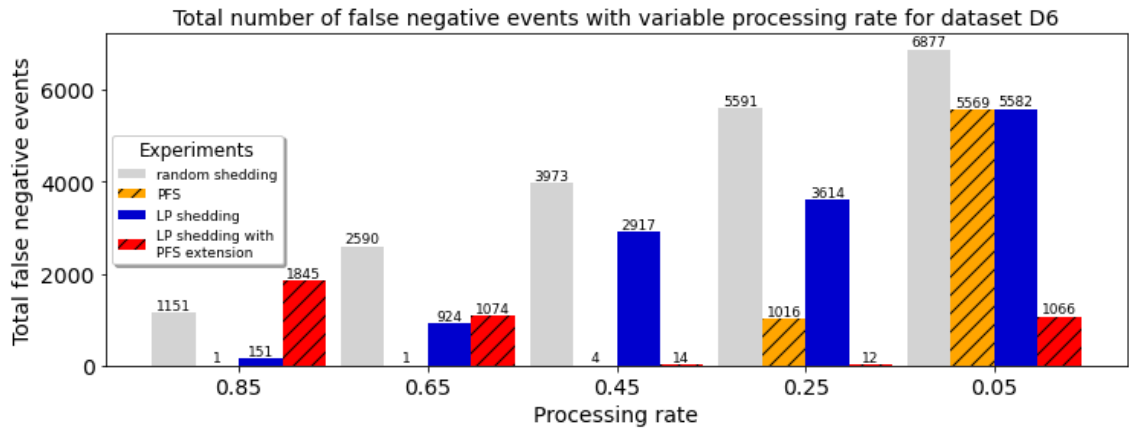
**Figure A.3:** Total number of false negatives for experiments with dataset D3



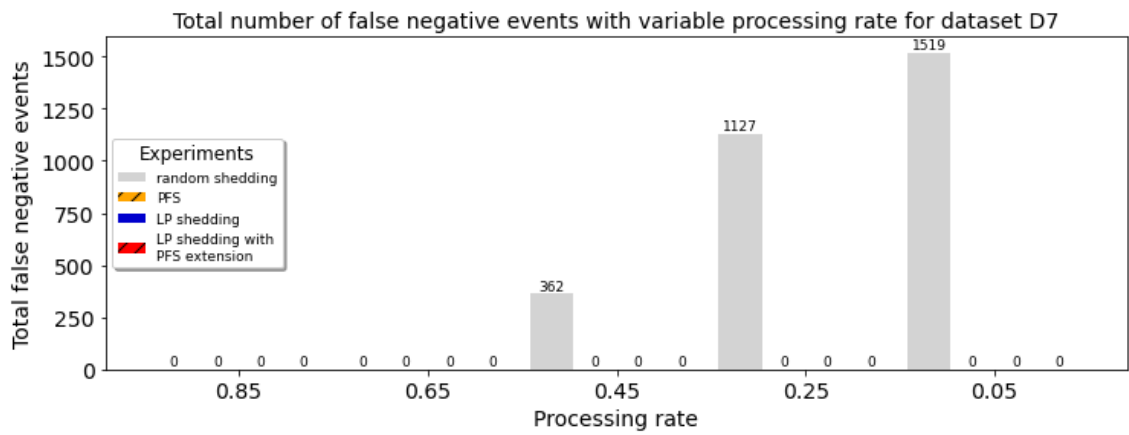
**Figure A.4:** Total number of false negatives for experiments with dataset D4



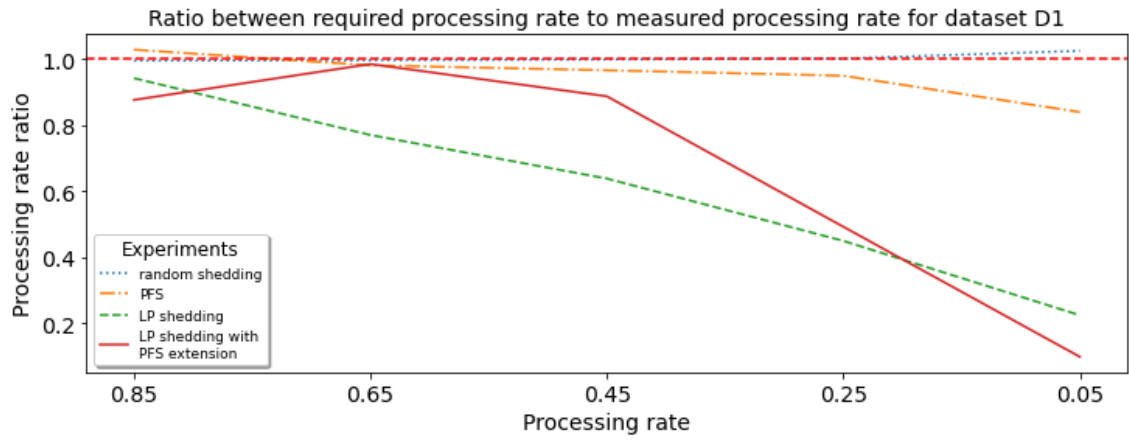
**Figure A.5:** Total number of false negatives for experiments with dataset D5



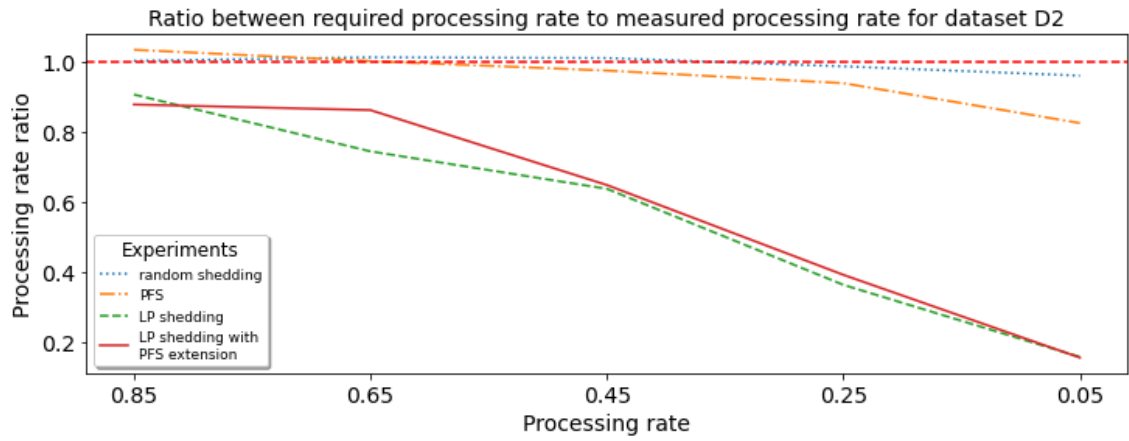
**Figure A.6:** Total number of false negatives for experiments with dataset D6



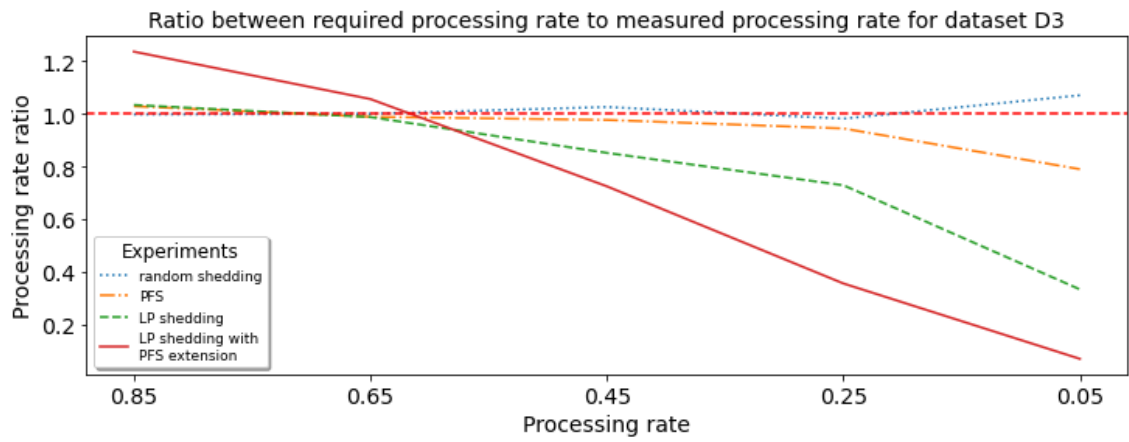
**Figure A.7:** Total number of false negatives for experiments with dataset D7



**Figure A.8:** Ratios of required to measured processing rates for experiments with dataset D1

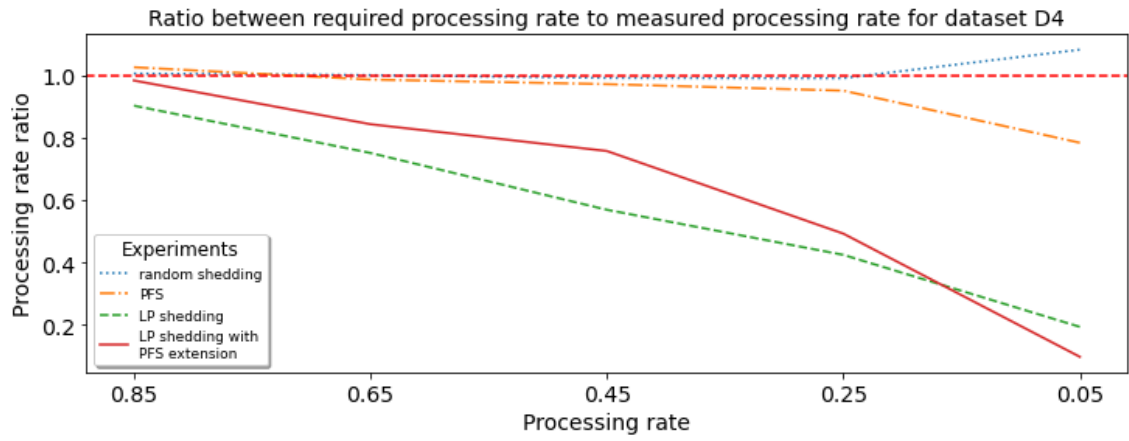


**Figure A.9:** Ratios of required to measured processing rates for experiments with dataset D2

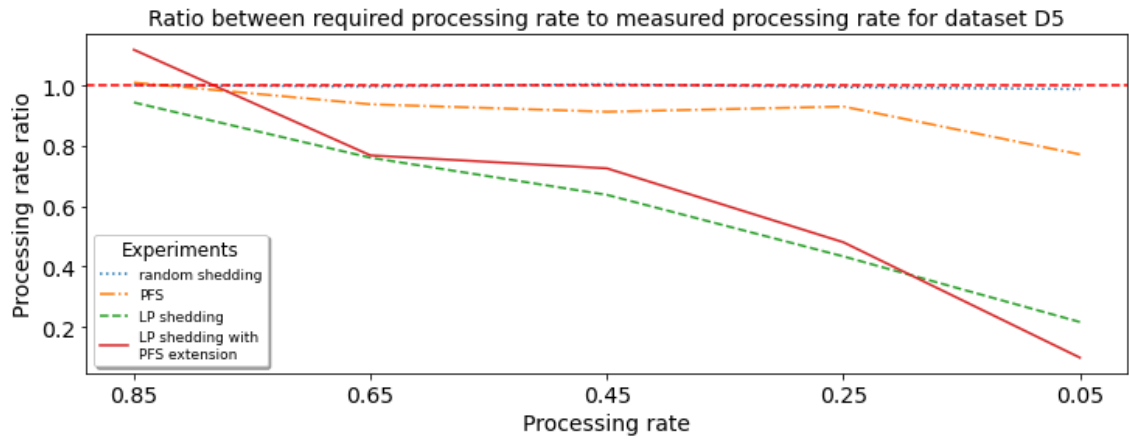


**Figure A.10:** Ratios of required to measured processing rates for experiments with dataset D3

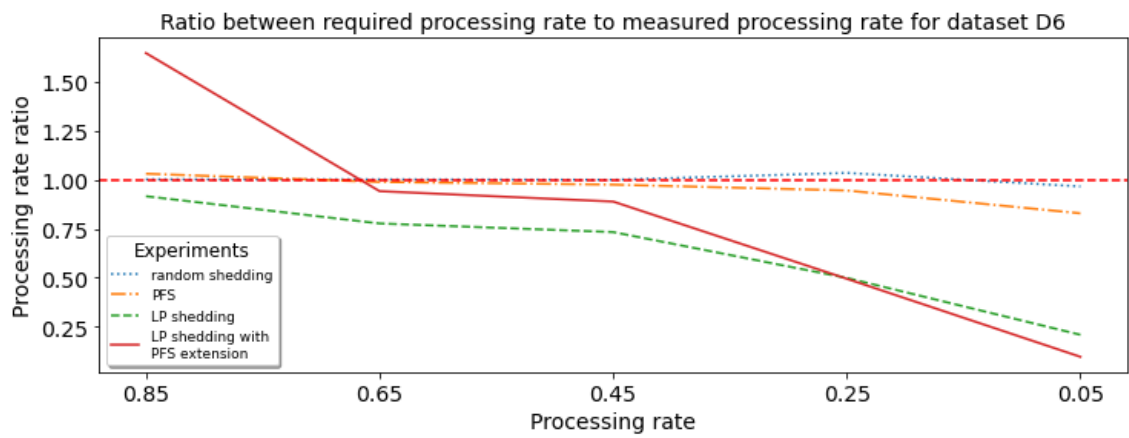




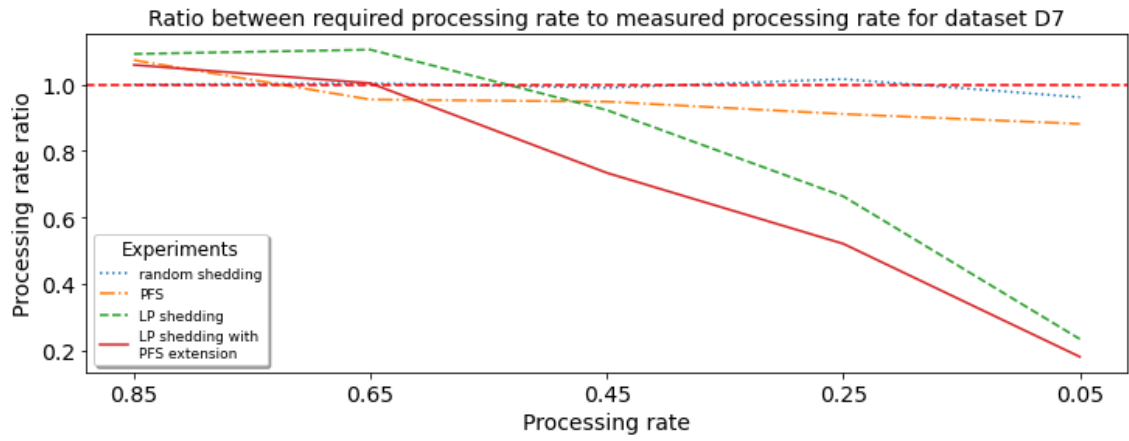
**Figure A.11:** Ratios of required to measured processing rates for experiments with dataset D4



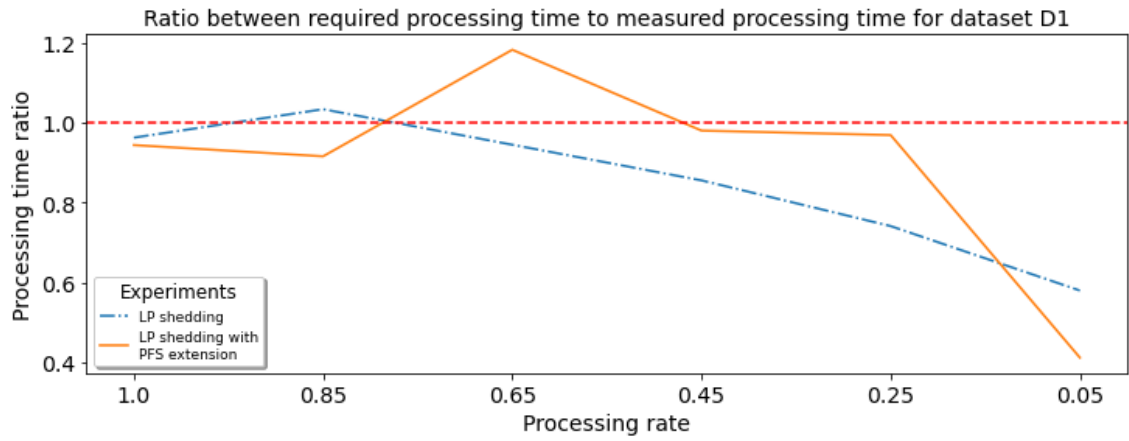
**Figure A.12:** Ratios of required to measured processing rates for experiments with dataset D5



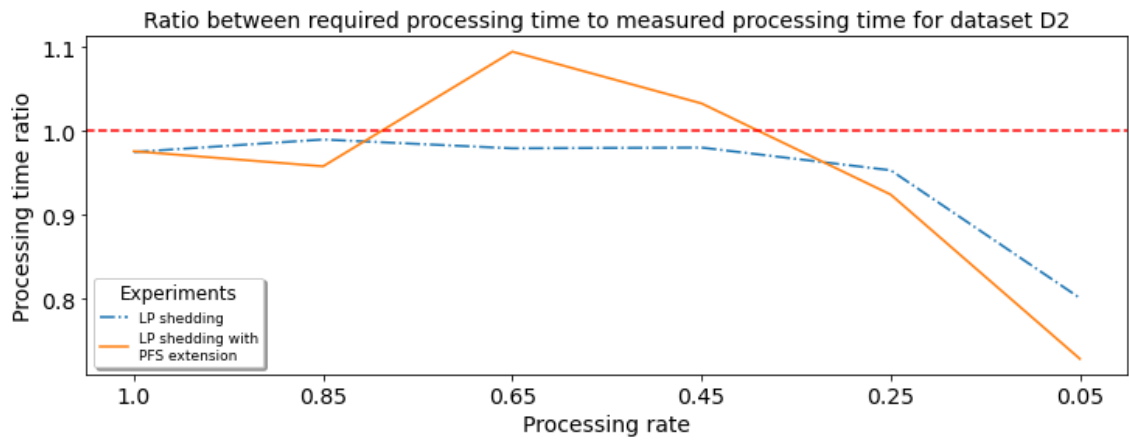
**Figure A.13:** Ratios of required to measured processing rates for experiments with dataset D6



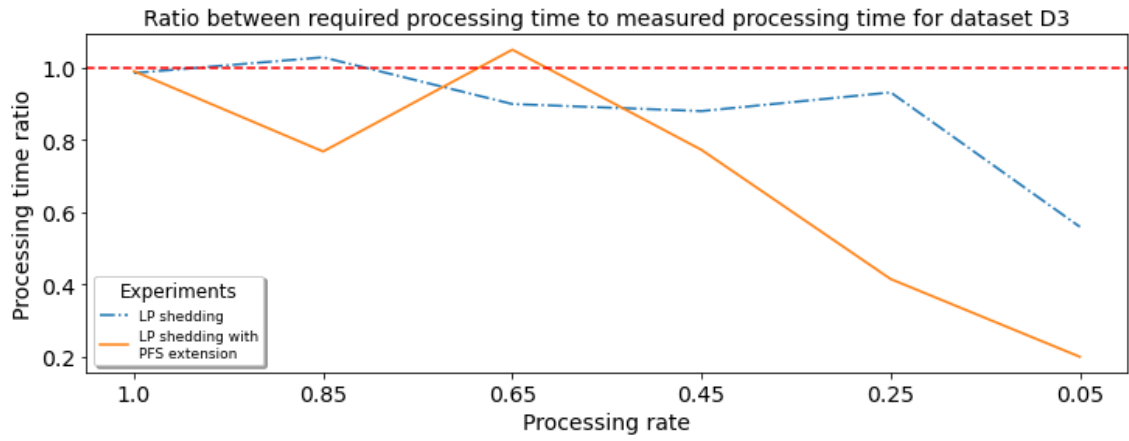
**Figure A.14:** Ratios of required to measured processing rates for experiments with dataset D7



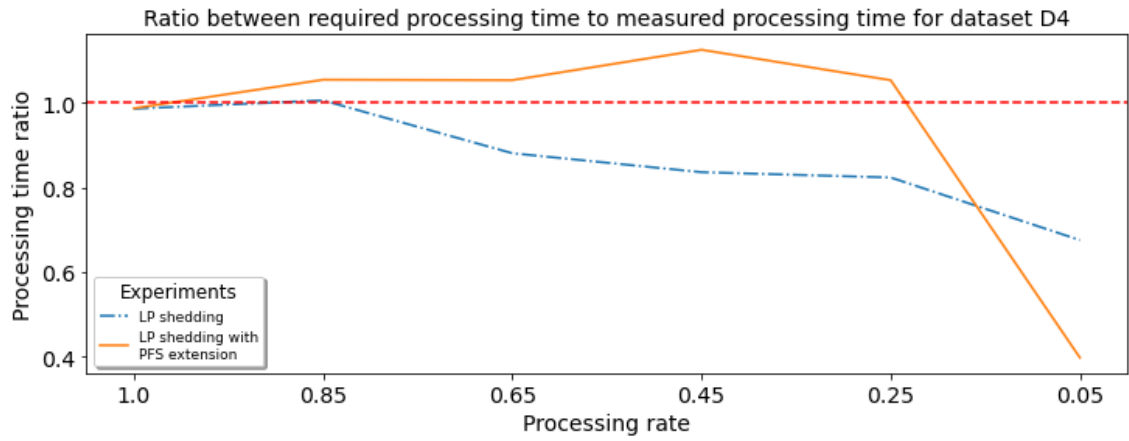
**Figure A.15:** Ratios of required to measured processing times for experiments with dataset D1



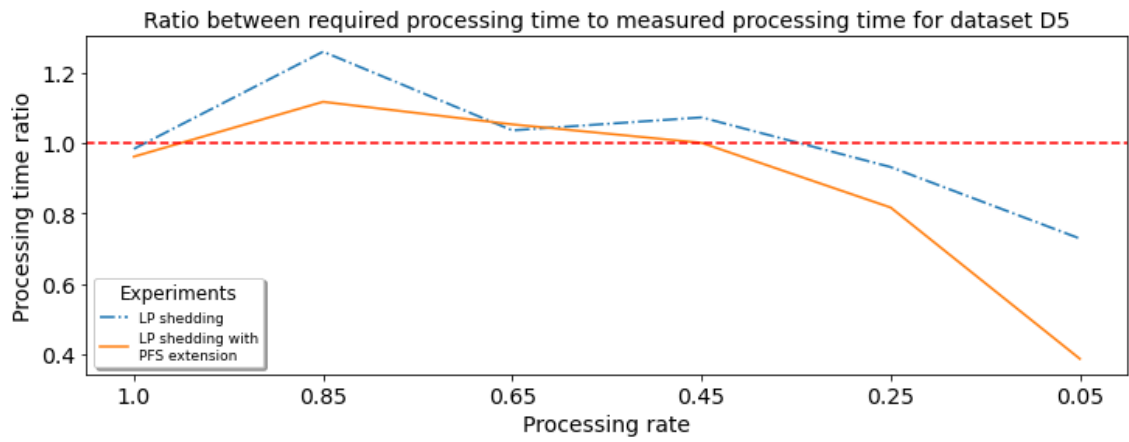
**Figure A.16:** Ratios of required to measured processing times for experiments with dataset D2



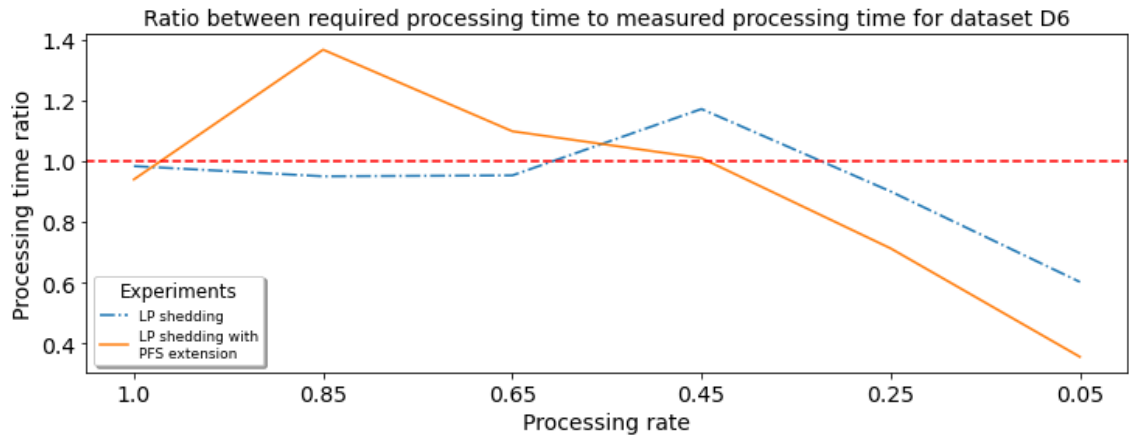
**Figure A.17:** Ratios of required to measured processing times for experiments with dataset D3



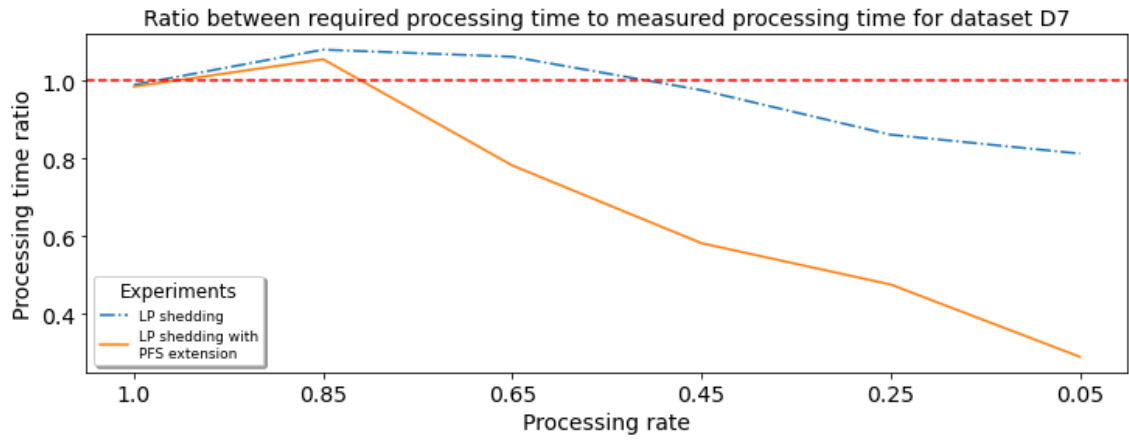
**Figure A.18:** Ratios of required to measured processing times for experiments with dataset D4



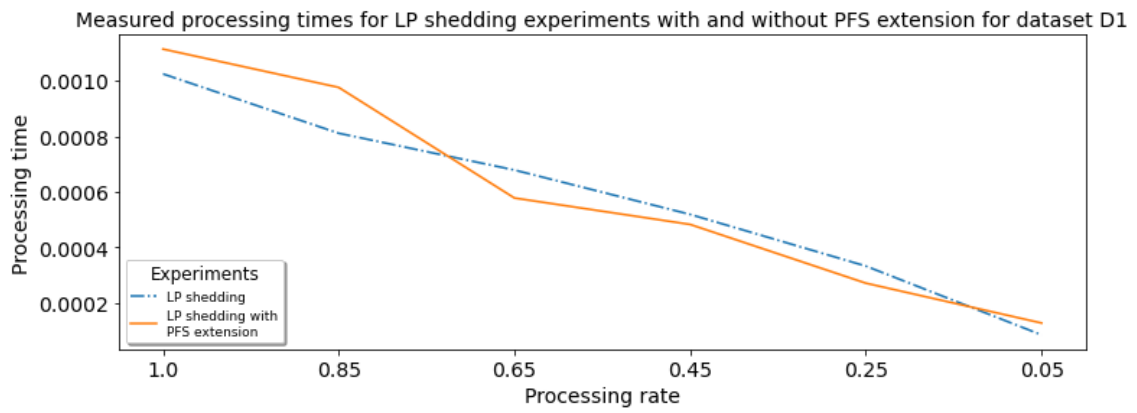
**Figure A.19:** Ratios of required to measured processing times for experiments with dataset D5



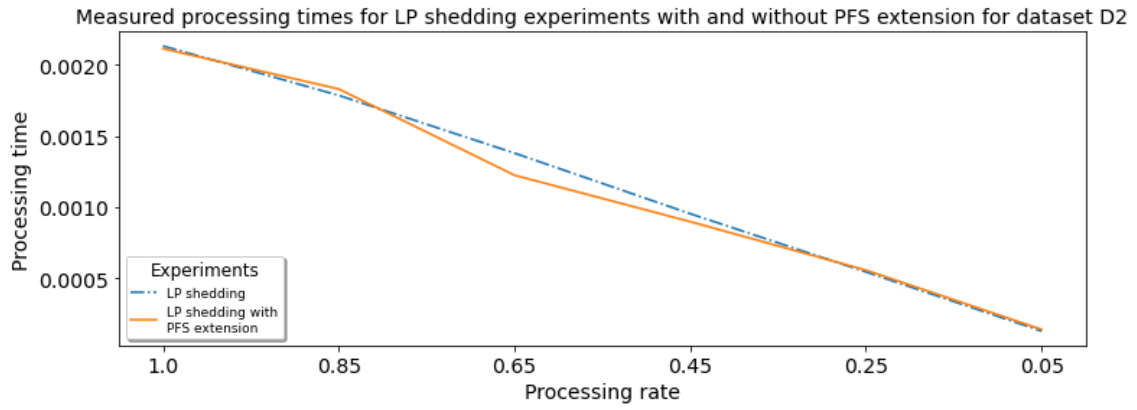
**Figure A.20:** Ratios of required to measured processing times for experiments with dataset D6



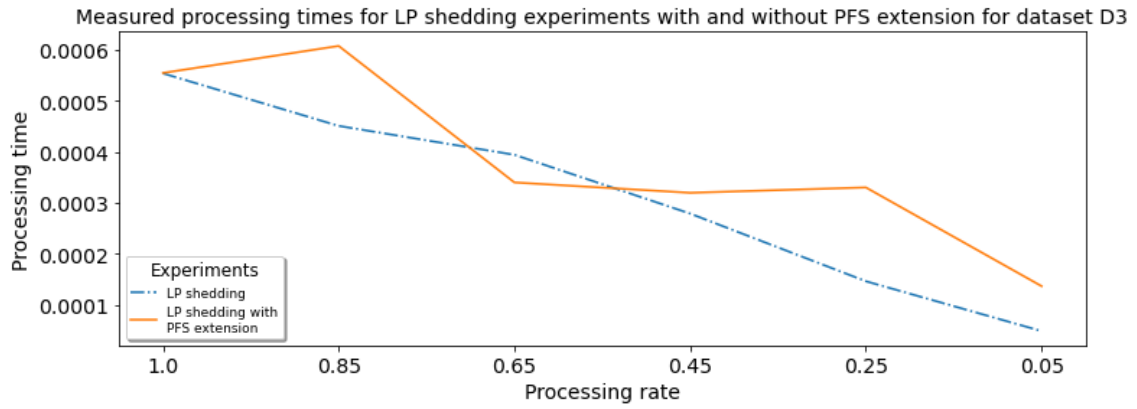
**Figure A.21:** Ratios of required to measured processing times for experiments with dataset D7



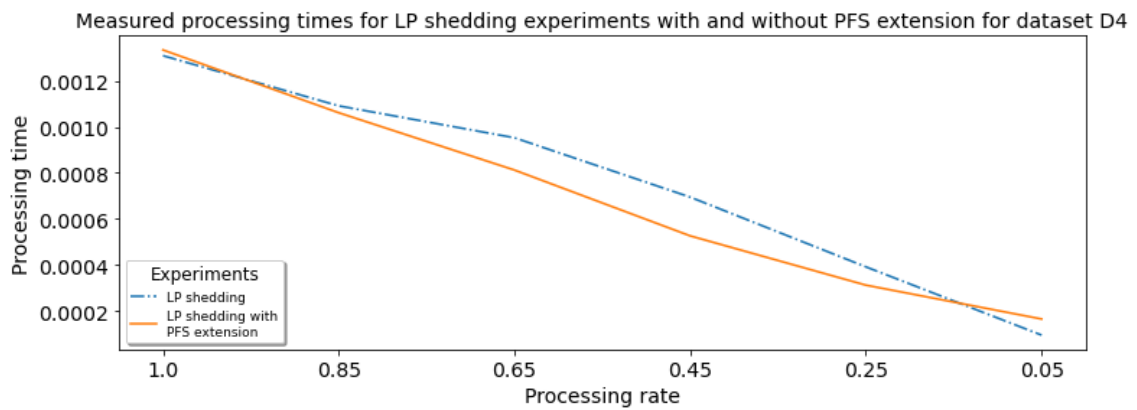
**Figure A.22:** Measured processing times for experiments with dataset D1



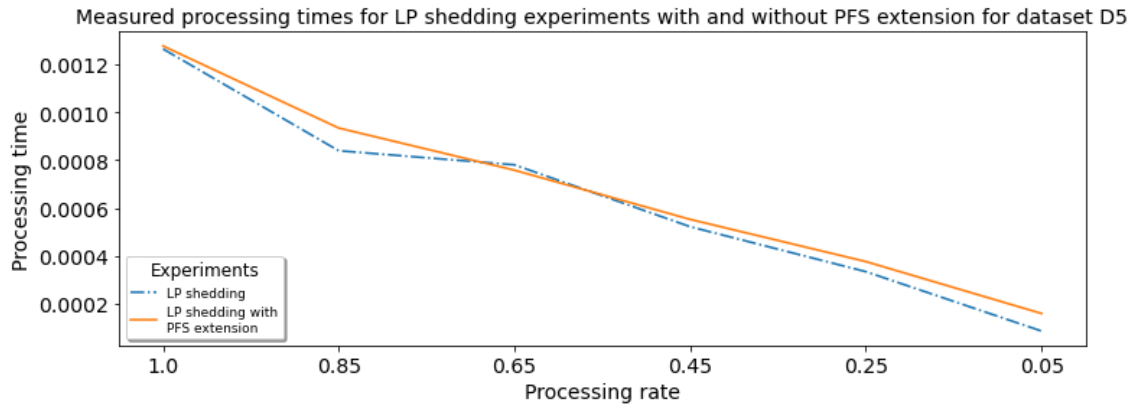
**Figure A.23:** Measured processing times for experiments with dataset D2



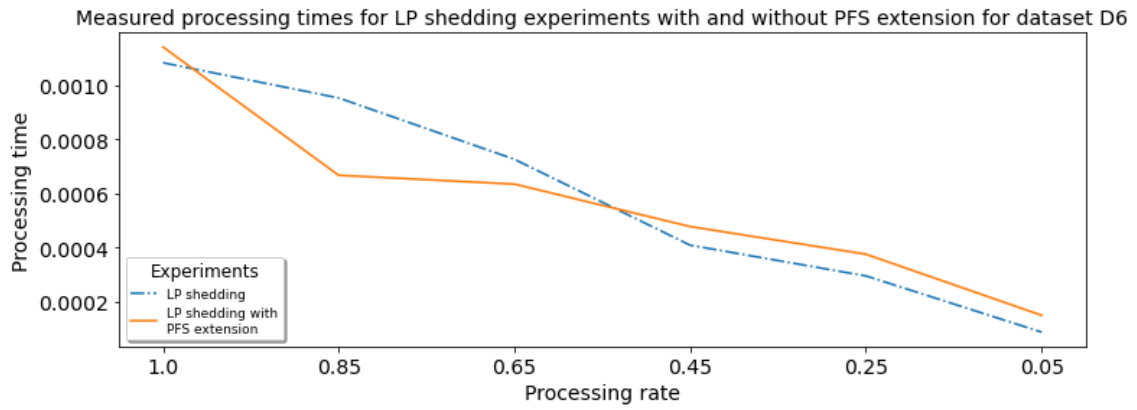
**Figure A.24:** Measured processing times for experiments with dataset D3



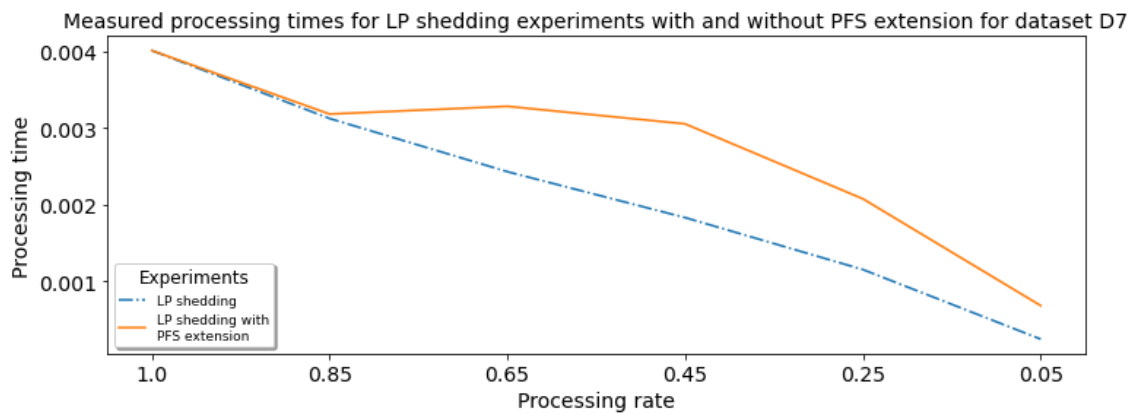
**Figure A.25:** Measured processing times for experiments with dataset D4



**Figure A.26:** Measured processing times for experiments with dataset D5



**Figure A.27:** Measured processing times for experiments with dataset D6



**Figure A.28:** Measured processing times for experiments with dataset D7

### **Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature