Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Modeling Interferences of CEP Operators on Limited Resources

Simon Glaub, B.Sc.

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Henriette Röger, M.Sc. , Dr. Sukanya Bhowmik
Commenced:	November 17, 2020
Completed:	Mai 17, 2021

Abstract

Complex Event Processing (CEP) systems are used to combine low-level data from an input stream into high-level information. To account for workload peaks load shedding can be used to drop events. To determine when to drop events, the delay of the CEP system needs to be predicted by its workload. But if multiple operators of the CEP system share a resource, the workload of one operator does also influence the performance of the other operators. In this thesis, we examine the interference effect between multiple operators by building a prediction model. To solve this task we consider it a regression problem, where we use the arrival rate of an operator to predict the processing time of another operator on the same node. To also take into account the difference between the arrival rates of different event types, we introduce the balance score as the second input variable. Next, we design an experiment to generate diverse data. The data generated this way is then used to build prediction models by using two different methods: regression analysis and a neural network. After finding the best prediction model for each method, we compare the performance of these models. Here we show that which model is better mostly depends on the specific use case of the CEP system.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface. 46
CEP	Complex Event Processing. 3
LAN	Local Area Network. 35
MSE	Mean Squared Error. 24
RELU	Rectified Linear Unit. 23
SGD	Stochastic Gradient Descent. 24

1 Introduction

1.1 Motivation

Nowadays in all areas of life huge amounts of data are generated. This leads to a high demand to analyze these information streams. In many of these areas, it is important that these data streams get evaluated fast to make it possible to react fast to the observed events.

Usually, CEP applications have a high arrival rate of input data, which can also fluctuate. These fluctuations are caused trough traffic peaks. For example, an infrastructure monitoring application usually gets more data during the daytime, because then the interaction is higher compared to during the nighttime. But even with these fluctuating arrival rates, the CEP application needs to produce high-level information reliable and quick.

One possibility to deal with workload peaks is to use load-shedding techniques. For load-shedding techniques, every event gets a utility value assigned, which shows how important this event is for the output quality. Then if the workload passes a certain threshold the events with low utility values are dropped. This ensures that the resources can process the more important events.

To make load-shedding techniques work reliable it is important to define the amount of data that needs to be dropped to ensure a certain latency bound. This is especially challenging if multiple processes of the CEP application share a resource. In this case, the workload of one process does not only influence its own performance, but also the performance of the other processes. So, to determine how many events need to be dropped, it is necessary to model the influence the workload of a process has on the latency of the resource.

1.2 Problem Statement and Goals

We assume that a CEP application consists of one or multiple operators. These operators are located on nodes, which represent the resources that are used. Every operator receives events as input, which either are provided by an external source or from another operator. The operators then process these events and combine them to new events, which are emitted. The arrival rate of events at an operator has an impact on the processing time of this operator. If multiple operators are located on one node, there might be interference effects. Since a resource only has limited processing power, it is unavoidable, that one or multiple operators under heavy load are degrading each other's performance. To take such interference effects into account, a model needs to be developed that can predict performance changes of all operators on a node. To achieve this the model should use the change of the arrival rates of single operators as input. It is expected that the interference effects have a complexity that is too high to solely rely on analytical models. Therefore machine learning techniques should also be used to train the model. The model developed shall then be implemented on top of the existing CEP framework *Precept II*. Also, the performance and correctness of the implemented model shall be evaluated. So, in the end, it shall be possible to provide a vector of arrival rates to the implemented model and receive the predicted processing times for the operators.

1.3 Structure

In this section, we explain how this master thesis is structured. Chapter 1 shows the motivation for this thesis and then defines the problem statement and the goals. In Chapter 2 we convey the background knowledge which is necessary to understand the following chapters of this master thesis. Here we first explain CEP and then introduce the Precept II framework. We also describe two methods that can be used to solve regression problems: regression analysis and neural networks. In Chapter 3 we explain the approach with which we want to achieve the goal of this master thesis and which steps are necessary for it. Chapter 4 describes how we generate the data which is necessary to build our prediction model. Here we first show which metrics we use as variables for our prediction model. Then we describe the design of the experiment we use to generate and gather the necessary data. Last we explain how we adjust the Precept II framework so that it meets our advanced requirements. In Chapter 5 we show how we use the gathered data to build prediction models. Here we first explain how we need to preprocess the data. Then we show how we can build prediction models by using regression analysis and neural networks. In Chapter 6 we discuss our results. Here we compare the prediction models we built and describe the limitations of our results. Finally, we summarize the results of this master thesis in Chapter 7 and provide an outlook on how the topics dealt with in this thesis can be pursued further.

2 Background

In this chapter, we explain the background, which is necessary to understand this thesis. The first two sections explain the concept of CEP and describe the CEP framework that is used in this thesis. The other two sections of this chapter introduce two concepts that can be used to solve a regression problem. The first of these sections explains the different types of regression analysis. The second of these sections explains how to build a neural network and how it works.

2.1 Complex Event Processing (CEP)

Complex Event Processing (CEP) denotes concepts and methods to process arriving events from event streams and extract information from them. These events often happen in the external world and are observed for example by sensors [CM12]. CEP is usually used in real-time situations. Therefore its goal is to identify meaningful events as quickly as possible. To achieve this CEP aggregates, filters, and matches low-level events to combine them to new higher-level events [RC10]. These events are then consumed by sinks, which either trigger actions or forward the new high-level event. Also in CEP providers and receivers of information are decoupled. That means that the providers do not need to know anything about the potential receivers and the receivers do not need to know anything about the possible sources or the events they might receive [BK09].

2.1.1 Areas of Application

CEP can be used in a variety of different fields. In the following, we shortly describe some of the most common areas of application as described in [EB09]. One area of application to use CEP is in the field of business activity monitoring. Here the CEP application tries to identify problems and opportunities of a business domain at an early stage. For that, the application monitors business processes and company-critical resources. Another possibility is to use CEP in combination with sensor networks. Here sensors capture events from the outside world and send them to a CEP application. Then CEP can be used to combine the data from multiple sensors to minimize measurement errors. CEP can also be used to combine events from different sensor types to detect a more complex situation. So for example the data from a temperature sensor and a smoke sensor could be combined to detect a fire. Another field in which CEP can be useful is to analyze market data. Data like stock or commodity prices can also be viewed as events. To identify trends at an early stage this data must be analyzed promptly and continuously. If necessary the application can then react automatically. This is also called algorithmic trading.

2.1.2 Events

A CEP application consumes events from an event stream to process them further. According to Luckham [Luc02] an event records an activity in a system in the form of an object. An event contains data regarding the recorded activity and meta-data like a sequence number. Events can be related to each other through time, causality, and aggregation. Time is saved as a property of an event in the form of a timestamp. Causality between two events exists if one event causes the other event. If event A causes event B this is denoted as A -> B. If event A caused event B, this means that A had to happen before B. This also means that the timestamp of A needs to be earlier than the timestamp of B. Some events can also be aggregated, which is one of the main tasks of a CEP application. Formally this means that a set of events B_i can be combined to a higher-level event A. This event A consists of the activities recorded by the aggregated events. Such higher-level events may also be aggregated further into even more high-level events. The more high-level an event gets, the further it usually separates itself from the hardware level and the closer it gets to the business domain.

In [HBN13] it is described that an event stream is usually modeled as a sequence of events. In such a sequence $S = (e_1, e_2, ..., e_n)$ each event e_i has a timestamp t_i and an event type q_i. The sequence is ordered according to the timestamps, so that $t_i < t_j, i < j$

2.1.3 Operator Graph and Patterns

A CEP application contains one or multiple operators which together form a directed acyclic graph. The edges between the operators transmit events between the operators. At its boundaries, the operator graph receives events from the input stream and delivers output events. An operator graph can be executed on one machine or distributed in a network consisting of multiple machines.

An operator receives events from the event stream or from another operator and tries to aggregate them to new higher-level events according to predefined patterns. These higher-level events are also called complex events [SBR20]. Two of the most simple pattern types are sequence-patterns and and-patterns. A sequence-pattern is denoted by a sequence $P_{seq} = (q_1, q_2, ..., q_n)$ of event types q_i. As soon as the specified sequence of events P_{seq} is captured by the operator a new complex event is created. An and-pattern is denoted by a set $P_{and} = \{q_1, q_2, ..., q_n\}$ of event types q_i. As soon as all events from the specified set P_{and} are captured by the operator a new complex event is created. Each pattern P is only evaluated over a certain time frame [HBN13]. If no complex event is created within this time frame, the corresponding pattern instance is dropped. This is necessary because of usually CEP applications process time-sensitive data. So after a certain amount of time the events received become useless. Therefore a complex event must be created before this point in time.

2.1.4 Load Shedding

The arrival rate of event streams is often fluctuating and is especially high during peak times [HBN13]. This can quickly become a performance bottleneck for CEP applications. The reason for this is that the processing of events is stateful [ZVW20]. Each operator stores a set of partial matches for the pattern it is looking for. This state can grow exponentially if the arrival rate greatly surpasses the processing rate of events. This leads to slower processing of the arriving events. This can be a serious problem, because in most CEP applications complex events need to be created within a certain latency bound, otherwise they become useless [SBR19].

To avoid this problem load shedding can be used. Load shedding means that a part of the events arriving at an operator is dropped. This reduces the load on this operator and lead to lower latency at detecting complex events [SBR19]. Simple load shedding strategies just drop events randomly. But if events are dropped that are very important, this may lead to a decreasing quality of detecting complex events. Therefore it can be beneficial to estimate how great the impact of an event on the quality of results is. This value is usually called the utility of an event. It can be best determined by a user or administrator, which has a good knowledge of the application [HBN13]. Instead of dropping random events, this can be used to drop events with a lower utility resulting in a better quality of results. The method explained so far is called input-based load shedding. Another method that can be used is state-based load shedding. Here, instead of dropping arriving events, partial matches of patterns is dropped. By this, the size of the state of an operator is reduced, which also leads to a decreased processing time.

A load shedding system usually has three main tasks [HBN13]. First, it needs to decide when it needs to conduct load shedding. Therefore it must be able to detect when a load peak happens and if this peak is so big that the latency bound of the operator is violated. Second, the load shedding system needs to decide which events should be dropped. For that it can for example use the utility value described above. And third, the load shedding system needs to decide how many events should be dropped. So it must be able to calculate how much a dropped event improves the processing time and when the latency bound of the operator is not violated anymore. Additionally, the load shedding system should use an efficient method to drop events, so that the overhead of the load shedding does not become too big [SBR19].

2.2 Precept II Framework

The Precept II Framework is the CEP framework that is used in this thesis. It was build as part of the PRECEPT II project, which is carried out by the Institute for Parallel and

Distributed Systems of the University of Stuttgart. The goal of this project is to ensure a given latency bound of a CEP application. Therefore load shedding techniques are developed, which maximize the perceived quality of result. The framework currently is still under development. In the following we describe the components of the Precept II Framework and how they work together.

2.2.1 Components

The core components of the Precept II Framework are the operators. An operator receives events and tries to combine them into complex events. These complex events are created according to predefined patterns. A pattern is represented by a statemachine, which is handed to an operator when the operator is created. Two types of statemachines are already defined: the sequence-statemachine and the and-statemachine. The and-statemachine looks for a specified set of events. As soon as all specified events arrived at an operator with this type of statemachine, a complex event is created. The order in which these events arrive does not matter. The sequence-statemachine looks for a specified sequence of events. In contrast to the and-statemachine, here it matters in which order the events arrive. As soon as all specified events arrived in the specified order at an operator with this type of statemachine, a complex event is created. Operators can open multiple statemachines to store partial matches. So, if an event arrives, that can not be used in any of the currently opened statemachines and represents the start of the pattern, a new statemachine is opened. A statemachine is dropped after a certain amount of time if it has not created a complex event till then. It is possible to define this time. An operator is also the place where load-shedding can happen. Currently, only random load-shedding is implemented. Here it is possible to define the percentage of events that should be kept. Every time an event would be processed a random number between 0 and 1 is generated. If the percentage of events, that should be kept, is larger or equal than the value of [1 - this random number], then the event is processed. Otherwise, the event is dropped.

Two more components that are used in the Precept II Framework are producers and sinks. A producer creates events and sends them to an operator. Every event contains an event type, a payload, a message-id, and a timestamp. The event type, the message-id, and the timestamp are metadata, whereas the payload contains the actual data of the event. The event type usually is a number that identifies the type of the event, the message-id is used to clearly distinguish the events and the timestamp contains the time of sending. The output rate of a producer can be adjusted by increasing the delay between the emitted events. If we view a producer as the starting point of a CEP process, a sink is the end of the process. If an operator creates a complex event it can either forward it to another operator or it can send it to a sink. In the current implementation, a sink does not do anything when receiving a complex event, apart from collecting some metadata like the arrival time. But in the future, this would be the location, where actions can be triggered, depending on the received complex events.

The last component that is a part of the Precept II Framework, is the metrics consumer. Operators, producers, and sinks also send metadata that can be used for logging. The metrics consumer is the component that collects all this metadata and saves it to a file or a database. The metrics consumer also further process some data. For example, it calculates the processing time of an operator, based on the start time and end time of the processing. The data collected by the metrics consumer can for example be used to examine the performance of the CEP system.

2.2.2 Topology

The Precept II Framework uses so-called scenarios to define the topology of a CEP system. A scenario needs to define which operators exist in the CEP system and how they are connected. Every operator needs an identifier, which usually is an integer number greater than zero. This identifier is later used to refer to this operator when forming a topology. An operator consists of one or multiple sub-operators. By that is it possible for one operator to have multiple different processing steps. If only one processing step is needed, only one sub-operator needs to be defined. Every sub-operator needs to have the following properties: identifier, name, and a list of output consumers. The identifier usually is an integer number greater than zero. The name property defines which operator class is used by the sub-operator. An operator class defines how events are processed and is explained in more detail in Section 2.2.1. The list of output consumers defines where the generated complex events is sent to. This can be other operators or sinks. Additionally, a sub-operator can have the properties sleep_time_mean and sleep_time_sdv. These properties can be used to generate random delays according to a gauss distribution. These delays can be used to slow down the processing of operators.

A scenario also needs to define which nodes exist in the CEP system and how the operators are distributed among them. Every node needs to have the following properties: an identifier and a list of included operators. The identifier usually is an integer number greater than zero. The list of included operators consists of the identifiers of all operators that run on that node. Additionally, a node can also have the properties sleep_time_mean and sleep_time_sdv. These properties can be used to generate random delays according to a gauss distribution. These delays are then used for all operators on that node. Nodes can be started separately and can therefore run on different physical machines. This makes it possible to distribute the CEP system among a network.

2.3 Regression Analysis

Regression analysis is a statistical method to investigate the relationship between variables. It is one of the most used methods to analyze data that consists of multiple factors. Regression analysis is usually used to determine if an independent variable has an effect on a dependent variable and how strong this effect is. To avoid confusion with the concept of statistical independence in regression analysis the independent variable is usually called the predictor or regressor variable and the dependent variable is called the response variable [MPV12]. Regression analysis uses an equation to represent the relationship between a set of regressor variables and the response variable. The data used in a regression analysis can originate from an observational study, or from existing historical records. Another possibility is to specifically design an experiment to gather the needed data.

2.3.1 Linear Regression Model

Regression analysis can be used to build a linear regression model. First, we look at the cases where only one regressor variable is used. Such a regression analysis is called simple linear regression [Syk93]. The goal of simple linear regression is to build a linear regression model which is represented by Equation (2.1).

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.1}$$

Here y denotes the response variable and x denotes the regressor variable. The parameter β_0 denotes the value of y when x = 0 and is called the intercept. The parameter β_1 denotes how fast the value of y changes when the value of x changes and is called the slope. If we view the linear regression model as a graph, it takes the form of a straight line. So in this context, β_0 denotes the point where the line intersects with the y-axis and β_1 denotes how steep the slope of the line is. The main goal of a regression analysis is to determine the values of β_0 and β_1 so that Equation (2.1) is as correct as possible with regard to the examined data. Therefore these parameters are usually also called the regression coefficients. Usually, it is not possible that the linear regression model fits perfectly to the data only on the basis of the regression coefficients. Therefore Equation (2.1) contains the parameter ϵ which can be viewed as a statistical error. The value of ϵ is a random variable that shows the difference between the model and the actual data. So ϵ should ensure that the model fits the examined data exactly [MPV12]. In reality, the regression model usually is still only an approximation of the actual relationship between the regressor variable and the response variable. It is also important to keep in mind that the regression model generally only can be applied to the range of regressor variables that are contained in the examined data.

Apart from simple linear regression, it is also possible to perform a multiple linear regression. The main difference to simple linear regression is that multiple linear regression uses multiple different regressor variables. This makes it possible to determine the impact multiple factors have on a single response variable [Syk93]. For multiple linear regression, the linear regression model needs to become more generalized and is therefore now represented by Equation (2.2).

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + \epsilon \tag{2.2}$$

Compared to Equation (2.1) this new equation nearly stays the same. The only difference is, that now the equation takes multiple regressor variables into account, which are denoted by $x_1, x_2, ..., x_k$. Also, the equation can no longer be drawn as a simple straight line. If we only have two regressor variables we can still construct a graph by using three dimensions. In this case, the graph takes the form of a plane. If we have more than two regressor variables it is no longer possible to construct a graph.

Like mentioned earlier the goal of regression analysis is to determine the unknown regression coefficients. Therefore usually the method of least squares is used. First, the estimated error is defined as the difference between the actual value of the response variable and the value the response variable has according to the regression model. The regression analysis then calculates the sum of the squares of the estimated errors for all data points. By choosing the regression equation with the minimum value of this sum, the model fits the data as closely as possible. This technique can be used regardless of whether a simple or multiple regression is performed.

2.3.2 Polynomial Regression Model

If we can not find a linear regression model that fits our data well, it is possible to build a polynomial regression model. A linear regression model tries to capture the relationship between a regressor variable and a response variable by an equation that only uses the first degree of the regressor variable. A polynomial regression model on the other hand also uses higher degrees of the regressor variable in its regression equation. Therefore this type of model can also capture a curvilinear relationship between the regressor variable and the response variable [Ost12]. The simplest example for a polynomial regression model is the second-order polynomial model in one variable. This model is also called a quadratic model and is represented by Equation (2.3) [MPV12].

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon \tag{2.3}$$

As in a linear regression model y denotes the response variable, x denotes the regressor variable, β denotes the regression coefficients and ϵ denotes the error. Here β_1 is also called the linear effect parameter and β_2 the quadratic effect parameter. If we create a graph from this equation, it takes the form of a quadratic function. A polynomial regression model can also use higher degrees for x. The generalized k^{th}-order polynomial model in one variable is therefore represented by Equation (2.4) [MPV12].

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_k x^k + \epsilon \tag{2.4}$$

Using higher degrees for x makes it possible to capture even more complex relationships between the regressor variable and the response variables.

It is also possible to build a polynomial regression model with more than one regressor variable. The simplest example for this would be the second-order polynomial model in two variables, which is represented by Equation (2.5).

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \epsilon \tag{2.5}$$

This model is usually called a response surface [MPV12]. A polynomial regression model can theoretically use any number of regressor variables combined with any degree for x. The values of these factors depend on the concrete model that should be built.

To successfully build a polynomial regression model, the optimal values for the regression coefficients β need to be determined. For that a polynomial regression model usually is viewed as a multiple linear regression model [Ost12]. This means that the x variables with higher degrees are handled as new regressor variables. Therefore the examined data needs to be preprocessed. For example, for a second-order polynomial model in one variable, we need to square the regressor variable and label it as a new regressor variable. This makes it possible to just use the method of least squares like for linear regression models.

2.3.3 Coefficient of Determination

After a regression model has been built, it is important to check how well it fits the examined data. One of the most commonly used metrics is the coefficient of determination R^2. The coefficient of determination shows to which extend the regression model can explain the variation of the response variable [Syk93]. This metric can be calculated by using Equation (2.6) [MPV12].

$$R^2 = 1 - \frac{SS_{Res}}{SS_T} \tag{2.6}$$

$$SS_{Res} = \sum_i (y_i - \bar{y})^2 \tag{2.7}$$

$$SS_T = \sum_i (y_i - f_i)^2 \tag{2.8}$$

SS_{Res} is calculated by using Equation (2.7) and SS_T is calculated by using Equation (2.8). Here y_i denotes the actual measured response variables, $\bar{y}$ denotes the mean value of the measured response variables and f_i denotes the response variable determined by the regression model. So R^2 calculates how much better the regression model can determine the response variable compared to just taking the mean value of the response variable. The coefficient of determination usually takes a value between 0 and 1. A value of 0 means that the response variable is determined by the regression model not any better than just taking the mean value of the response variable. A value of 1 means that the regression model fits the examined data perfectly. In this case, the regression model determines exactly those values for the response variables that have also been measured. The coefficient of determination theoretically can also take a negative value. This means that the regression model is worse at determining the response variable than taking the mean value of the response variable. This usually indicates that the approach to how the model was built is entirely wrong or that there are faults in the measured data. It should also be noted that a bad value of R^2 does not necessarily mean that the regression model is bad. A bad value of R^2 can for example also result from a huge amount of noise in the data [Syk93].

2.4 Neural Networks

This section mostly uses information from [Nie15] to describe neural networks. A neural network can use training data to learn something from it. Neural networks are used a lot for classification tasks, for example in the field of image processing. But neural networks can also be used to solve regression problems.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Figure 2.1: Output of a perceptron with the usage of a threshold [Nie15]

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Figure 2.2: Output of a perceptron with the usage of a bias [Nie15]

2.4.1 Neurons

A neural network consists of multiple neurons, which form a graph. Originally neural networks used perceptrons as neurons. A perceptron gets multiple binary values $x_1, x_2, ..., x_n$ as input and produces a single binary value as output. Also a weight $w_1, w_2, ..., w_n$ is assigned to every input. The output of the perceptron is determined by comparing the weighted sum $\sum_{i=0}^{n} w_i x_i$ to a certain threshold. Figure 2.1 shows that the output is 1 if the weighted sum is greater than the threshold and 0 otherwise. Usually, the threshold is moved to the left of these equations, where it is called bias. These new equations are shown in Figure 2.2.

Nowadays instead of perceptrons neural networks usually use sigmoid neurons. A sigmoid neuron basically works like a perceptron with the difference that its output is processed further. For that it uses an activation function σ, which takes the weighted sum $x = \sum_{i=0}^{n} w_i x_i + b$ as input and produces a new output value. The most commonly used activation functions are:

sigmoid: $\sigma = \frac{1}{1+e^{-x}}$

tanh: $\sigma = \frac{2}{1+e^{-2x}} - 1$

linear: $\sigma = a * x$

Rectified Linear Unit (RELU) $\sigma = max(0, x)$

For the sigmoid and the tanh activation function this output is between 0 and 1. The important difference between perceptrons and sigmoid neurons is, that the output of sigmoid neurons is changing less for changing input than it does for perceptrons. And this is exactly the property that is needed for a neural network, to better learn from the provided data and adapt to them.

2.4.2 Stochastic Gradient Descent

The main goal of a neural network is to minimize a cost function. This cost function shows how much the output values predicted by the neural network differ from the actual output values. This difference is called loss. One of the most commonly used cost functions is the Mean Squared Error (MSE) denoted by Equation (2.9).

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a||^2 \tag{2.9}$$

Here, w denotes all weights, b denotes all biases, $y(x)$ denotes the expected output value for the input value x, and a denotes the output value predicted by the neural network. The value of a is calculated using w, b and x. The task of the neural network is to find a combination of weights and biases, so that $C(w,b) \approx 0$. This can be achieved through gradient descent. For that, we first need to calculate the gradient vector ∇C for the cost function C. The gradient vector consists of the partial derivatives for all weights in w and all biases in b. If ∇C is calculated we know the direction in which the cost function rises. So, to minimize the cost function we just need to move in the opposite direction. By establishing a learning rate η, it is possible to control the speed of the movement in this direction. The complete formula to calculate the next values for w and b is denoted by Equation (2.10).

$$(w', b') = (w, b) - \eta\nabla C \tag{2.10}$$

By repeating this procedure multiple times the cost function can be minimalized. It is important to choose a good value for η. Because when η is too high, the changes in the weights and biases is too high to precisely reach the minimum. And if η is too low, it takes a long time to get some meaningful changes.

Usually, large amounts of data are being used to train a neural network. Since it would be quite time-consuming to calculate the gradients for all this data, usually Stochastic Gradient Descent (SGD) is used. SGD works like gradient descent with the difference, that average gradients are calculated for small subsets of the training data. These subsets are called mini-batches and are randomly chosen. When all training data has been used in mini-batches to train the neural network, an epoch of training is finished. Then a new epoch is started, where the training data is assigned to new mini-batches. The more epochs of training a neural network performs, the more accurate its predictions are.

2.4.3 Architecture

A neural network is separated into multiple layers: the input layer, hidden layers, and the output layer. This architecture is shown in Figure 2.3. The neurons located in the input layer transfer the training data to the neurons in the first hidden layer. They are not real neurons per definition but are normally just modeled as such. The amount of input neurons is equal to the number of input variables. The hidden layers use an activation function on the weighted input data and forward the result. The more hidden layers a neural network has, the more complex problems it can solve. Usually, one hidden layer is

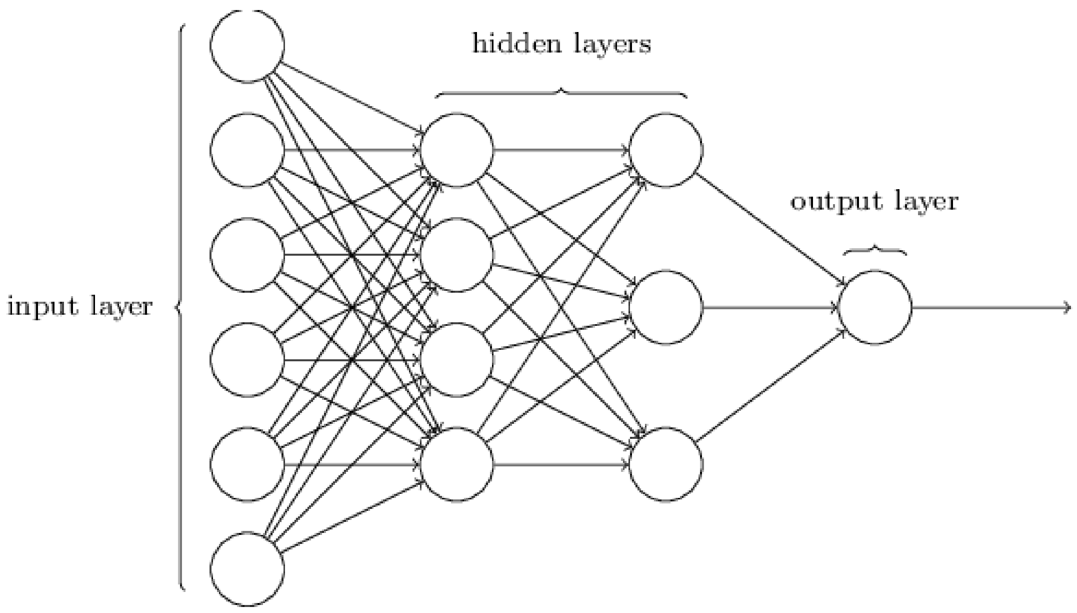

Figure 2.3: The layers of a neural network [Nie15]

enough to solve the most common problems. How many neurons a hidden layer needs to contain, is dependent on the concrete problem. Heaton [Hea08] describes the following rules of thumb that are often used to get a starting point:

- Choose a number of hidden neurons between the size of the input layer and the size of the output layer.
- Choose a number of hidden neurons equal to $\frac{2}{3}$ the size of the input layer plus the size of the output layer.
- Choose a number of hidden neurons that is less than twice the size of the input layer.

The output layer puts out the final result of the neural network. The amount of output neurons is equal to the number of output variables to be predicted.

3 Approach

The core task of this master thesis is to build a model that can predict the processing times for the operators in a CEP system. This prediction model gets a vector as input, which contains the arrival rates for the events arriving at the operators of the CEP system. This input vector should then be transformed into an output vector containing the processing times for these operators.

This task can be understood as a regression problem. At a regression problem, you want to understand the relationship between a response variable and one or multiple regressor variables. Therefore a model is built which describes this relationship. This process is called a regression analysis. Such a model can also be used to predict the response variable based on the regressor variables. This functionality is exactly what we want to achieve. So if we view the goal of this master thesis as a regression problem, the arrival rates are the regressor variables and the processing time of an operator is the response variable.

To be able to perform a regression analysis it is necessary to first collect data points. Each data point is a pair consisting of a set of regressor variables and the corresponding response variable. Often this data already exists or can be extracted from a running application. But in the context of this master thesis, we have no already existing CEP system, which could be used for that purpose. So the first step is to build a CEP system that can be used to generate the necessary data. Here the focus should be specifically set on the interference effect between operators on the same node because the understanding of these effects is also a goal of this thesis. How this CEP system is built and how the data is generated and captured is described in more detail in Chapter 4.

In this thesis, we use two different methods to solve the regression problem. The first method is to perform a regression analysis, which is described in Section 2.3. The second method is to build a neural network, which is described in Section 2.4. We implement prototypes for both methods and then evaluate their performance. Therefore we check how accurate the predictions of the models are, how fast the prediction is, and how long it takes to build the model. Then we compare the two prototypes and discuss how good they serve their purpose and which one is better.

4 Data Generation

To generate meaningful prediction models we need data regarding the relationship between the regressor variables and the response variable. Because we do not yet have any data that we can use, we need to generate that data. In this chapter, we first define which metrics we use as regressor variables and response variable. Then we define the experiment setup to gather the needed data. Here we also show the CEP system that is used for this. Last we show how the existing CEP framework is adjusted to make it possible to perform the experiment and gather all needed data.

4.1 Selection of Variables

The goal of the master thesis is to predict the performance of a CEP system. We decided that the best way to do this is to measure the processing time of each operator. Therefore we set the processing time of an operator to be the response variable.

The problem statement defines that the arrival rates of a CEP system should be used as input to the prediction model. Therefore we define that one regressor variable is the arrival rate of an operator. An operator usually receives events of different types. A predefined pattern of these event types then triggers the creation of a new complex event. As explained in Section 2.2.1 an operator stores partial matches for a pattern in open statemachines. We suspect that the number of open statemachines influences the processing time of an operator. If the arrival rates for the different event types do not fit the pattern of the operator good enough, this leads to more open state machines. Therefore we originally planned to add the individual arrival rates for different event types as regressor variables. But this would mean that a high number of different event types would greatly increase the number of configurations that we need to generate data for. So instead we determine the proportion between these arrival rates and the pattern of the operator and combine them into a single value. We call this value the balance score and formally define it in Section 4.1.2. So the two regressor variables we use are the arrival rate and the balance score of an operator. The two following sections describe the execution of experiments to test if there is a correlation between each regressor variable and the response variable. This way we can verify that it makes sense to use the arrival rate and balance score of an operator as regressor variables.

4.1.1 Arrival Rate

One regressor variable we want to use is the arrival rate of an operator. The arrival rate of an operator is defined as the number of events that arrive at an operator per second. Therefore we do not make a difference between the different event types and instead count all the events that arrive at the operator. In Section 4.3.2 we explain in more detail how this arrival rate is calculated.

In the problem statement, the arrival rate is suggested as input for the prediction model. Intuitive this makes sense because if an operator receives more events he might not be able to process them fast enough. But to use the arrival rate as a regressor variable we wanted to make sure that there is at least some form of correlation between it and the response variable. Therefore we conducted an experiment to show that such a correlation exists. In this experiment, we use a small CEP system consisting of two operators that are on the same node. Both operators implement an AND state machine that searches for the pattern `[0,1,2]`. Since the prediction models should focus on the interference effect between operators, the goal of the experiment is to show that the arrival rate of operator 1 influences the processing time of operator 2. The experiment consists of twelve different configurations which all run separately. Each configuration defines different values for the arrival rate and balance score of the two operators. Operator 2 has constant values for all configurations so that it does not have any influence on its own processing time. The arrival rate of operator 2 is set to 300 events per second and the balance score is set to 1. Operator 1 has a different arrival rate for each configuration, but a constant balance score of 500. This way we can make sure that changes of the processing time only are caused by changes of the arrival rate. Each configuration runs for five minutes.

Table 4.1: Measured values for the correlation between the arrival rate of operator 1 and the processing time of operator 2

Id	Balance Score	Arrival Rate	Arrival Rate 0	Arrival Rate 1	Arrival Rate 2	Processing Time
1	519.625	49.1656	46.9067	1.37938	0.879527	0.0032202
2	517.702	98.8586	93.9938	2.93238	1.93245	0.0070381
3	526.293	199.373	189.824	5.87182	3.67731	0.00381435
4	537.76	300.794	287.135	8.32893	5.33001	0.0060878
5	543.536	404.868	387.077	10.8967	6.89418	0.00463933
6	529.491	496.462	472.072	14.68	9.70998	0.0084955
7	515.415	590.963	564.664	16.141	10.1573	0.00485561
8	526.093	693.669	660.153	20.2359	13.2801	0.00735489
9	532.332	795.69	755.898	23.8406	15.9516	0.00765038
10	547.75	995.761	959.899	22.0391	13.8229	0.0039904
11	544.454	1047.07	1001.59	27.608	17.8681	0.00591003
12	551.247	2784.57	2662.98	73.4541	48.1357	0.00578154

While performing the experiment we measured the arrival rate and balance score of operator 1 and the processing time of operator 2 for each arriving event. Then we took the mean of these values for every configuration. Table 4.1 shows the results of the experiment. Because it is very difficult to generate precise values for the balance score, the measured value of it only is in the close range of the planned value of 500. But the variation of the balance score should still be low enough so that we mainly capture the influence of the arrival rate. The measured arrival rates are in a range between ca. 50 and 2780 events per second. When we performed this experiment, 2780 events per second was the maximum we could reach and 1050 was the second-highest. In Section 4.3.3 we explain the reason for this in more detail. Table 4.1 also shows the arrival rates for the different event types, which determine the balance score.

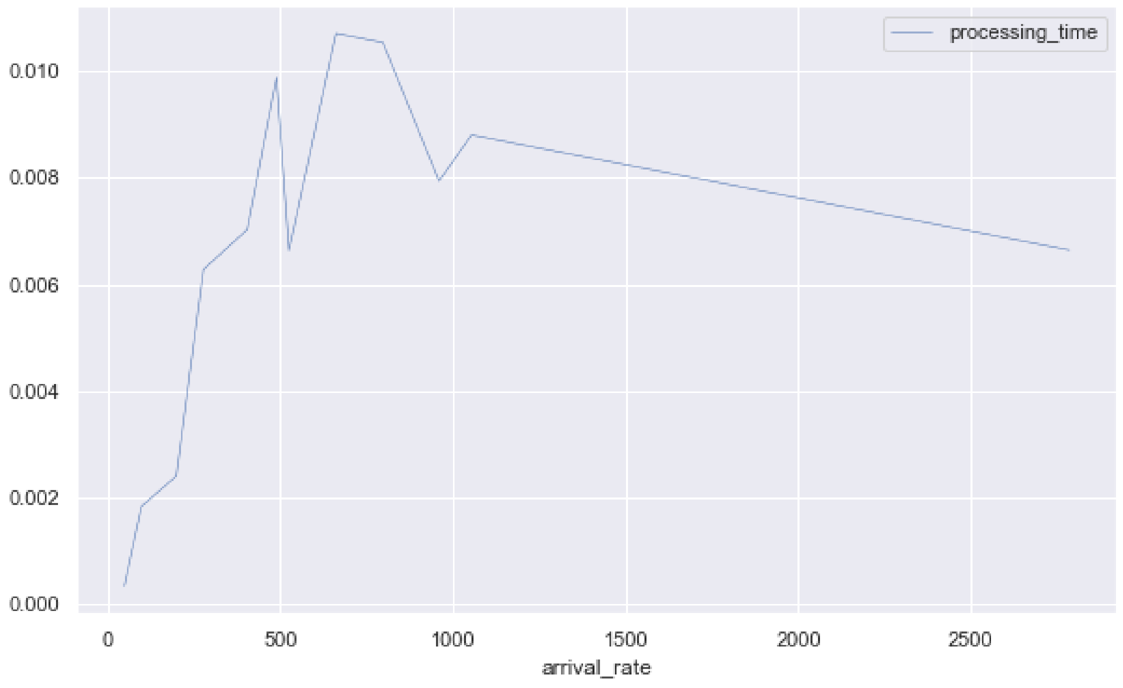

Figure 4.1: Correlation between arrival rate of operator 1 and processing time of operator 2

Next, we use these measured values to create a graph that shows the correlation between the arrival rate of operator 1 and the processing time of operator 2. In Figure 4.1 it can be seen that the processing time of operator 1 shows a clear upward trend till an arrival rate of around 660 with only one collapse. Afterward, the processing time is slowly decreasing. But because we could not take any measurements between the arrival rates of 1050 and 2780 events per second, it can not be said for sure how the processing time in this area actually evolves. But at least till a certain boundary, the arrival rate of operator 1 seems to influence the processing time of operator 2. So, the experiment showed that there seems to be at least some form of correlation between the arrival rate and the processing time.

4.1.2 Balance Score

The second regressor variable we want to use is the balance score of an operator. The balance score was created by us to determine the imbalance between arriving event types and the pattern of an operator. To calculate the balance score, we first need to get the proportion of an event type regarding the pattern of the operator. To calculate this proportion $proportion\_pattern_m$ we use Equation (4.1).

$$proportion\_pattern_m = \frac{n_m}{n} \tag{4.1}$$

Here n_m denotes the number of times an event of the type m is used in the state machine and n the total amount of events the pattern consists of. Next, for every event type, we need to get its proportion regarding the arrival rate for the whole operator. To calculate this proportion $proportion\_rate_m$ we use Equation (4.2).

$$proportion\_rate_m = \frac{rate_m}{rate} \tag{4.2}$$

Here $rate_m$ denotes the arrival rate of an event of the type m and $rate$ the arrival rate of the whole operator. Now we calculate the ratio $ratio_m$ between $proportion\_pattern_m$ and $proportion\_rate_m$ by using Equation (4.3).

$$ratio_m = \frac{proportion\_rate_m}{proportion\_pattern_m} \tag{4.3}$$

The closer the value of $ratio_m$ is to 1, the more balanced the arrival rate of an event type m is with regard to the pattern of the operator. Finally, we calculate the balance score by multiplying the values of $ratio_m$ for all event types as shown in Equation (4.4).

$$balance\_score = \prod_m ratio_m \tag{4.4}$$

There might be cases where some values of $ratio_m$ are greater than 1 and some are smaller than 1. In such a case the ratios balance out each other by multiplying, which leads to a wrong balance score. Therefore if the value of $ratio_m$ is smaller than 1, we transform it by using Equation (4.5).

$$ratio_m = \frac{1}{ratio_m} \tag{4.5}$$

This ensures that the value of $ratio_m$ is always greater than 1 while remaining the ratio factor that corresponds to the original value.

In the following, we show the calculation of the balance score on an example. The operator used in the example has the following pattern: `[1,1,2,3]`. This leads to the following values:

$proportion\_pattern_1 = \frac{2}{4} = 0.5$

$proportion\_pattern_2 = \frac{1}{4} = 0.25$

$proportion\_pattern_3 = \frac{1}{4} = 0.25$

The arrival rate for the whole operator is four events per second: $rate = 4$. The arrival rates for the single event types are $rate_1 = 1$, $rate_2 = 2$ and $rate_3 = 1$. This leads to the following values:

$proportion\_rate_1 = \frac{1}{4} = 0.25$

$proportion\_rate_2 = \frac{2}{4} = 0.5$

$proportion\_rate_3 = \frac{1}{4} = 0.25$

Now we caclulate the ratio between these values:

$ratio_1 = \frac{0.25}{0.5} = 0.5$

$ratio_2 = \frac{0.5}{0.25} = 2$

$ratio_3 = \frac{0.25}{0.25} = 1$

Because $ratio_1$ is smaller than 1 we need to transform it first: $ratio_1 = \frac{1}{0.5} = 2$. Finally we can now calculate the balance score: $balance\_score = 2 * 2 * 1 = 4$.

We defined the balance score to account for the different arrival rates of the event types. A higher balance score should lead to a greater amount of open state machines and therefore to a higher processing time. But to use the balance score as a regressor variable we wanted to make sure that there is actually some form of correlation between it and the response variable. Therefore we conducted an experiment to show that such a correlation exists. In this experiment, we use a small CEP system consisting of two operators that are on the same node. Both operators implement an AND state machine that searches for the pattern `[0,1,2]`. Since the prediction models should focus on the interference effect between operators, the goal of the experiment is to show that the balance score of operator 1 influences the processing time of operator 2. The experiment consists of 15 different configurations which all run separately. Each configuration defines different values for the arrival rate and balance score of the two operators. Operator 2 has constant values for all configurations so that it does not have any influence on its own processing time. The arrival rate of operator 2 is set to 300 events per second and the balance score is set to 1. Operator 1 has a different balance score for each configuration, but a constant arrival rate of 1000 events per second. This way we can make sure that changes of the processing time only are caused by changes of the balance score. Each configuration runs for five minutes.

While performing the experiment we measured the arrival rate and balance score of operator 1 and the processing time of operator 2 for each event. Then we took the mean of these values for every configuration. Table 4.2 shows the results of the experiment. The measured arrival rate is around 700 events per second as opposed to the planned 1000 events per second. But the measured arrival rate stays pretty constant across the configurations. This constancy is the most important point about the arrival rate if we want to make sure to only capture the influence of the balance score. Therefore the difference between the planned and measured values should not be a problem. The measured balance

Table 4.2: Measured values for the correlation between the balance score of operator 1 and the processing time of operator 2

Id	Balance Score	Arrival Rate	Arrival Rate 0	Arrival Rate 1	Arrival Rate 2	Processing Time
1	1.12723	719.689	239.039	238.825	239.444	0.000692008
2	1.95526	722.393	278.341	278.63	164.764	0.00489911
3	3.24937	726.564	386.968	169.981	169.998	0.0050639
4	6.11735	723.644	455.406	133.924	134.014	0.00602502
5	25.6039	711.125	556.353	71.0824	71.0531	0.0075465
6	61.5593	701.315	594.087	57.3274	38.6764	0.00830747
7	151.313	690.363	622.23	48.114	19.6725	0.00823603
8	249.258	688.157	619.541	57.2794	9.90859	0.00827435
9	530.521	697.727	658.197	9.91207	29.2938	0.0090639
10	1293.06	699.359	670.104	24.5081	4.97537	0.0094126
11	1556.77	684.334	664.141	9.91654	9.91824	0.00927385
12	2884.58	695.866	680.769	7.9433	6.95427	0.00951577
13	3639.2	691.34	675.994	10.9017	3.98386	0.00946009
14	4938.93	686.019	673.399	7.94341	3.98382	0.00884541
15	6314.39	685.066	674.904	4.97595	4.97592	0.00944013

scores are in a range between ca. 1 and 6300. Table 4.2 also shows the arrival rates for the different event types, which determine the balance score.

Next, we use these measured values to create a graph that shows the correlation between the balance score of operator 1 and the processing time of operator 2. We used a logarithmic scale for the balance score because this way it is easier to see the trend of the graph. In Figure 4.2 it can be seen that at the start the processing time rises pretty fast. Then the rise of the processing time slows down till it finally stays at a rather constant value for the rest of the graph. This means the influence of the balance score on the processing time gets weaker the higher the balance score becomes. So, the experiment showed that there is a correlation between the balance score and the processing time.

4.2 Experiment Design

In the last sections, we chose the overall arrival rate and the balance score of an operator to be the regressor variables and the processing time of an operator to be the response variable. In this section, we design an experiment that is used to gather the data necessary to build the prediction models. The experiment uses a CEP system, which consists of two operators that run on the same node. Both operators implement an AND state machine with the pattern `[0, 1, 2]`. We intentionally choose a rather simple CEP system, to make sure there are no other factors that possibly would distort the results. By this, we can ensure that the results focus on the interference effect between operators and how well this effect can be predicted by the prediction models. Also, this is basically the same setup

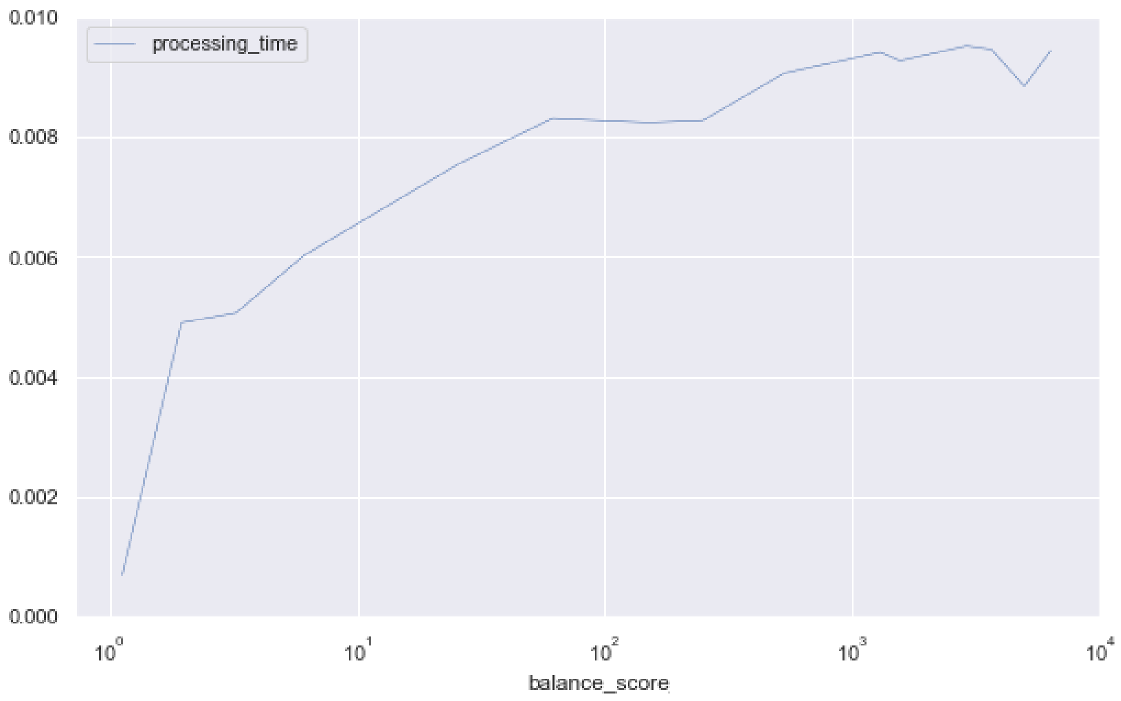

Figure 4.2: Correlation between balance score of operator 1 and processing time of operator 2

we already used for the experiments to show the correlation between the regressor and response variables. Since this worked fine, we can expect this CEP system also works fine in this case and we do not have to deal with unexpected problems.

To make sure the producers do not influence the processing time of the operators, we run these components on separate machines. The producers run on a laptop with a 1.70 GHz processor with 4 cores and 4 GB RAM. The operators run on a desktop computer with a 3.00 GHz processor with 4 cores and 16 GB RAM. The producers on the laptop generate events and send them to the operators on the desktop computer by using Local Area Network (LAN). This way the producers and operators use completely separated resources and therefore do not influence themselves.

To build meaningful prediction models it is important to have diverse data. This data should capture as many values and combinations of the regressor variables as possible. To reach that goal we first define a minimal and a maximal value for the arrival rate and the balance score. For the arrival rate, the range is between 50 and 1000 events per second. We choose 50 events per second as the minimal arrival rate because the experiments in Section 4.1.1 showed that at this arrival rate the influence on the processing time is negligible (see Figure 4.1). This ensures that there are data points, where only the balance score influences the processing time. Originally we wanted to choose 10.000 events per second as the maximal arrival rate. But when we performed some test runs with this arrival rate, we discovered that the operator takes much longer to process all events than the producer sends events. The cause of this is, that the mean processing time of the operator is at 0.0017 seconds. That means the operator processes around 590 events per second. It is obvious that at an arrival rate of 10.000 events per second this processing speed leads to a fast-growing queue of events. This means that the operator already is overloaded since it can not keep up with the incoming events. So we decided to use 1000 events per second as the maximal arrival rate. This arrival rate already pushes the operator to its limit and further increasing it should not increase the influence on the processing time. For the balance score, the range is between 1 and 1000. We choose 1 as the minimum balance score because at this score the arrival rates of the event types are perfectly balanced. So at this score, only the arrival rate influences the processing time. We choose 1000 as maximal balance score, because the experiments in Section 4.1.2 showed that at this value the processing time begins to stagnate (see Figure 4.2). So it can be assumed the effect of the balance scores on the processing time does not grow any further past this point.

Table 4.3: Configurations for operator 1

Ids	Arrival Rate	Balance Scores
1-10	50	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
11-20	156	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
21-30	261	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
31-40	367	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
41-50	472	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
51-60	578	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
61-70	683	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
71-80	789	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
81-90	894	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]
91 - 100	1000	[1, 112, 223, 334, 445 ,556 ,667 ,778 ,889 ,1000]

Now we choose ten values for the arrival rate and the balance score, which are evenly distributed in the ranges defined by these minimal and maximal values. For every combination of these values, we have a configuration that is executed individually. This results in a total of 100 different configurations, which can be seen in Table 4.3. The arrival rate in the middle column is combined with all balance scores in the right column. The goal of this experiment is to generate data that can be used to examine the influence operator 1 has on operator 2. Therefore the configurations shown in Table 4.3 are only used with operator 1. The arrival rate and balance score of operator 2 is constant across all configurations. We choose the minimal values from the ranges defined before. So operator 2 has an arrival rate of 50 events per second and a balance score of 1. By using these values the arrival rate and balance score of operator 2 do not have a noticeable influence on its own processing time. Therefore we can ensure, that only the interference effect operator 1 has on operator 2 is captured.

Next, we decided how long each configuration should run. To make sure that the generated data is equally distributed we need to make sure that each configuration produces the same amount of data points. Because we want to measure the processing time of operator 2, it is therefore important that operator 2 receives the same amount of events in every configuration. Since operator 2 has the same arrival rate for every configuration, every configuration needs to run the same amount of time to reach that goal. We decided that each configuration should produce 10.000 data points, which leads to a total of 1.000.000 data points. Because the arrival rate of operator 2 is 50 events per second, each configuration needs to run for 200 seconds. It may appear that producing so many data points from a configuration does not increase the diversity of the data points because they always use the same arrival rate and balance score. But actually, the variety of these data points is bigger, because the arrival rate is never exactly the same as the planned values. In reality, the individual arrival rates can fluctuate and are only somewhere around the planned values. This also leads to fluctuating balance scores, since they are directly dependent on the arrival rates. This fluctuation has the positive effect that the real variety of the data points is much bigger than the planned variety. Therefore the generated data should be diverse enough to build meaningful prediction models.

4.3 Implementation

In this section, we show how we adjusted the existing CEP framework so that we can run the experiments described in the previous sections. Therefore we first explain how the producers of the CEP framework are adjusted, so each event type can have an individual output rate. Next, we show how we adjusted the metric consumer to calculate and measure the variables that are needed to build the prediction models. In the last section, we explain how and why a new networking library was integrated.

4.3.1 Producers

In the existing CEP framework it is possible to create events and send them to an operator through a producer. It is possible to adjust the output rate of a producer by inserting a delay between the sending of two events. Usually, one producer sends all different event types in random or predefined patterns to the operator. But to perform the experiments described in the last sections we need the possibility to define a different output rate for every event type on an operator. To achieve this we assign one producer to each event type on an operator. Now we can set a different delay for each producer and therefore have different output rates. Every producer is started in a different thread. This way all producers can run in parallel. To make it easier to create a CEP system it is possible to enter which operators use which event types and assign the desired output rates directly to these pairs. These output rates are then converted to the equivalent delay by using Equation (4.6). Then for each specified event type on an operator a producer, which uses the previously calculated delay, is created and started on a new thread.

$$delay = \frac{1}{output\_rate} \tag{4.6}$$

When we tested the new implementation, we noticed that in many cases the actual output rate is much slower than the specified output rate. When we examined this issue more closely, it became apparent that with larger output rates the difference between the actual and specified output rate also becomes larger. The cause of this problem is that the producer also takes a small time to be executed and to send the events. When the specified output rates are low, this leads to a high delay between emitted events. In this case, the execution time is too small in relation to the delay to have a big impact. But when the output rate becomes higher, this results in a smaller delay between emitted events. The smaller this delay, the greater the influence of the execution time on the delay. This can be shown in a small example. When we specify the output rate to be 1000 events per second this results in a delay of 0.001 seconds. But we measured that the producer takes an execution time of 0.00035 seconds plus the 0.001 seconds delay to send one event. This makes a total of 0.00135 seconds, which is 35% higher than the delay that should be between the sending of two events. Therefore the producer has an actual output rate of 740 events per second. This problem also makes it difficult to assign precise balance scores to an operator. To induce some balance scores it is necessary to have great differences between the arrival rates of the event types. But since high arrival rates are affected more by the problem than low ones, this also changes the proportions of the event types differently. This often leads to an unpredictable balance score. So to run controlled experiments it is important to fix this problem. Our first idea was to find a mean execution time of the producer and then subtract it from the delay that is calculated from the specified output rate. But we could not find an execution time that is generally applicable. Therefore it is necessary to run preliminary experiments before running the actual experiment. In these preliminary experiments, we measure the execution times for every producer for every configuration of an experiment. These execution times can then be entered at the CEP framework and are automatically subtracted from the delays which are calculated from the specified output rates. So the delay for a producer is now calculated according to Equation (4.7).

$$delay = \frac{1}{output\_rate} - execution\_time \tag{4.7}$$

4.3.2 Metric Consumer

To build the prediction models, we need the arrival rate and balance score of one operator that corresponds to the processing time of another operator. The processing time of an operator is already measured by the existing CEP framework. So we only need to implement the calculation and measurement of the arrival rate and the balance score. These new implementations are added to the metrics consumer of the framework.

The arrival rate of an operator is defined as the number of events which arrive at an operator every second. We also need to measure the arrival rate for every event type individually, since these values are used to calculate the balance score later. To calculate the arrival rate of an event type at an operator we use Equation (4.8).

$$arrival\_rate = \frac{t_n - t_{(n-20)}}{20} \tag{4.8}$$

Here t_n denotes the arrival timestamp of the current event and t_{n-20} denotes the arrival timestamp of the event that arrived 20 events earlier. So this means that we calculate the mean arrival rate over the last 20 events. To be able to calculate this we save the arrival timestamps for the last 20 events, that arrive at an operator, individually for every event type. By calculating the arrival rate this way, the calculated arrival rates for the first 19 events are not as accurate as for the following events. Measurements showed that usually there is a rather large difference between the first 19 events and the following events. Therefore, the arrival rate of the first 19 events is set to zero, so that they can be sorted out easier later. To get the overall arrival rate of an operator we then calculate the sum from the arrival rates measured for the event types on this operator. The arrival rates for every event type and operator are calculated at every arriving event, no matter on which operator the event arrives. By this, it is later possible to get the arrival rate of one operator that corresponds to the processing time of another operator.

The balance score of an operator is defined in Section 4.1.2. To calculate the balance score we first need to calculate some intermediate variables. First, we calculate the proportion of an event type regarding the pattern of the operator according to Equation (4.1). Therefore we need to know the pattern of each operator. Since this pattern usually does not change on runtime, it is enough to enter the pattern manually before running the CEP system. This also means that the $proportion\_pattern$ does not change while running the CEP system and can therefore be precalculated. Next, we calculate the proportion regarding the arrival rate for the whole operator according to Equation (4.2). The values of the arrival rate which are needed for this are already measured as explained before. Then we calculate the ratio between the two proportions according to Equation (4.3). As the last step, these ratios are then combined to the balance score of the operator according to Equation (4.4). Because we start to calculate the arrival rates only when there are at least 20 events, there are some arrival rates with a value of zero. This also results in the fact that the balance score can not be calculated in such a case. These uncalculated balance scores get a value of zero assigned so that they can be sorted out easier later. The balance scores for every operator are calculated at every arriving event, no matter on which operator the event arrives. By this, it is later possible to get the balance score of one operator that corresponds to the processing time of another operator.

4.3.3 Network Library

When we started working on this thesis, the Precept II framework used Kafka as its network library. The network library is used to send events and log messages between the different components of the framework. But when we performed our first experiments we discovered that, even with a delay of zero between the sending of events, we could only reach a maximum output rate of around 2780 events per second. The second-highest output rate we could reach was 1050 events per second by using a delay close to zero. To increase these output rates we implemented ZeroMQ as a network library for parts of the framework. The implementation for the operators was done by Henriette Röger, one of the supervisors of this thesis. We then integrated ZeroMQ for our producers, so that they can connect to the new operator implementation. By using ZeroMQ instead of Kafka we could increase the maximum possible output rate from 2780 events per second to 10.000. Currently, ZeroMQ is only used for sending events to the operator whereas all other communication is still using Kafka. But in the future, it might be beneficial to completely replace Kafka with ZeroMQ.

5 Prediction Models

In this chapter, we describe how we build prediction models from the data created in Chapter 4. Therefore we first provide an overlook over the created data and explain how we need to preprocess this data. Next, we explain how we use regression analysis to build different regression models. Here we also decide which of these models has the best performance. Last we show how we build a neural network that can be used as a prediction model. Here we also explain how we need to adjust the parameters of our neural network to get the best results.

5.1 Data Preprocessing

In Chapter 4 we described how to generate the data necessary to build a prediction model. But this data can not be used directly to build the prediction models. We first need to preprocess it. As explained in Section 4.3.2, inaccurate arrival rates and balance scores get assigned a value of zero. These inaccurate values could distort the building of a prediction model. Therefore we delete all data points where either the arrival rate or the balance score has a value of zero. Our experiment normally can not produce a value of zero for these metrics. Therefore we can safely delete these data points without losing valid data.

The CEP framework measures many different metrics. But to build our prediction models we only need the regressor variables and the response variable. In our case, we use the overall arrival rate and the balance score of operator 1 as regressor variables and the processing time of operator 2 as the response variable. We extract these metrics from the collected data and store them into two different arrays. One array contains pairs of the regressor variables and the other contains the response variables. This separation is needed by the frameworks we use for the regression analysis and the neural network.

After building the prediction models we need a way to measure their accuracy. Therefore we need to compare the predicted response variables to the measured response variables. But the data fluctuates and mostly does not assign one unambiguous response variable to a pair of regressor variables. So using this data to measure the accuracy most likely would not work as intended. What we need is one unambiguous response variable for each configuration of the experiment. Therefore we take the mean value of the regressor variables and the response variables for every configuration. This gives us a total of 100 data points which we can use to measure the accuracy of the prediction models. The triangulated surface graph formed by these data points is shown in Figure 5.1. This graph can also be used to compare the expected and predicted values more intuitively.

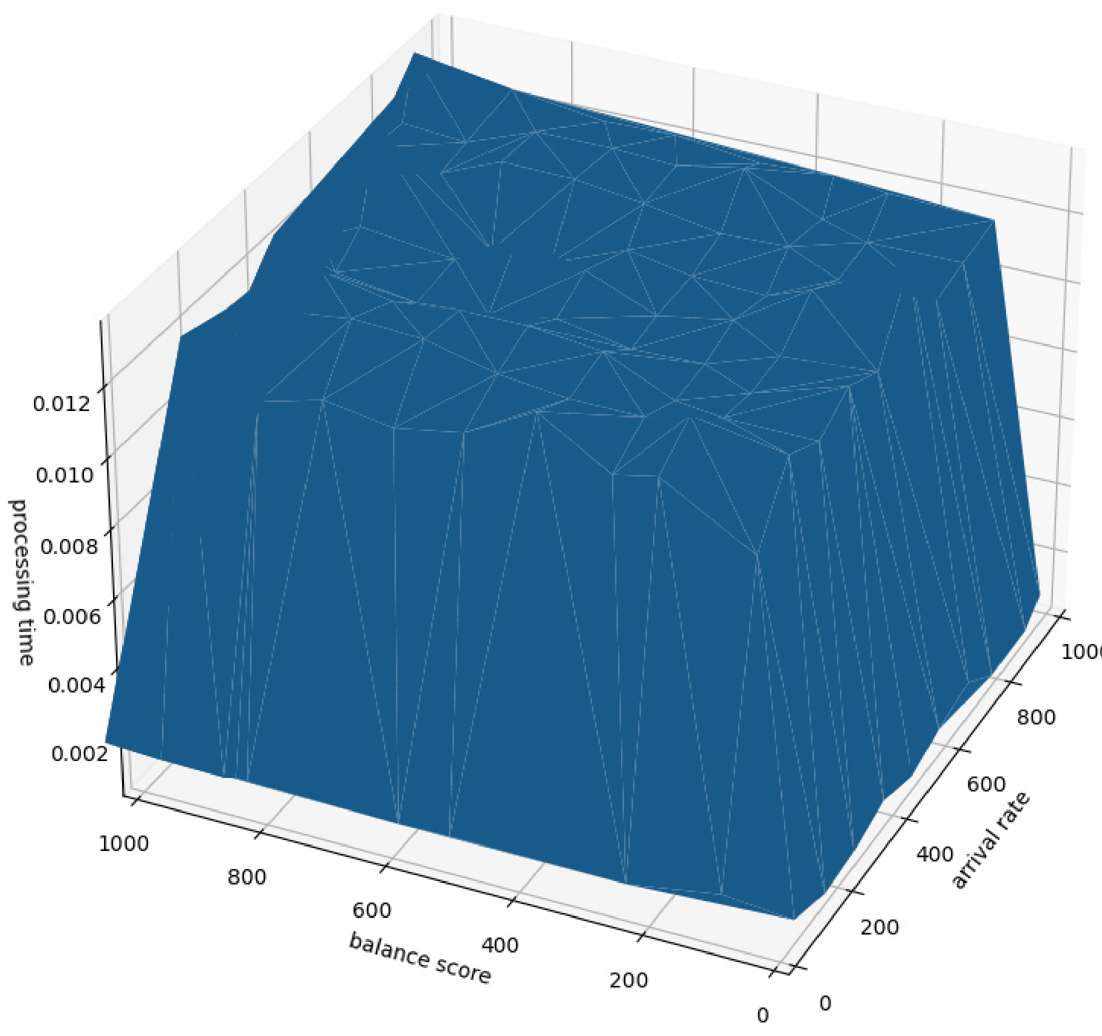

Figure 5.1: Surface graph for the mean values of each configuration of the experiment

5.2 Regression Analysis

One method to build a prediction model is to perform a regression analysis. There are mainly two libraries for python, which can be used to perform a regression analysis: *scikit-learn* and *statsmodels*. Both libraries work pretty similarly, but there are a few differences. One difference is that scikit-learn is significantly faster than statsmodel for datasets with more than 1000 data points [Sri20]. Statsmodel on the other hand provides more statistics than scikit-learn. But these additional statistics are not relevant for prediction. Because our goal is to build a prediction model and we have a total of 1.000.000 data points, we use scikit-learn for our regression analysis.

First, we build a linear regression model. This is the simplest form of a regression model and gives us a first impression of the complexity of the needed prediction model. Since we already preprocessed our data in Section 5.1 we now only need to provide this data to

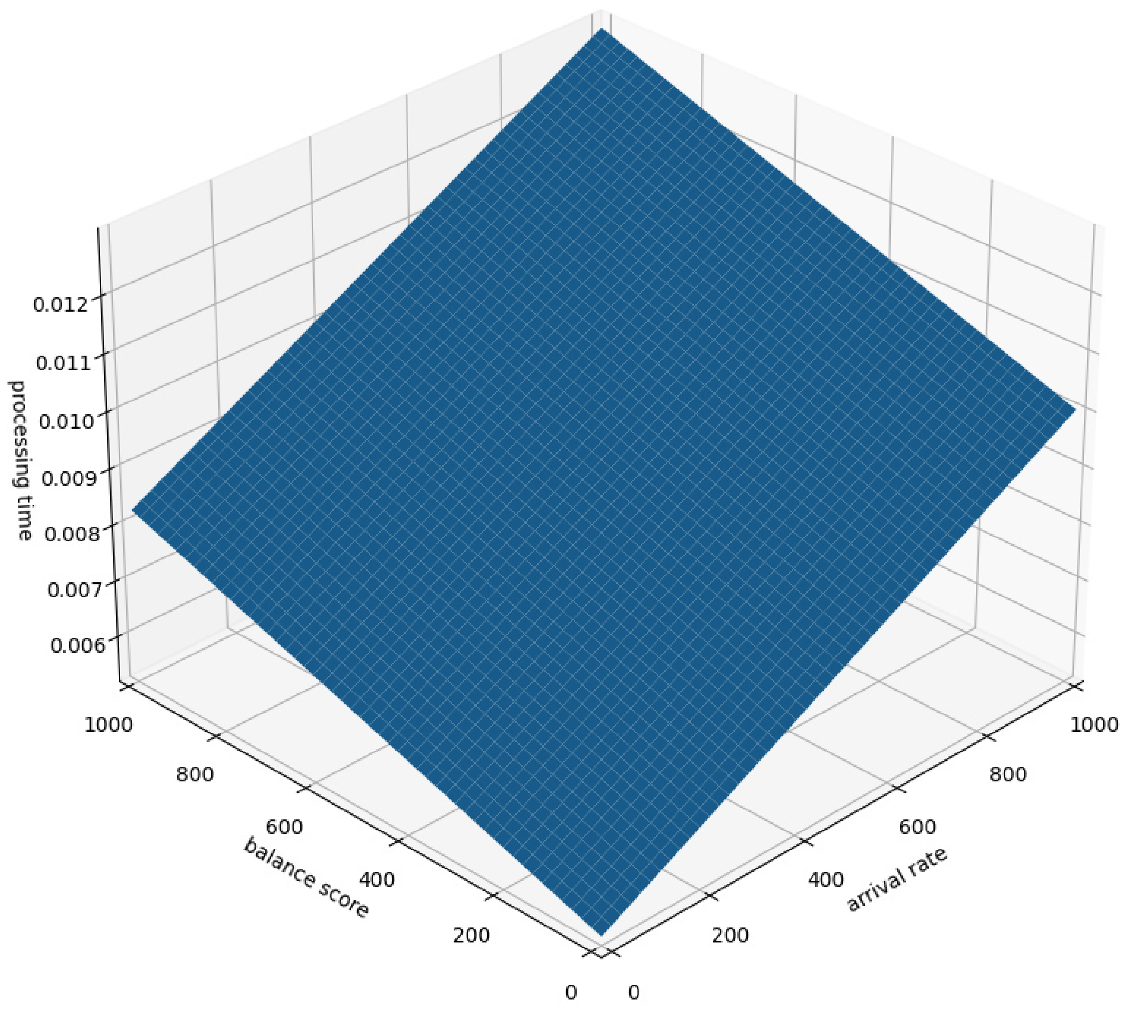

Figure 5.2: Surface graph for the linear regression model

the scikit-learn framework. The framework then builds a linear regression model from the data. The model that was built from our data is described by Equation (5.1).

$$y = 0.000004708x_1 + 0.000002938x_2 + 0.00534446 \tag{5.1}$$

Here y denotes the processing time of operator 2, x_1 denotes the arrival rate of operator 1 and x_2 denotes the balance score of operator 1. Figure 5.2 shows the surface graph that corresponds to this equation. It can be seen that the graph takes the form of a plane, where the processing time rises when the arrival rate and the balance score rise. Here the influence of the arrival rate is a bit smaller than the influence of the balance score. To measure the accuracy of the linear regression model we use the coefficient of determination, which is explained in Section 2.3.3. By using the mean data points we created in Section 5.1 the scikit-learn framework calculates a coefficient of determination of $R^2 = 0.2262$. That means that our model fits the data to 22.62%, which is not very good. This also become clear when you compare Figure 5.1 and Figure 5.2. If the prediction model has a high accuracy these two figures should look pretty similar. But obviously Figure 5.1 is more complex then just beeing a plane as in Figure 5.2.

So, since a linear regression model does not fit our data well enough, we need a more complex prediction model. For that, we can use a polynomial regression model, which adds regressor variables with higher degrees to the regression equation. The scikit-learn framework builds a polynomial regression model basically the same way as it builds a multiple linear regression model. For this, the framework views the regressor variables with higher degrees as individual regressor variables. So we first need to calculate the higher degree versions of the regressor variables for each of our data points. The scikit-learn framework provides a method for that purpose, where we only need to specify the order of the polynomial regression model.

Table 5.1: Performance of the Regression Models

Order	R^2	Build Time	Prediction Time
1	0.2262	0.0973	0.00000099
2	0.4208	0.2047	0.00000099
3	0.6494	0.5082	0.00000099
4	0.8171	0.8757	0.00000099
5	0.8433	1.2256	0.00000099
6	0.7666	1.9035	0.00000099
7	0.5925	2.4135	0.00000099
8	-2.3636	3.4745	0.00000099
9	-0.9657	5.0166	0.00000099
10	-20.7504	7.2353	0.00000100

We built polynomial regression models with orders between two and ten to find out which fits the data best. Therefore we calculated the coefficient of determination for each of these models and compared them. Table 5.1 shows the coefficient of determination for each polynomial regression model we built. First, the coefficient of determination rises till it reaches a maximum of $R^2 = 0.8433$ at the fifth-order polynomial regression model and then decreases again. For the orders between eight and ten the coefficient of determination even gets negative. This means these models do not fit the data at all. So the model that fits the data best is the fifth-order polynomial regression model, which is represented by the surface graph in Figure 5.3. If we compare this graph and Figure 5.1 we can see that they are pretty similar at the upper part. The lower part of Figure 5.3 shows negative processing times, which obviously does not make sense. So to properly use this model in a real application we could view the negative processing times as zero.

We also measured for each model the time it takes to build the model and how fast it can predict a single response variable. For this, we build the model ten times and let it perform a prediction for all 100 configurations ten times. Then we measure how long these processes take and calculate the mean values of these times. Table 5.1 shows these times

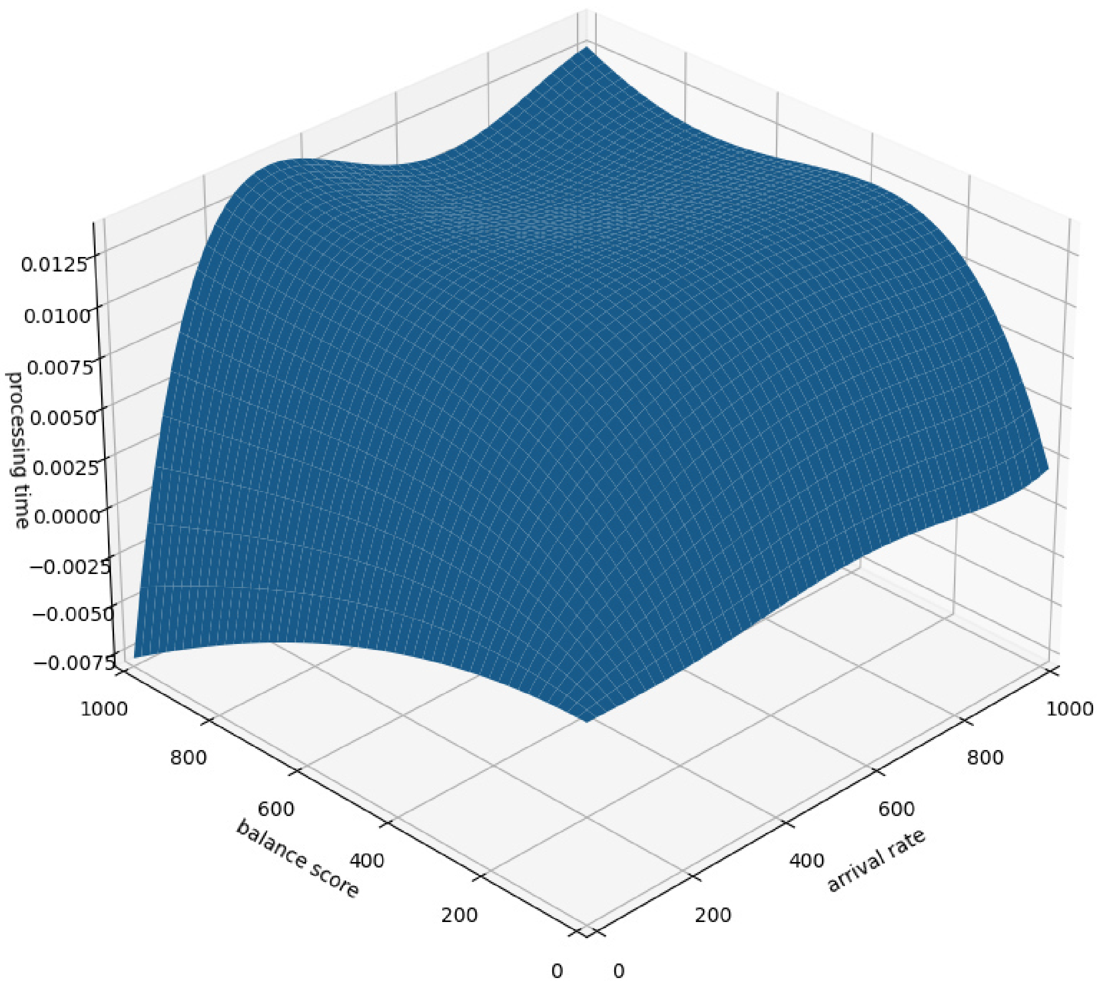

Figure 5.3: Surface graph for the fifth-order polynomial regression model

in seconds. Here we can see that the build time rises with rising order. But the maximum is only 7.23 seconds, which is negligible since we normally only need to build the prediction model one time. So even if the lower-order models are faster this does not represent an advantage in a real application. The time to perform a single prediction is 0.00000099 seconds long for each prediction model. Because there is no meaningful difference for the build and prediction times between the models, the best regression model is the one that has the highest accuracy. In our case, this is the fifth-order polynomial regression model with a coefficient of determination of $R^2 = 0.8433$.

5.3 Neural Network

Our second method to build a prediction model is using a neural network. There are many different libraries for python, that can be used to build and train a neural network. Since there too many different libraries to compare them all, we selected two out of them, which we look into in more detail: *PyTorch* and *Keras*. Keras and Pytorch are both widely used and have good community support. Keras is a high-level Application Programming Interface (API), which can run on top of different backends, for example, Tensorflow and Theano. It has an easy-to-understand and to-use syntax, which results in low development times for building and training neural networks. To achieve this Keras abstracts many details from its backend away. This results in Keras being less flexible and adjustable than its backend. PyTorch on the other hand is a lower-level API, which is much more flexible than Keras. In PyTorch, you can adjust the neural network in much more detail. But this comes at the cost of a less simple and concise code compared to Keras. We also tested how fast PyTorch and Keras are, by building and training a small neural network for each. Here we discovered, that PyTorch took much longer than Keras to train a neural network. Because of this fact and the easier syntax, we are going to use Keras. Since the prediction model we want to build should not become too complex, we do not think that we need the additional flexibility PyTorch offers.

To build a neural network we need to set several parameters. In the following, we first define some initial values for these parameters. Then we adjust these parameters so that the neural network fits our data best. First, we need to decide which activation function the neurons should use. By looking up some tutorials on that topic we found out that for a regression problem mostly RELU is used. The next step is to set the layout of the neural network. Our input layer needs to contain two input neurons, one for the arrival rate and one for the balance score. Next, we need to decide how many hidden layers we want to use and how many hidden neurons they should contain. Usually, one hidden layer is enough to solve the most common problems. Since solving a regression problem should not be too complex, one hidden layer should be enough. One rule of thumb for the number of hidden neurons is, to use less than twice the size of the input layer. Because our input layer has a size of two, our hidden layer contains 3 hidden neurons at the start. Our output layer needs to contain one input neuron for the processing time. The next step is to decide which optimizer we use to train the network. In Section 2.4.2 we describe the SGD optimizier. For the SGD optimizer you need to set a single learning rate that is used for all weight updates. Nowadays the standard optimizer most neural networks use is the Adam optimizer. The Adam optimizer has the advantage that it adjusts the learning rate individually for each weight on runtime [KB17]. This means the Adam optimizer automatically chooses the best learning rates. Therefore we use the Adam optimizer to train our neural network. The last step is to decide the batch size and the number of epochs we want to train our neural network. Usually, a larger batch size leads to shorter epochs but also lower accuracy. To maintain a good balance between the accuracy of the neural network and the time it needs to be trained, we use a batch size of 100 for 10 epochs.

After setting the initial parameters we built this model and tested the accuracy. Keras provides a metric to measure the accuracy of a neural network. But this metric just calculates how many of the predictions are absolutely correct. Since we almost never can assign one definite processing time to a pair of arrival rate and balance score, this accuracy is always 0% in our case. Therefore we use the coefficient of determination as an accuracy measurement instead. This also leads to better comparability between the regression analysis and the neural network. Since the initial weights of the neural network are chosen randomly, the accuracy of two models with the same parameters is still different normally. Therefore we build the neural network ten times and take the mean of the measured coefficient of determinations. For our initial model, we measured a coefficient of determination of $R^2 = -4.9702$ So this initial neural network does not fit our data at all.

Table 5.2: Accuracy for Neural Networks built with different Activation Functions

Activation Function	R^2
relu	-4.9702
sigmoid	0.9622
tanh	0.6553
linear	-14533.4958

Now we tested different combinations of parameters. Here we discovered that if we use the sigmoid activation function instead of RELU the accuracy of the neural network seems to be much higher. Therefore we compare the most common activation functions to find out which ensures the highest accuracy. Since our tests also suggest that a higher number of hidden neurons increases the accuracy, we use ten hidden neurons for this comparison. By this, we can make sure that the neural network actually can produce meaningful results. Table 5.2 shows the coefficient of determination for each activation function. We can see that neural networks built with the RELU and linear activation function do not fit the data at all. Neural networks built with the tanh activation function at least do fit the data somehow, but the accuracy still is not too good. Neural networks built with the sigmoid activation function have the best accuracy and therefore we use this activation function from now on.

Table 5.3: Performance of Neural Networks with different amounts of Hidden Neurons

Hidden Neurons	R^2	Build Time	Prediction Time
2	-0.0005	85.85	0.00063
3	0.4603	78.24	0.00064
4	0.7687	79.91	0.00066
5	0.8756	85.90	0.00065
6	0.9653	87.29	0.00065
7	0.9808	79.53	0.00064
8	0.9766	80.20	0.00061
9	0.9814	85.29	0.00063
10	0.9776	79.74	0,00060

As mentioned before our tests with different combinations of parameters suggest that a higher number of hidden neurons also increases the accuracy of the neural network. Therefore we compare different numbers of hidden neurons to find out which number ensures the highest accuracy. Table 5.3 shows the coefficient of determination for each number of hidden neurons. We can see that the accuracy of the neural network indeed rises with the number of hidden neurons. With seven and more hidden neurons the accuracy stays rather constant with a value around $R^2 = 0.98$ and does not rise further. We also measured the time it takes to train the neural network and how fast it can predict a single response variable. For this, we train the neural network ten times and let it perform a prediction for all 100 configurations ten times. Then we measure how long these processes take and calculate the mean values of these times. Table 5.3 shows these times in seconds. We can see that the build time is between 78 and 86 seconds and follows no clear trend. The prediction time also follows no clear trend and stays between 0.00060 and 0.00066 seconds. So because there is no meaningful difference for the build and prediction times, the neural network with the highest accuracy is the best one. Therefore we use seven hidden neurons in our neural network from now on.

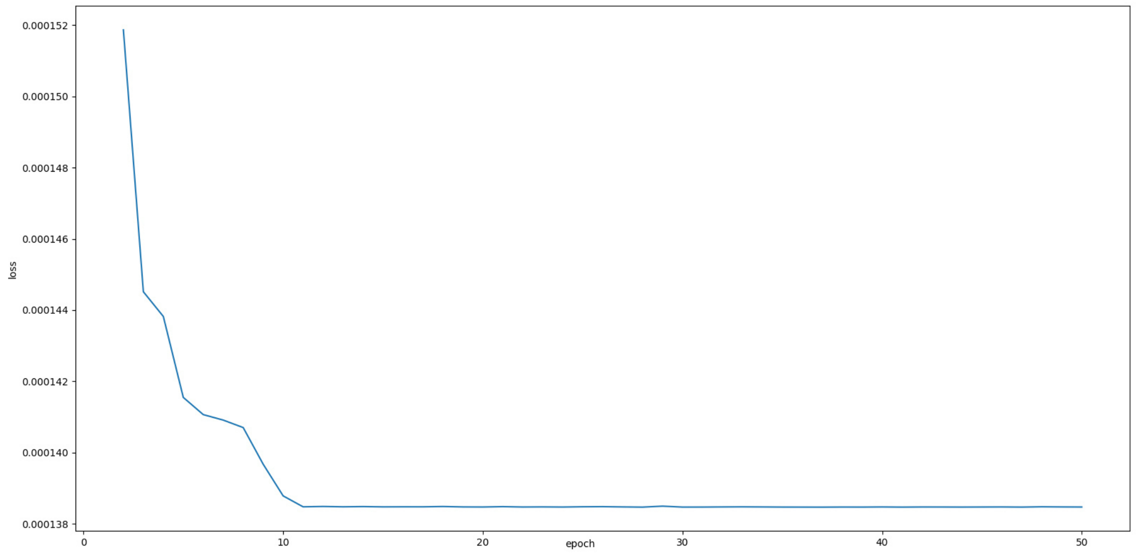

Figure 5.4: Relation between the loss and the number of epochs

Next, we inspect how the number of training epochs influences the neural network. As explained in Section 2.4.2 a neural network tries to minimize the loss over the course of the training. If the loss does not decrease anymore, additional training epochs are not increasing the accuracy of the neural network any further. Therefore we examine the relationship between the loss and the number of epochs. Figure 5.4 shows this relationshship for a maximum of 50 epochs. We can see that the loss decreases greatly within the first ten epochs. After this point, it seems that the loss stays rather constant. To further inspect the loss after ten epochs we zoomed in on the graph. In Figure 5.5 we can see that the loss actually is still decreasing slightly and reaches its minimum after 28 epochs. After this point, the loss fluctuates but does not decrease any further. Therefore it should be enough to train our neural networks for 28 epochs.

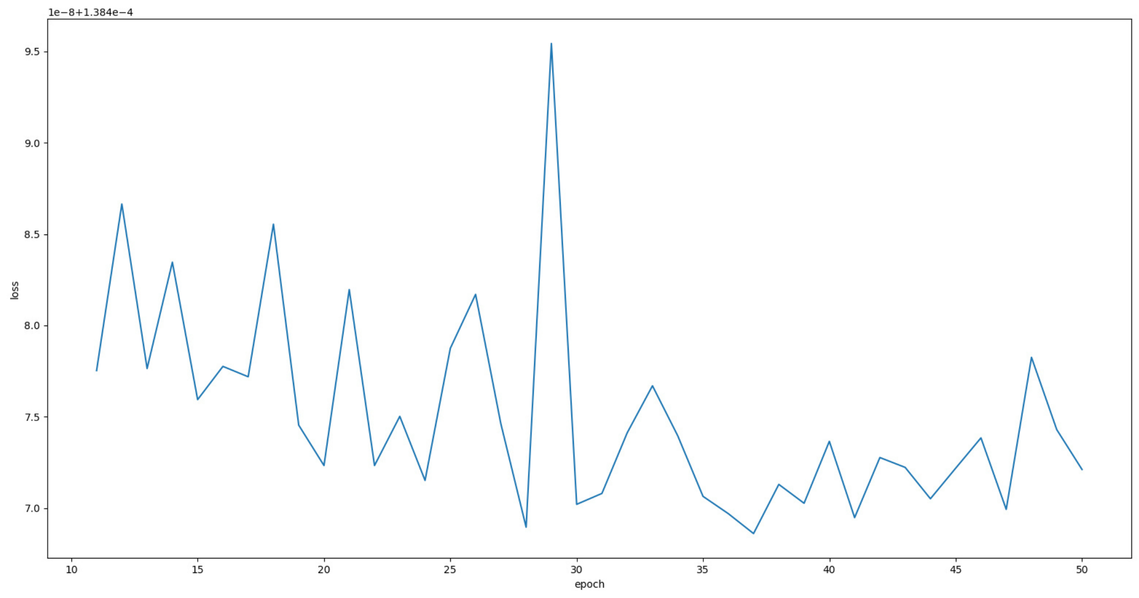

Figure 5.5: Relation between the loss and the number of epochs (zoomed)

Table 5.4: Parameters of the final neural network

Activation Function	sigmoid
Hidden Layers	1
Hidden Neurons	7
Optimizer	Adam
Batch Size	100
Epochs	28

Our final neural network uses the parameters shown in Table 5.4. It has an accuracy of $R^2 = 0.9864$, a build time of 230.78 seconds, and a prediction time of 0.00062 seconds. Figure 5.6 shows the surface graph for our neural network. If we compare this graph and Figure 5.1 we can see that they are pretty similar. This shows that the prediction model we built by using our neural network fits the data pretty well.

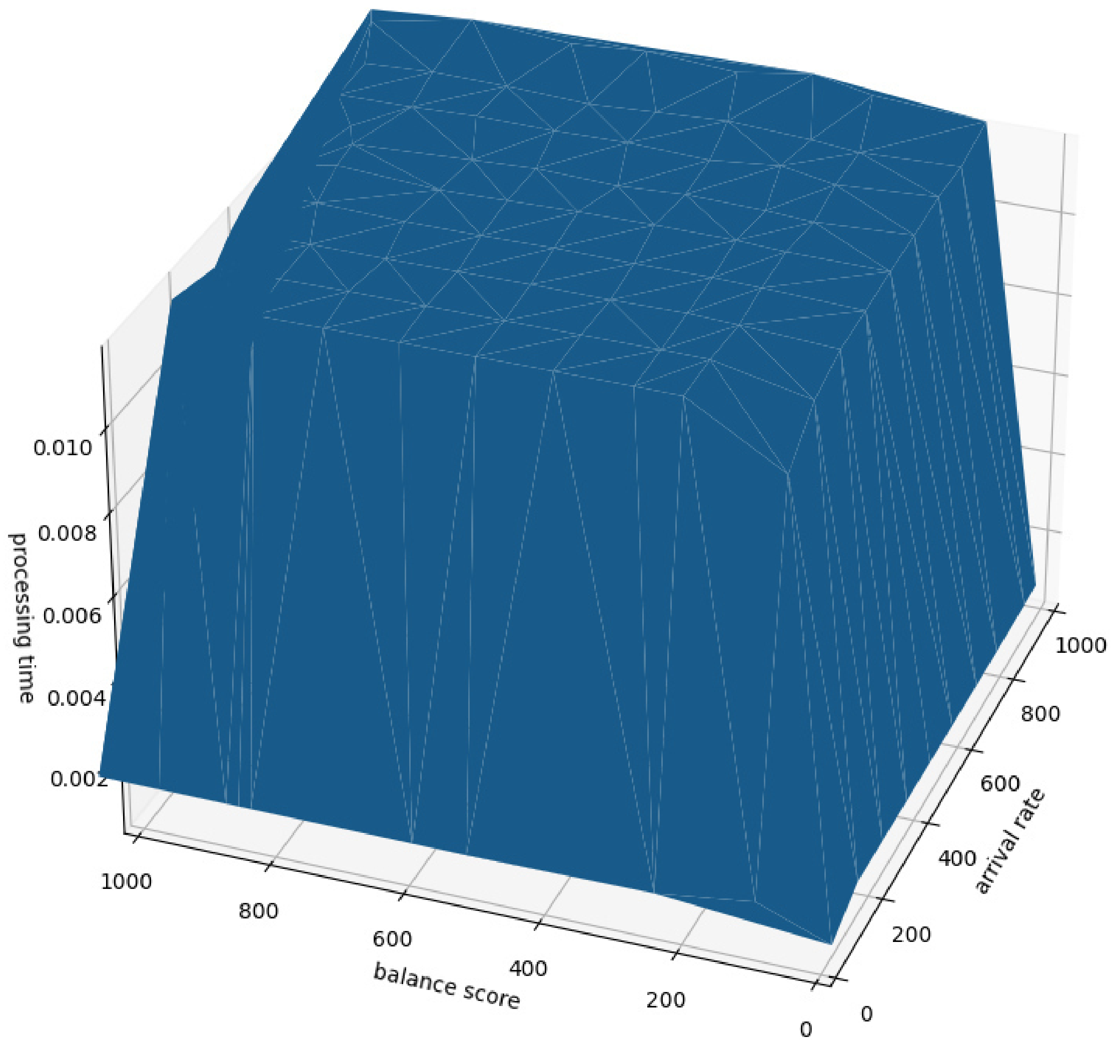

Figure 5.6: Surface graph for the neural network

6 Discussion

In Chapter 5 we used the data generated in Chapter 4 to build prediction models. For this, we used regression analysis on the one hand and a neural network on the other. For both methods, we tried to find the best prediction model. For regression analysis, this is the fifth-order polynomial regression model and for the neural network we figured out the parameters shown in Table 5.4. In this chapter, we discuss and compare these prediction models. We also explain the limitations under which our results are valid.

Table 6.1: Comparison between the Prediction Models of the Regression Analysis and the Neural Network

Method	Accuracy	Build Time	Prediction Time
Regression Analysis	0.8433	1.2256	0.00000099
Neural Network	0.9864	230.78	0.00062

To compare the prediction models we look at the following three metrics: the accuracy, the build time, and the prediction time of each model. Table 6.1 shows the values of these metrics for the best prediction model that we built by using regression analysis and a neural network. We can see that the neural network model has higher accuracy than the regression analysis model. But the regression analysis model has a lower build and prediction time than the neural network model. So which model is better? To answer that question we need to look at the context in that we want to use the prediction model.

The prediction model shall later be used to improve the load shedding of the CEP framework. Here it shall be possible to predict the processing time of an operator and then decide if events need to be dropped. For that, the predictions must be accurate to a certain extent. If the predicted processing time is lower than the actual one, we might not drop events when it would be necessary. This could lead to an overload of the operator and increase the delay above what is acceptable for the CEP system. If the predicted processing time is higher than the actual one, we might drop events when it would not be necessary. In the worst case, important events are dropped without a reason for it. This could lead to a degradation of the results of the CEP system. For the load shedding to work correctly, the processing time needs to be predicted before the event would be processed. To gain an advantage from dropping an event, the prediction time must not be higher than the processing time. But we also need to take into account that often it might not be necessary to drop the event. In this case, an only slightly smaller prediction time still greatly increases the overall processing time for this event. So in fact the prediction time should be significantly smaller than the processing time. Otherwise, the load shedding might increase the overall delay of the CEP system instead of lowering it. The build time normally is not important for the load shedding mechanism, because the prediction model usually is built in advance. So in most cases, we can ignore this metric.

Now we can put the measured values of our prediction models into context. First, we compare the build time of the prediction models. Even if the build time of the neural network is much higher than the build time of the regression analysis model, it is still only around four minutes long. For a process that usually only is run one time before the actual CEP process, this is still really low and does not pose a disadvantage. Next, we compare the accuracy of the prediction models. Both prediction models have an accuracy of over 80% which is pretty good for data that can be so ambiguous. But with an accuracy of ca. 85% for the regression analysis model this still means that in 15% the decision of the load shedding mechanism might be wrong. The accuracy of nearly 99% for the neural network model on the other hand means that only a very small part of the decisions are wrong. So if only necessary events are allowed to be deleted, the neural network model is the better choice. Last we compare the prediction times of the prediction models. If we directly compare the values the regression analysis model is much faster than the neural network model. But as explained before what actually matters is the proportion with regard to the processing time. The mean processing time of our data is 0.009442 seconds. This means that the prediction time of the regression analysis model is ca. 9500 times faster than the processing time, whereas the prediction time of the neural network model is only ca. 15 times faster. So to not increase the overall delay of the CEP system every fifteenth event needs to be dropped when using the neural network model. This means that in phases where the load on the CEP system is relatively low, the load shedding could actually worsen the performance. This effect could be countered by not performing a prediction for every event, but instead only after certain time intervals or when a higher arrival rate is detected. A CEP system which uses the regression analysis model on the other hand only needs to drop every 9500^{th} event. This means even in phases where the load on the CEP system is relatively low, the load shedding should not worsen the performance. So, the prediction time is a great advantage the regression analysis model has compared to the neural network model. Another factor we can compare is how complex it is to build the prediction model. To find the optimal prediction model both methods need some testing. For the regression analysis, this mostly consists of finding an appropriate degree for the polynomial regression model. A neural network on the other hand has much more parameters that need to be adjusted. Finding the optimal configuration of these parameters most likely takes much more time than for regression analysis. The process to build a neural network can also be much harder to understand, if you are not familiar with this theme area. So we can see that the regression analysis model has more advantages on its side. Using this model makes sense when you want an easy building process for the model and want to guarantee that the overall delay rarely gets degraded. However, if events are only allowed to be deleted when this is really needed, the neural network model may be a better choice. The better accuracy of this model ensures that the results of the CEP system are not degraded too much. So all in all which model is better mainly depends on the specific use case of the CEP system.

In the following, we describe the limitations under which our results must be viewed. The prediction models we built are only valid within the ranges of the gathered data. If we try to use them with higher or lower values for the arrival rate or balance score, the predicted processing times might not be correct anymore. Therefore it is important to define realistic boundaries before building a prediction model. We also only used synthetically generated data, where we could ensure a high diversity. Real data is not necessarily as diverse, which might lead to a worse accuracy of the prediction models. Our prediction models also might have different results when used on another machine. A stronger machine can endure higher arrival rates and balance scores before getting overloaded, whereas a weaker machine might get overloaded earlier. We also only examined the interference effect between two operators. The relationship between the regressor and response variables most likely becomes more complex if we increase the number of operators. With huge numbers of operators, it might become difficult to find polynomial regression models that fit the data well. So for more complex CEP systems the accuracy advantage of a neural network over a regression analysis might become much bigger than for our prediction models. Despite all these limitations we still think that the main benefits we described for the two different methods to build a prediction model should still be correct for most cases.

7 Summary and Outlook

CEP applications can be used to analyze information streams. To deal with workload peaks events can be dropped by using load shedding techniques. Here it is important to decide when and how many events need to be dropped. But if multiple operators of the CEP application share a resource, the workload of one operator does also influence the performance of the other operators. Therefore the goal of this master thesis is to develop a model that can be used to predict performance changes of a CEP system caused by interference effects.

To solve this task we consider it a regression problem. A regression problem tries to find the relationship between a response variable and one or multiple regressor variables. As the response variable, we choose the processing time of an operator. As one regressor variable, we choose the arrival rate of an operator. To also take into account the difference between the arrival rates of different event types, we introduce the balance score as the second regressor variable. The balance score measures the difference between the pattern an operator expects and the actual arrival rates of the event types and is represented by a single value. The higher this value, the greater the difference between the pattern and the arrival rates. A balance score of 1 means that the arrival rates and the pattern are perfectly balanced. We also show that the chosen regressor variables actually influence the response variable by performing some experiments.

To solve our regression problem we need to generate and gather data. To get meaningful results this data needs to be diverse enough. Therefore we design an experiment that consists of 100 different combinations of arrival rates and balance scores. For each of these combinations, we set the arrival rate and balance score on one operator and measure the corresponding processing time of another operator on the same node. To be able to perform this experiment we first need to adjust the existing Precept II framework. On the one hand, we add the possibility to easily set different arrival rates for each event type. For this, we use one producer per event type each running in its own thread. We also ensure to generate the correct output rates by taking the execution time of the producer into account. On the other hand, we extend the metric consumer so that it can measure the arrival rate of each individual event type and the whole operator. We also directly calculate the balance score for each event inside the metric consumer.

Next, we run the planned experiment and use the gathered data to build prediction models. For this, we use two different methods: regression analysis and a neural network. For each method, we try different configurations and determine the best prediction model. The best prediction model we could build with regression analysis is the fifth-order polynomial regression model. To get the best prediction model by using a neural network we used the parameters from Table 5.4 to configure the neural network. Then we compare the two prediction models with regard to their accuracy, build time, and prediction time. The build time of both models is low enough that it does not make a difference in a real application. The neural network model has a higher accuracy whereas the regression analysis model has a lower prediction time. By looking at the context in which these metrics are important, we concluded that which model is better mostly depends on the specific use case.

The prediction models we built only look at the interference effect between two operators. So one could build prediction models for the interference effect between more operators. By this, it might be possible to generalize our results for CEP systems with more operators. Our prediction models also only look at the interference effects between operators. But to actually use a prediction model in a load shedding mechanism, the model needs to look at the workload of all operators on a node. Therefore, a model would need to be built for each operator that predicts the operator's processing time by taking into account the arrival rate and balance score of each operator on the same node.

Bibliography

[BK09] A. Buchmann, B. Koldehofe. “Complex Event Processing.” In: *it - Information Technology* 51 (Sept. 2009), pp. 241–242 (cit. on p. 15).

[CM12] G. Cugola, A. Margara. “Processing Flows of Information: From Data Stream to Complex Event Processing.” In: *ACM Comput. Surv.* 44.3 (June 2012) (cit. on p. 15).

[EB09] M. Eckert, F. Bry. “Aktuelles Schlagwort "Complex Event Processing (CEP)".” 2009. URL: http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:19-epub-14902-9 (cit. on p. 15).

[HBN13] Y. He, S. Barman, J. F. Naughton. “On Load Shedding in Complex Event Processing.” In: *CoRR* abs/1312.4283 (2013) (cit. on pp. 16, 17).

[Hea08] J. Heaton. *Introduction to Neural Networks with Java*. Heaton Research, 2008. ISBN: 9781604390087 (cit. on p. 25).

[KB17] D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. 2017 (cit. on p. 46).

[Luc02] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. 2002. ISBN: 9780201727890 (cit. on p. 16).

[MPV12] D. C. Montgomery, E. A. Peck, G. G. Vining. *Introduction to linear regression analysis*. 2012 (cit. on pp. 19–22).

[Nie15] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/index.html (cit. on pp. 22, 23, 25).

[Ost12] E. Ostertagová. “Modelling using Polynomial Regression.” In: *Procedia Engineering* 48 (2012). Modelling of Mechanical and Mechatronics Systems, pp. 500–506. ISSN: 1877-7058 (cit. on pp. 21, 22).

[RC10] D. Robins, Csep. *Complex Event Processing*. Mar. 2010 (cit. on p. 15).

[SBR19] A. Slo, S. Bhowmik, K. Rothermel. “eSPICE.” In: *Proceedings of the 20th International Middleware Conference* (Dec. 2019) (cit. on p. 17).

[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*. Association for Computing Machinery, 2020, pp. 109–120. ISBN: 9781450380287 (cit. on p. 16).

[Sri20] H. Srinivasan. “Linear Regression in Scikit-learn vs Statsmodels.” May 2020. URL: https://medium.com/@hsrinivasan2/linear-regression-in-scikit-learn-vs-statsmodels-568b60792991 (cit. on p. 42).

[Syk93] A. O. Sykes. “An introduction to regression analysis.” In: *Coase-Sandor Institute for Law Economics Working Paper No. 20* (1993) (cit. on pp. 20, 22).

[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 (cit. on p. 17).

All links were last followed on May 17, 2021.

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.

Stuttgart, 28.05.2021, [illegible]

place, date, signature