Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 228

Performance Profiling of a Parallelization Framework for Complex-Event-Processing Operators

Valentin Siegert

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Dipl.-Inf. Ruben Mayer
Commenced:	April 22, 2015
Completed:	Oktober 22, 2015
CR-Classification:	C.2.4; C.4

Abstract

The Internet of Things (IoT) is an upcoming technological complex of themes, whereby the recognition of complex events, happened in the real world, is no more neglectable for any new application. Complex Event Processing (CEP) builds a middleware to generate due a operator network complex events for a application out of simple input events from e.g. some sensors and thereby it should stay in a real-time latency budget at all. Since we concentrate on parallelized CEP operators and more especially on the framework PACE, the latency of one operator can be influenced by several factors, e.g. adapting the parallelization degree and or or using batch scheduling. To adjust the different factors more perfectly, we need to know the latency of the splitting and the merging before the specific situation comes up in the real process, whereby the prediction of the splitting's latency is a more interesting question. By analyzing the latency of the splitting in the framework PACE we determine that the most influencing parameter of the latency is the number of opened selections and give a solving approach by predicting the latency of the splitting with regression learning and some time series predictions. Finally we will see that our predictions deliver nice results without outliers and that our approach is simple enough to not generate a higher runtime or rather a higher latency by asking for and calculating the prediction.

Kurzfassung

Das Internet of Things (IoT) ist eine aufkommende technologische Thematik, wobei die Erkennung von komplexen Ereignissen, welche in der realen Welt geschehen, für neue Applikationen nicht mehr zu vernachlässigen ist. Complex Event Processing (CEP) erstellt eine Middleware um Mittels eines Netzes von Operatoren für eine Applikation komplexe Ereignisse aus, von z.B. Sensoren erkannten, simplen Ereignissen zu generieren und dabei sollte es im Gesamten in einem Echtzeit-Latenzbudget bleiben. Nachdem wir uns auf parallelisierte CEP Operatoren konzentrieren und genauer auf das Framework PACE, kann die Latenz eines Operators durch verschiedene Faktoren beeinflusst werden, z.B. durch die Veränderung des Parallelisierungsgrades und/oder durch das Einsetzen von batch scheduling. Um die unterschiedlichen Einflussfaktoren besser einstellen zu können müssen wir die Latenz des Splittings und des Mergings vor der bestimmten Situation kennen, wobei die Vorhersage der Latenz des Splittings eine weitaus interessantere Fragestellung ist. Wir haben bei der Analyse der Latenz des Splittings im Framework PACE herausgefunden, dass der am meisten beeinflussende Parameter die Zahl der offenen Selections ist und geben dahingehend einen Lösungsvorschlag um mit Regression und time series prediction die Latenz des Splittings vorherzusagen. Schließlich werden wir sehen, dass unsere Vorhersagen gute Ergebnisse ohne Außreiser ausgeben und dass unsere Lösung simpel genug ist um bei der Erfragung und Berechnung der Vorhersage keine längere Laufzeit bzw. größere Latenz zu generieren.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The Internet of Things (IoT) is an upcoming technological complex of themes with many new possibilities and challenges to deal with. It is mostly the term for getting all things we use in our everyday life connected to the Internet and giving them at least some sort of thinking. It is a kind of thinking, because at least every thing gets personal suited for the end-user and or or is maintained by itself at practical work. We call such things "smart things". On the assumption that we just equip any device with a little chip and or or with a sensor, we can process all generated data, with e.g. cloud computing, to control those smart things remotely. The IoT is one of the topics of research, where as a result every new application or even sensor will rise our level of comfort.

When we consider any application using smart things, we note that even an application using just one type of sensors in the real world to gather information could be busy by processing all incoming events. Thus the application does not stay in real-time computing. So it is nice to have a system between the application, or even the user, and the sensors. Such a middleware is realized by a Complex Event Processing (CEP) System. Those CEP Systems are accomplished in a distributed network of operators, the so called operator-graph. This structure is needed for striking faster computing time and operating on a extraordinary wide field of sensors to gather the needed information.

To handle a specific situation just in time or even give the user the chance to react in time, our applications need to compute in real-time. Supposing such a low latency for our application, concludes that also our whole system of sensor, CEP and application must gather, provide and infer information in real-time.

In the physically world, the workload of new events triggered by sensors is a quite high burden. Hence the nodes of the CEP System, the operators, shall be parallelized. A parallelized framework is in our approach quite similar to map reduce and every operator node is replaced by this model. So in our approach the input stream received by an operator node and the first instance is the so called splitter. This splitter splits the stream to interchangeable instances of the same operator. Those instances working like the specific operator before parallelization. To get just one output stream in beyond, a merger will merge the outputs of the instances to one output stream.

For staying in real-time we have a confined latency budget, which is deployed to all operators and so to the parallelized framework of one operator node. In preceding work the instances of the operators were investigated in regard to such a latency budget. It was determined by concerning latency bounds, that the resources, in this context the computing power and the CEP network, can be adjusted by adapting dynamically the parallelization degree and by conducting batch scheduling. [MTKR15]

With a runtime environment in the framework, we also have the ability to assign a latency budget to one specific node. This budget allows to save resources for processing the events and still stay in real-time computing. For instance with the trade-offs (1) buffering level or rather buffering delay versus resources, which is known as parallelization degree and (2) communication overhead versus processing delay, which is known as batch scheduling. So we can adjust only parameters at the part of the operator instances. To comply the specific latency budget in one parallelized operator, we need to know the latency of the splitter and the merger, because then we know the latency, which is left for the operator instances. For adjusting the latency of the operator instances a specific lead time is needed, because for example the starting of a new instance will take some time. Thus there is a need to predict the latency of the splitter's splitting, because it will help to react nearly perfectly on every upcoming situation. Further on it is one step to have no longer a problem to stay with a parallelized CEP System in real-time computing. In this work we want to concentrate on the prediction of latency of the splitting. This prediction can be taken to adjust the framework in a way that the defined latency budget is not overdrew and more in detail we can easily change our parallelization degree and instruct our batch scheduling in a better kind of way.

In the following the content is structured in that kind, that we will first introduce in chapter 2 the Complex Event Processing System and the model of our parallelized framework. In addition we define the terms latency and latency budget in our context and give a overview on related work at CEP systems. Afterwards we will term our special issue, the LoS issue, and analyze what are influencing parameters and how we want to deal with it in chapter 3. In chapter 4 we will introduce some prediction techniques and compare different models of the same technique, to specify which model should be the best for our purpose. Thereby we will use them all for our solving approach of the LoS issue. Beyond we utilize those models to get a solution for our problem. The final solving approach will then be invented in chapter 5. After that, in chapter 6 we will evaluate our system with a mostly realistic scenario, to get a view how it works in practice, and discover a naive approach for comparison with our approach. Finally in chapter 7 we will summarize the results of this work and give a prospect, how we could use this in future work.

2 Complex Event Processing

In this chapter, we first introduce the model of a CEP system and give some overview, what is also possible with CEP systems by linking to related work. Furthermore we conclude the relevance of parallel CEP systems and brief the model of our framework. As a slide-in we will conclue this chapter with the definition of the terms, latency and latency budget in our context.

2.1 CEP Systems

Complex Event Processing (CEP) was originally invented by D. Luckham in [Luc02]. According to [BK09] CEP is the state-of-the-art to detect and react on events in situational applications. Moreover [EB09] says that CEP has more than one roots at research and that CEP is moreover a general term of methods, techniques and tools for processing events in the same time as they are happen. As [KKR10] profiles Cordies, we have a method to distribute a CEP system and are not limited due a fact of a centralized system or a synchronous communication. Further on our comprehension of a CEP system builds on the system models in [VKR11, KORR12, SKRR13, KMR$^+$13, OKRR13, OKR$^+$14b, MKR15, MKR14]. As to see in figure 2.1, to detect events in the world, the basis of every CEP system are the sensors or rather sources. The sensors may also be realized in a so called sensor network. As the name terms, a few or likely more sensors are connected in one network. It is recommend because the CEP system just need one stream for more than one sensor. Thus it is easier to handle complex situations in the system and so in the application. Next, is to be named, all sensors will push the events through a event stream to the actual CEP system, which provides a network of operators. Finally, after processing all events, the more complex events will reach the consumers or rather the applications.

So the heart of a CEP are operators, where one of those ω process the incoming event stream I_ω to more complex events and deliver them in the outgoing event stream O_ω. This is also called the correlation function and defined as $f_\omega : \mathbb{E}_{in} \mapsto \mathbb{E}_{out}$. To get a better understanding we could think about a traffic monitoring system as in [MKR15, MKR14].

There is therefor a distance where passing other vehicles is prohibited. Thereby it is only important to print a ticket and punish the prohibition of passing, when one car passes another one. So an application wants to know whether someone passes another car in this zone or not. For instance, there are two sensors monitoring the line and moreover to see as s_1 and s_2

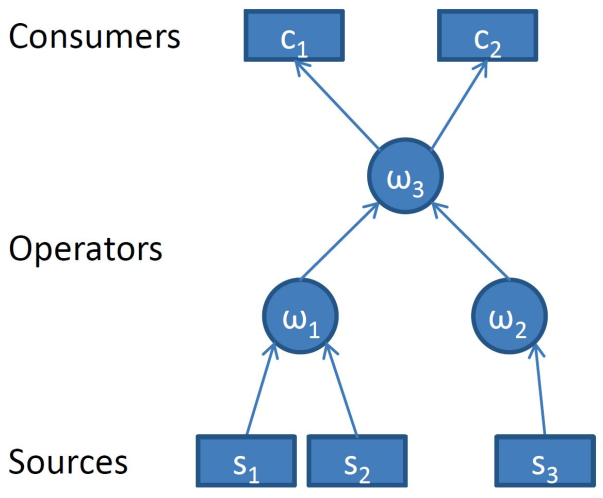


Figure 2.1: Operator graph of a CEP Network connecting sources, operator and consumers by event streams. [MKR15]

in figure 2.1. You can assume hereby also the sensor s_1 as pos_1 and the sensor s_2 as pos_2. Furthermore we need as in figure 2.1 two steps in our event processing. The first operator ω_1 detects the incoming and leaving of every car at the distance of passing prohibition by analyzing the time difference between a pos_1-event and the corresponding pos_2-event. It results those two events to one event of when they arrived and how long they stay at the line. The second operator ω_3 now compares the time windows of each vehicle and pushes an event to c_1 if and only if a vehicle passes another one in the zone of prohibition.

For that fact it is very important that the CEP system is consistent, i.e. there are only true-positves and true-negatives. This means there are in the example of traffic monitoring neither unjustified tickets printed (false-postive events) nor transgressors unpunished (false-negative events).

Moreover CEP systems can be deployed also in other scenarios. So for instance, it was mentioned to take the advantage of CEP systems and benefit by the communication at multiplayer online games [KTKR10], which use peer2peer systems. Another example is the natural language processing for a crowdsourced road traffic alert system in [AGPS15]. On an other side, CEP systems are also very interesting to work with queries at different locations of the same user, so it is commonly to achieve a sort of CEP system, which is mobility-aware. One example for such a mobility-aware CEP system is given in [OKR$^+$14a]. To be mobility-aware it takes some advantage of the work of [OKRR13], where the migration of parts of an CEP system was focused and a method to place and migrate was presented. Thereby it uses the knowledge of an CEP system to improve live migration techniques and save time and bandwidth in the infrastructure. Also the point of moving range queries play a central character of being mobility-aware, because the interest radius of information at a mobile user is mostly linked to its location and as he is mobile, he changes his location during time. Thus the applications on a mobile device consume information at different locations and always need informations on a given radius around their actual location. [KORR12] adopts the concept of range queries, propose a mobility-aware event deliver semantics and present a corresponding execution model. Thus it invents moving range queries to CEP systems. Moreover the events in a mobility-aware context needed mostly to be detected in a consumer-centric manner to hold low latency and high quality of the results. To have the possibility of reuse some of events for another consumer or rather user, [OKR+14b] introduced the RECEP to increase the scalability of mobile CEP systems. It thereby offers methods to efficiently reuse computations in a mobile context and withal even save resource requirements.

As CEP is the paradigm of choice to use as a event processing middle-ware or rather infrastructure it can be interesting, who can access specific data and gather informations out of specific events. To preclude unauthorized access on delivered informations over the Internet or the distributed CEP system, it is recommendatory to apply a access policy for the different events in a stream. To consolidate such a access policy for CEP systems, [SKRR13] presents a fine-grained access management for CEP systems. Furthermore a CEP systems should be fail-safe and support a strong reliability at the delivering of events. Therefore [VKR11] presents a replication scheme to hold strong reliability in a CEP system, thus a failure of one single operator node should not impact the correctness of the results. Because strong reliability means that each complex event is detected and delivered to all consumers or rather applications for exact once. Due the need of strong reliability in a CEP system, a possible rollback-recovery is auspicious, because even if one node failes and no others replicas are alive, the CEP system should hold reliability and consistency. [KMR+13] shows that we even do not need any checkpoints to apply such a rollback-recovery. The clue for no need of checkpoints is to use savepoints in time, when the execution solely depends on the incoming events. The incoming events are reproducible due the prior operators, thus the operators state can be saved without using checkpoints.

2.2 Parallelized CEP

Parallelization is a very interesting topic for everyone, who is engaged with computer science. A parallel CEP is not just a temporary fashion in cause of everybody wants to parallelize near everything. Only a parallel CEP can work with low latency and keep a buffer limit at low costs. The applications getting more an more complex with regard to more and more sensors on the world and the world we live in getting faster and faster. So one operator could easily be overwhelmed with a lot of events pushing from his sensors at nearly the same or also the same time. In fact of this overwhelming, there are only two options for a linear CEP system. Either delete some not grave events, which implies a not consistent processing, or just buffer all events till every event is processed, which entails a high latency at a high workload.

Therefor a parallel CEP system is the answer. To this point more than one idea for parallelize an CEP system is invented. They can be separated in the intra-operator parallelization and the data parallelization. We will now go on in this section with three subsections and describe the termed ideas of parallelize a CEP system. Thereby we closely refer to [MKR15, MKR14].

2.2.1 Intra-operator parallelization

As the name terms, in intra-operator parallelization the site where the parallelization degree is scheduled are the processing steps. Also know from pipelining, the steps, which can be run in parallel, are identified by scrutinizing the query. Thus the states and transitions of the operators are deduced. One of the most common ideas, how a CEP could be parallelized with intra-operator techniques, is Siddhi, which is presented in [SGLN+11]. Both approach have their difficulties with high event rates. For instance in the scenario of traffic monitoring, we consider four different variables in our query. For each car one variable for the beginning of the line and one for the end. In cause of the interest to punish passing another car, there are in general always pairs of cars processed. So finally we can have there a parallelization degree of four with intra-operator parallelization. For a high event rate this is not sufficient, as to see in [MKR15].

2.2.2 Data parallelization

The data parallelized approaches concentrate more on the data streams. Therefor the input stream of one operator is split into partitions, which can be processed parallel by a number of interchangeable instances of one operator. Figure 2.2 gives an overview of such a framework with a splitter and a merger in front or rather behind the operator instances. According to that, this framework exchanges one operator node. The operating logic states a partitioning model, how the splitter has to build the partitions and send those to the operator instances. There is also a runtime environment for controlling the operator execution, overviewing the accessible instances and maintaining the parallelization degree. Subsequently the merger warrants a ordering of all produced events, if this is necessary for the following operators or consumers. This could be realized by assigning sequence numbers to the events. Some examples for such a data parallelization can be found in [BDWT13, sto14, MBF14, CCA+10, BMK+11, NRNK10] to just name some of the available data parallelization approaches.

The fundamental difference between those approaches is the underlying partitioning model. So far there are three different partitioning models available key-based, run-based and pattern-sensitive. The key-based partitioning is the prevalent one and for example to find in [sto14, MBF14, CCA+10, BMK+11, NRNK10]. It partitions the incoming stream by keys, whereby the key is encoded in every event. The parallelization degree is as a consequence limited to the number of available keys and the membership of a specific event to one pattern must not depend on the appearance on other events. But the membership of the events in the traffic

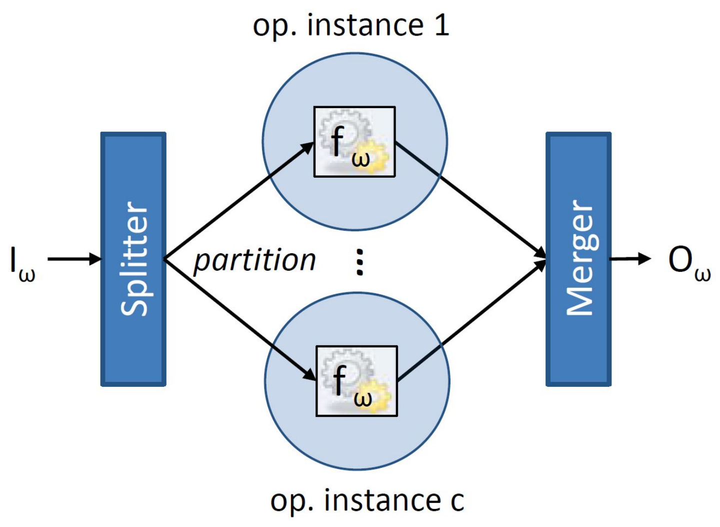


Figure 2.2: A CEP Operator parallelized with data parallelization. [MKR15]

scenario for instance are strictly related to the appearance of other events. Another point is that even a common key has not to be available for one specific event. RIP in [BDWT13] is a run-based approach, which splits the stream in batches, whereby each batch is large enough to fit a match to a queried pattern. This can cause a massive communication overhead, if patterns fluctuate in their size and the approach is unable to detect patterns of an unknown size. As considered in [MKR15] the run-based and the key-based approaches are not consistent enough, in cause of their named disadvantages and the point that they can not parallelize some types of operators consistently. This analysis can be found in [MKR15] and moreover it can be seen that a pattern-sensitive partitioning model is free of these disadvantages and can parallelize all compared operator types. Thus we concentrate here on the pattern-sensitive model. Here the input stream is split in selections which enclose the patterns to be detected. There are one or more selections included in one partition. That means that two selections can be processed independently and parallel on different instances. An event is consumed, when all operator instances, where the event has been assigned to, acknowledged it and deleted it from their splitter queue. So for instance, in our traffic scenario a car will always open a new selection with its event of type pos_1 and will close this selection with its event of type pos_2. If a car arrives at our traffic scenario, the new event of pos_1 will be part of every selection, which is opened when the event arrives. Thereby the event influences the detected pattern of one specific selection due the fact if there is also a event of the type pos_2 of the same car in the selection or not. If it is so, it can be detected that the car, which events of type pos_1 and pos_2 are in another selection, overtook the car of the specific selection.

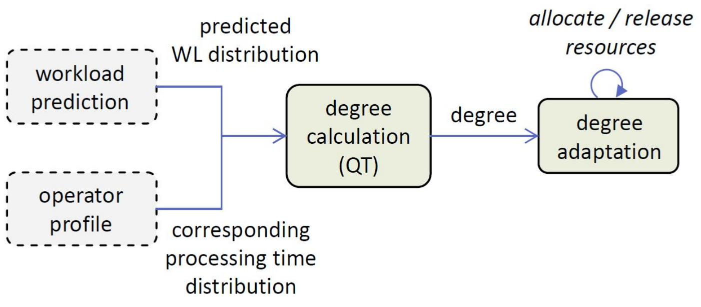

predicted WL distribution
allocate / release resources
workload prediction
degree calculation (QT)
degree
degree adaptation
operator profile
corresponding processing time distribution

Figure 2.3: The approach of the adaption of the parallelization degree. [MKR15]

2.3 Framework: PACE

For evaluating later on and for understanding the future progress it is required to name, what the framework we are working with is about.

In [MTKR15] the framework got the name PACE. It is a parallelized CEP network on the model of data parallelization. The event stream splitting ensued with a pattern-sensitive splitter. Moreover it uses two other methods to handle the parallelization so good as possible. Thereby it mostly tries to realize the context of preferable real-time computing and low computational costs. One of those methods is the dynamical adaption of the parallelization degree and the other one is the scheduling of the selections in means of the available operator instances.

This section will go on with a brief of the adaption of the parallelization degree in PACE. Then we describe the work of the splitter and the merger in PACE more in detail. Further we go on in this section to mention two different scheduling methods of the splitter, who schedules the selections in means of the available operator instances.

2.3.1 Parallelization Degree

There is for fast and low cost computing a need of dynamically adapting the parallelization degree. As mentioned in the introduction in chapter 1, it is absolutely necessary that the CEP systems work in real-time, but also just need minimized computing power. Therefor we aim to have always the optimal parallelization degree and in the fact that in practice the incoming event rate is dynamically changing, we need to adapt our parallelization degree also dynamically. The optimal degree is here the minimal that keeps the assigned buffer limit with the required probability. The buffer limit of one node is linked to the underlying hardware and may also depends on the used programming language and the used data structures. The most influence on the exact buffer limit has the underlying hardware, so [MKR15] takes for adapting the parallelization degree an assigned buffer limit.

The invented approach of [MKR15] defines, how the adaption of the parallelization degree in PACE works. As to see in figure 2.3 the approach is split in four entities. The workload prediction and the operator profile give the knowledge to calculate the optimal degree of parallelization. Further on the actual degree of parallelization could be adapted by the knowledge of the optimal degree.

The workload is predicted by an model of a time sliding window on different time slices. The time sliding window ends at a specific time and shifts about a given frequency. The length of the window is also defined before, but should provide enough time slices to calculate a established distribution of the inter-arrival time of events. The distribution of the inter-arrival time has the influence in the prediction of the future workload at a specific time. Thereby the prediction of an future event becomes ascertained by the workload at the latest measured time slice.

The given workload at a specific point of time for adapting the parallelization degree is quite important, but the perfect degree is also influenced by the processing time of the operator. Therefore the operator is profiled before the actual run-time. The measurements of the processing time of the given operator with different probably real workloads, lead to a profile of the operator. This profile could be used to predict the processing time at a specific workload. So if we know a predicted workload, we could also predict the corresponding processing time.

With these two influences, the optimal parallelization degree can be calculated. The approach concentrates thereby on the probability, that one instance will hold the assigned buffer limit. The buffer gets filled up, when the processing time is appreciable longer than the inter-arrival time of events. If the probability of staying with the fullness of the buffer in the limit is too small, the parallelization degree is increased. This increasing stops, when the minimal degree is found, where the probability is high enough. When the probability is high enough at the beginning of the calculation, the degree will be decreased until the minimal degree is found. Note that the parallelization degree has to be recalculated, whenever the predicted workload changes.

The adaption of the degree is the most simplest part of the approach. Thereby the degree is changed to the optimal parallelization degree. If there are to much instances, the surplus of them will be canceled. More interesting is the part, where more instances are needed. The adaption provides that probably some instances are canceled but not killed yet. If this is the point, the approach recalls first such canceled instances. When there are no more instances available for recalling and there is still a need of more, the adaption will start new instances.

2.3.2 Splitter and Merger

The splitter and the merger are the instances we did not explained more in detail yet. The central point of the work concentrates on the splitter, so it is absolutely necessary to talk over their practical work at the runtime of PACE. Thus we see their work more in detail at this subsection. Therefor we take a look on the processing of an arrived event at a CEP node in PACE.

When a event arrives at a CEP node, first of all it will reach the incoming event queue of the splitter. Then, when the splitter begin to work with the event, it will first decide if this event opens a new selection. Afterwards it will schedule the event in order of the underlying scheduling method, to decide whether an event should be send to an specific operator node or not. It will decide the preferably best distribution over all instances, thus stay with all computing in the given latency budget. Note, that if the scheduling strategy is quite simple, the latency budget will not play a role for the decision. Finally it will decide, if the event closes an existing selection. The decisions upon opening and closing selections are based on the user-defined query or rather the functional definition of this node. We call those decisions by the predicates PStart and PClose. Note, that one specific event can be part of many overlapping selections and that the overlapping of selections is not an exceptional case, but the mostly normal case. Moreover it needs to be mentioned that PStart is simply evaluate once a time for each event, but whereas PClose is evaluated for each pair of all opened selections and the specific event.

After this procedure the event stays in the outgoing queues of the splitter. Every operator instance possesses its own outgoing queue at the splitter. The event is copied to all queues in dependence on, whether it is send to the specific instance or not. The event will be removed from one outgoing splitter queue, when the applied instance processed the event. All instances process their incoming events sequentially in the scope of each selection it is part of. The finally step of the event, is the point, where the splitter removes the last occurrence of the event in any outgoing queue. When a selection is closed by this event, the splitter will delete the specific selection or rather selections from is set of opened selections. Hence the merger starts to reorder the selection result with all other closed selection results to an outgoing stream. Thereby, most commonly, the outgoing events are in order of the arrival time of the applied start event of the selection, which resulted in the outgoing event.

2.3.3 The Splitter's Scheduling

As the inter-arrival time of the events and so on new selections could dynamically change, we need a scheduling of the new selections by the splitter. This scheduling should determine, which operator instances have to work with the new selection. In this connection we can think about two different scheduling methods, a Round-Robin scheduling and a predictive batch scheduling. Note that there could be of course more ideas for scheduling the selections. But rather it is to say, that we prefer the predictive batch scheduling of [MTKR15] for PACE as it is the most auspicious so far.

The Round-Robin is the probably easiest scheduling method so far. It assigns each operator instance one selection in one round. When it has assigned one selection to all running instances, the scheduler begin its round again at the first one. If there are the same number of instances as the number of selections, all instances compute one selection at a time. Else if there are more opened selections than instances every instance has to work on around $\frac{\#selections}{\#instances}$ selections at the same time. Of course the selections are not split in a smaller part. So every selection will be adapted by exact one instance. The fraction builds the common number of selections at one instance.

The predictive batch scheduling of [MTKR15] differs there a lot. While Round-Robin simply assign every instance one selection, it usually generates a communication overhead. One event can be a member of quite a lot selections and therefor it has to be delivered to every instance working on one of these selections, where the specific event is part of. The predictive batch scheduling approach introduces a batch of one ore more selections at one operator instance. Note that if there are more selections opened, than instances available, Round-Robin will also create some sort of batch. The difference is therefor, that the predictive batch scheduling will do this in common of its predictions and not due the coincidence that there are more selections than operator instances. The batching decreases the communication overhead, but in contrast the workload of one instance increases. The approach handles this trade-off with two predictions. The prediction of all events of the batch is one of the predictions. So the splitter predictively knows all events of the specific batch. Even if they already arrived or will arrive in future progress. The prediction of the processing latency is the other prediction in the scheduler. So the splitter knows the processing latency and can compare it with the time to stay all over in real-time computing. More details on the used batch scheduling approach can be found in [MTKR15].

2.4 Latency and Latency Budget

When we talked about the parallelization of a CEP network, we mostly come back to the argument of the need of real-time computing. In this context it is quite important to name the terms latency and latency budget.

The latency is like anywhere else a time delay and moreover in our circumstances the delay between the input of one event and the outgoing result at one working instance. Hereby the working instances are not only the interchangeable operator instances, but rather all instances working on the event streams. Thus the instances of one operator, the splitter and the merger. Furthermore we like to assume the latency of one parallelized operator node or rather the whole system. Thereby the latency is specified by the situational changeable critical path of the framework or rather the network. While the events are fluctuating by time, the parallelization degree and our batch scheduling will therefor be dynamically changed. The critical path in one node or rather in the whole system will as impact also change dynamically.

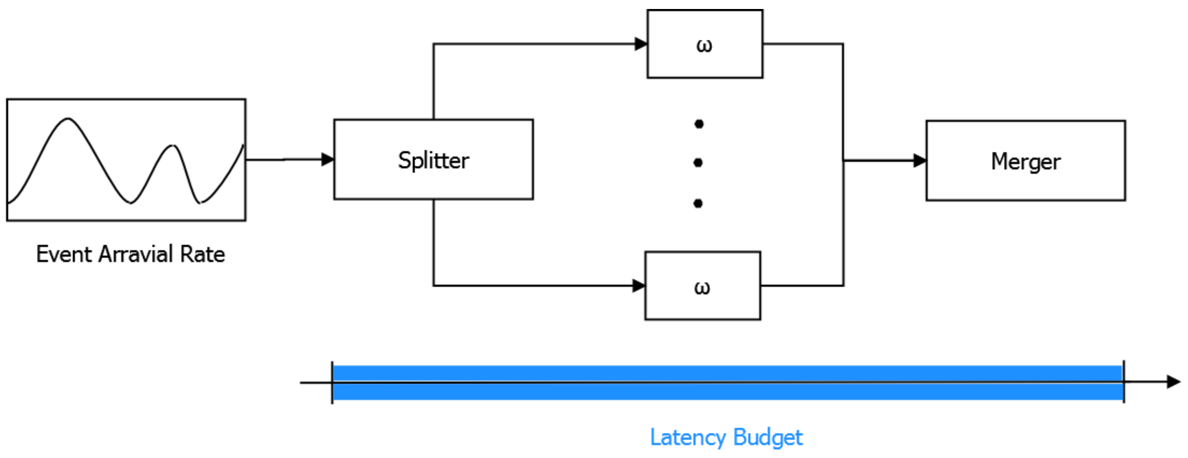


Figure 2.4: Graphical depiction of the latency budget of one CEP node.

To stay at practical work in real-time computing we need to compute in a certain time limit. We call this limit the latency budget. More precisely it exist quite a lot time limits in one system to stay finally in real-time. As to see in figure 2.4, the latency budget could also be the time limit for exact one operator node in the CEP network. Rather its possible to calculate latency budgets for every step in den process, when we know a limit for an superior level of abstraction. For instance, an application could define a latency budget for one event reaching the CEP Network until it arrives at the application. Therefor the CEP system can now calculate the budget dynamically at the process and try to stay in the latency budget. It can follow different strategies to do so, but finally it is only interesting that the CEP system does not need more time than it has got from the application by the latency budget. So finally we got some latency budget for one node in the CEP network as you can see in figure 2.4.

3 LoS issue

The central context of this chapter will be the definition of the *Latency-of-Splitting* (LoS) issue, which we want to solve in this work. To get a better understanding, why this issue is significant to contemplate about, we first introduce this chapter with a motivation. After this we will give a definition of the LoS issue. Finally we will close the chapter with a first analysis of the LoS issue. The analysis is there for concentrating on two points:

- identify the parameters, which we should observe to evaluate a smart result.
- lead to an first idea, how to solve the LoS issue.

3.1 Motivation

In the usage of CEP systems for handling complex and staggering situations in real-time, parallelization becomes momentous. One cause is the possible high load of different events generated by thousands of thousand sensors, which could generate an extraordinary overhead at one node. In the sequel of parallelization the dynamically changeable computation power is very important. On one hand applications should always be optimized as most as possible to realize low-energy computing and make it possible to get scheduled by an operating system without a consequence of a too high latency in doing a distinctive step. On the other hand applications should always stay in real-time computation for an optimal reaction flow. Sometimes this trade-off is hard to implement or rather hard to explore at a local optimum. As proved in preliminary work the framework PACE is an auspicious option to handle this parallelization trade-off at CEP systems.

PACE could be optimized by profiling the performance of the merger and the splitter. This profiles will lead to easily predict and dynamically change the parameters of parallelization in PACE and stay all time in real-time computing. Because if we know the latency of the splitter and the merger at a specific point of time, we also know our remaining latency budget for the operator instances. We can assume, that an increasing of the parallelization degree takes some time. In fact, we need to spend some time to get more computing power of the cloud in practice. If we assume, that such a startup of a new instance will take one minute, it would be nice to know, when we need more instances to handle the upcoming situation in real-time. So we want to know the point of time, when we need more instances than now to react fast enough. As one minute is in real-time computing a really long time, we want to recognize future requirements before the situation, where we need them, came up to process it. So when we know the latency profiles of the splitter and the merger, we can predict the future requirements. With this prediction the parallelization degree and the batch scheduling could be adjusted more perfectly for every upcoming scenario of the CEP progress. Important is, that we not only adjust them, you can infer that we also prepare the whole framework for an predicted upcoming situation. This leads to a CEP system, which uses only as much computing power of an CPU or cloud computing architecture as needed, but even enough to stay in the given latency budget. Moreover the applications, who uses the CEP system, could define a personal latency budget for all needed informations. Thus they can ensure a real-time processing over all events reaching the application. The applications could define the latency budget, because they know the future steps in their process for designated events. Thereby they can contemplate about the differences in the future progress and so define the latency budget for the CEP system, to stay after all in real-time computing.

The knowledge of the latencies of splitter and merger, can moreover be separated. First, the splitter and the merger differ in their work, hence in their latency. So for knowing the splitter's latency we have to think about other methods than for knowing the merger's latency. Moreover the latency of the splitter is influenced by two separated factors. These factors are the runtime of the splitting and of the scheduling. The splitting is the point, where the decisions upon PStart and PClose are made. Thereby the inter-arrival time of events and the evaluation of the predicates will probably influence the runtime of the splitting at most. Whereas the scheduling is the part, where the selections are scheduled by the splitter in means of the available operator instances. This runtime is probably mostly influenced by the underlying scheduling strategy. For instance, a Round-Robin scheduler will never take as much time to process as the predictive batch scheduling. All parts have their need to contemplate about them. Therefor we will concentrate us in this work on the LoS. Thus on the evaluation time of the predicates PStart and PClose.

3.2 Definition

The LoS issue is the problem, how to profile the performance and so the latency of the splitting at PACE to finally predict it. Therefor we want to understand, which factors influence the latency and where we can asses a reinforcement learning model to learn the function of the latency of a specific splitting. Thus the LoS issue can be defined as the problem, to know the latency of the splitting in a specific CEP node, before the situation is real. With this knowledge we can stay the whole processing time better than before in the real-time latency budget. Because we know the latency of the splitting, and even the minimal latency of the whole splitter. Hence we could react to all upcoming situations more perfectly.

To predict the latency at all times and so to handle situations before they even come up in real progress, we will now first analyze the LoS issue. More in detail we will concentrate on the influences of the predicates in means of the splitter's latency.

3.3 Analysis

To get a better understanding of the LoS issue and its influences, we will analyze some possible inputs. Thus we are concentrating us on the predicates PStart and PClose of the splitter. Thereby we set as scheduling strategy the most easiest version of our knowledge for now, the Round-Robin scheduling as introduced in subsection 2.3.3.

To structure this section, it is split in two parts. First we will describe the setup of our execution of PACE and our measurement of the splitter's latency. Second we will examine the results of our execution to analyze the factors of the LoS issue.

3.3.1 Setup

The technical basis for this execution will usually be a simple PC. It is equipped with 3x2 GB DDR3 RAM, an AMD Radeon R9 290x and a Intel(R) Core(TM) i7 930 CPU. The motherboard is an ASUS Rampage II Extreme and the operating system is Windows 7 Ultimate x64. In this context 'usually' means, that the splitter will always run on this PC, but the other objects, as the rest of the framework, will may run on a second PC. We call this mostly used PC the main PC for the analysis. As one scenario of the analysis is too much load for the main PC in cause of a high workload, we may use another PC for specific parts of the framework. If we do that we always will take the second PC, which is a notebook. This notebook is a Lenovo ThinkPad Twist N3C28GE 334728G (Edge S230u). The technical equipment is there 8 GB RAM, a Intel Core i7-3517U and a Intel HD Graphics 4000 (IGP). The operating system is Windows 8.1 x64.

Further on we will always execute only one operator or rather node of a CEP network. Therefor we will take the PACE setup for this node. Note that we limit PACE for concentrating on the predicates PStart and PClose at the splitting and just use Round-Robin as scheduling. Further on we assume just two operator instances and one source will deliver upcoming events to the splitter. The splitter will then decide as normal the predicates PStart and PClose and schedule by means of the Round-Robing scheduling strategy. In addition we disable any dynamic changing of the parallelization degree.

We want to know the splitter's latency as result. Thus we measure this latency at the specific work of one splitter. The latency of an event at the splitter can be seen as the whole processing part of the splitter. Thereby, as mentioned in subsection 2.3.2, the splitter first evaluate PStart. Then he schedules the probably new selection and will finally decide the truth of PClose for the specific event and each at this time opened selection. So we just log the timestamp in front and beyond each execution of these splitting decisions. Thus we know the splitters latency of this specific event. As we just use Round-Robin, we assume to neglect the latency of the scheduling in cause of this simple and never changing scheduling. The tests will just differ in their predicator operators and their parameters. Note again, that PStart is just evaluated once for each event, whereas PClose is evaluated for each pair of current opened selections and the specific event.

In cause of analyzing the influences of different parameters, we will investigate on three different predicate operators. Therefor we will take the *tuple-based window*, the *time-sliding window* and the *traffic scenario*, which was mentioned in section 2.1.
The tuple-based window (tu) has two parameters for the exactly window or rather selection size. The first one adjust the selection size in means of, how many events are in one selection. The second parameter defines the sliding in means of, about the number of next events, when the next selection starts. In the case of tu we will use only the main PC to simulate the whole node plus the source. Moreover we can ignore any content of the event and work with just simplest events. We can do so, because the opened selections are defined at the splitter in the parameters and we do not care for the real result in our context. In cause of this simplest events, we just use a dummy operator for the instances. So at the time as one operator instance handles a new received event, the instance is finished and just send an ACK to the splitter.
Very similar to tu is the time-sliding window (ti). It is also adjusted by two parameters, where the first one gives the size of the selections and the second one the sliding size. The difference to tu is that it is defined by the time and not by the number of events. Hence the selection size is given in milliseconds and the sliding size is there also a given time. As ti is similar to tu, we again use there just the main PC and simplest events without any content. As a consequence we also just use a dummy operator for the operator instances.
As last predicate operator we will investigate the traffic scenario with its two positions pos_1 and pos_2. As revision, when one car reaches pos_1 an selection will be started with the related event and when it arrives at pos_2 the selection of the car will be closed with the related event. As a next step it should be concentrated on if one car overtook another one. That means if the selection B starts after the selection A, but B closes before A closes, the car of selection B has overtaken the car of selection A. Thus further steps has to go on. Note, that in our analysis this further step is not important. Since we focus on the latency of the splitter in cause of the opening and closing of one selection. In addition it is to note, that only the traffic operator will use the second PC in cause of high load operator instances. The measurement of the latency will, as mentioned before, always run on the main PC, but for the traffic scenario, all other instances of the node and the source, will be running at the second PC. We need this outsourcing of processing at the traffic scenario, in cause of the computational work of the instances at this scenario. At the traffic scenario the events are of type pos_1 and pos_2 and have the content of an specific car ID. Therefor the instances has to calculate the outgoing events of how long they stay at the road and when they arrived, so the workload is much higher than at the tu or at the ti.

overlaps	selection size in events	inter-arrival time in ms
1000	1000	1
5000	5000	1
10000	10000	1
15000	15000	1
20000	20000	5

Table 3.1: Table of the different analysis runs with tu.

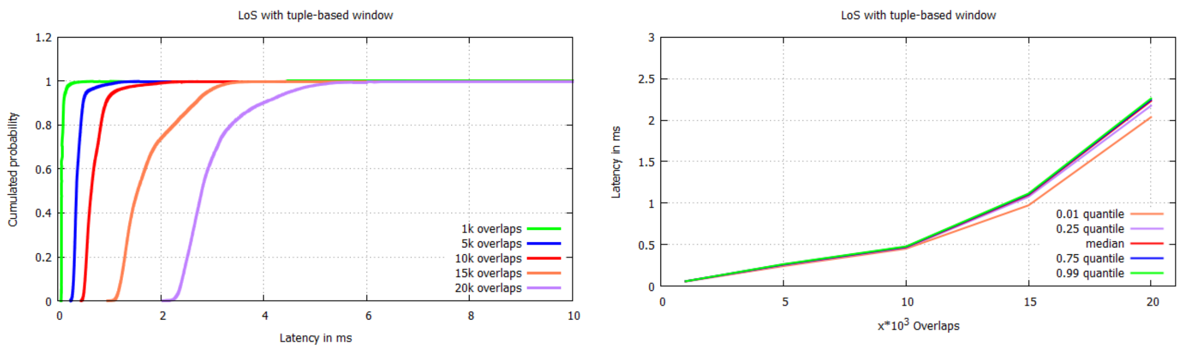
Always remember that the splitter has two different parts, which affect his latency. One hand the evaluation of the predicates will take some time and on the other one the scheduling strategy will also affect the latency. In this work we want to concentrate on prediction of the latency in means of the predicates, but for the predicting the whole latency of the splitter with a small error, the scheduling influences on the latency of the splitter are also necessary to investigate.

3.3.2 Execution Results

The execution can be split in three different parts by using the different predicate operators. We will follow thereby to first present the results of the tu, then the ti and finally the traffic scenario.

The tu is adjusted for how many events are in one selection and for what is the shifting number of events. So for tu we use five different test runs. These runs are all executed with the shift parameter of one event. The size of one selection differs by the parameters of one thousand, five thousand, 10 thousand, 15 thousand and 20 thousand. Note that in cause of the simple events, all events will open one selection and can close a selection. If they fill one selection as the last event so that the size of one selection is reached, the event closes this selection hereby. Thus when the sequence number of the event is bigger than the specified size of one selection, every following event will open and close one selection. This holds because the shifting parameter is equal to 1. All events are coming in an inter-arrival time of 1 ms, except the 20 thousand measurement. This case is special because the test run needs at the overlap of 20.000 selections at a time with the sizes of 1 to 20.000 to much time in processing the events with an inter-arrival time of 1 ms. So for to hold a most similar performance as at the other four runs, we just rise the inter-arrival time for the overlap of 20.000 a bit to 5 ms. Finally, we get five different numbers of overlaps and the runs as to see in table 3.1, whereby each new event starts a new selection.

As to see in figure 3.1a, a different count of overlaps generates another distribution of the latency. This is a bit what we expected due the fact, that PClose is evaluated for each pair of the



(a) Cumulative probabilities of the latencies at different overlaps. **(b)** Quantiles of the latency at different overlaps.

Figure 3.1: Plots of the LoS measuring with tu.

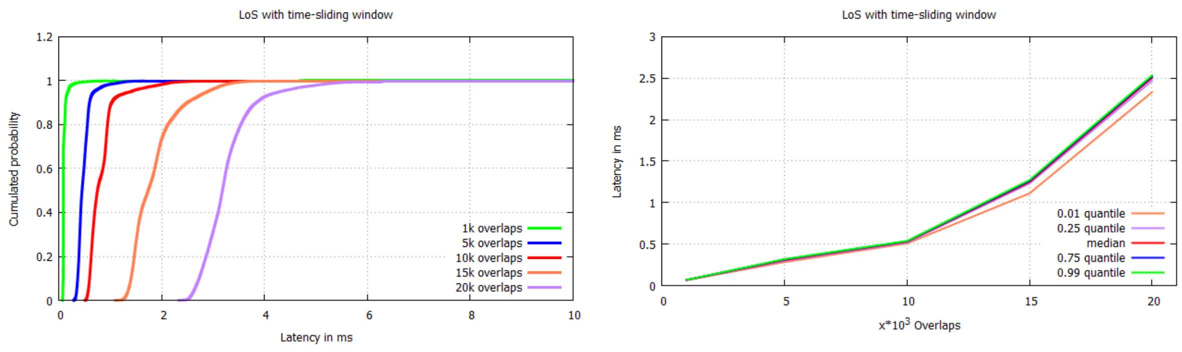
specific event and current opened selections. So at a higher overlap PClose will of course need more time to evaluate. Moreover the cumulative probability of the latency shows a probably close link of the latency to the count of overlaps. Thus the overlap seems to be mostly the building factor of the splitting latency at the tu. This thesis is underlined by the second plot of the LoS measuring with a tu. Figure 3.1b shows an plot, with the different quantiles of all measured latencies at a specific number of overlaps. It is to see, that mostly all quantiles are really near together at their values. Only the lower quantiles seems to depart a bit from the other ones at higher overlaps. It is to recognize that still all quantiles are increasing with increasing number of overlaps.

Next the ti is adjusted by the parameters of how long one selection is opened and on how much milliseconds the ti shifts until the next selection will be opened. We will set the parameters, such that 1, 5, 10, 15 and 20 thousands overlaps will be generated. Since we did the same at the test of the splitting's latency with tu. So from 1 to 15 thousand overlaps the inter-arrival time is 1 ms and the shift parameter is also 1 ms. The size of the selections infer to be 1 to 15 thousand ms. The overlapping of 20 thousand selections is again a special case and need a inter-arrival time of 5 ms. Hence the selection shift parameter is also set to 5 ms and the selection size is set to 99 995 ms to generate an overlap of 20 thousand. So finally we got again five runs with a different number of overlaps, which are summarized in table 3.2. Note that each event again starts a new selection as at tu.

The plots of the test runs in figure 3.2 are nearly the same as with the tu. Thus we can see in figure 3.2a again that the overlap seems to be the most limiting factor in cause of increasing latencies by the same cumulative probability and different counts of overlaps. Moreover the overlaps also seems to be the mostly influencing factor of the latency of the splitting in mention of figure 3.2b. Thereby it is to recognize as by the quantiles of the splitting's latency with tu, that all quantiles are rising with respect to an increasing of the number of actual overlapping selections.

overlaps	selection size in ms	inter-arrival time in ms
1000	1000	1
5000	5000	1
10000	10000	1
15000	15000	1
20000	99995	5

Table 3.2: Table of the different analysis runs with ti.



(a) Cumulative probabilities of the latencies at different overlaps.

(b) Quantiles of the latency at different overlaps.

Figure 3.2: Plots of the LoS measuring with ti.

So we assume further on, that the number of overlaps is the most commonly factor to predict the latency of the splitting. tu and ti are quite similar predicates and both are quite simple and have low computational burden, where the type of the event is irrelevant for processing at the splitter. So in fact, we need one more distinguish predicate operator to get a knowing of the general influence of the overlaps. Therefor we have our last predicate operator, the traffic scenario.

As the tu and ti were quite simple predicates, the traffic scenario is bit more complex in cause of differencing the event types. Events of pos_1 will open one selection and events of pos_2 will close one. Moreover the content plays a role in this scenario, because one car, which opened a selection at pos_1, will always close again one selection at pos_2. Therefore it is necessary to identify the car, which triggered the event of the specific position and so the content becomes important for the splitter. For the test runs with the traffic scenario, we need to remember, that the second PC is involved, but the measurements were still taken on the main PC. So we did run the scenario also 5 times with different parameters as we did before with the other predicate operators. The difference hereby was that we could not generate a specified and constant number of overlaps. At the tu and ti we can adjust the operators more

number of cars	inter-arrival time in ms (exponential distributed)
20000	5
20000	1
20000	0.1
50000	10
50000	5

Table 3.3: Table of the different analysis runs with the traffic scenario.

in detail at the splitter's parameters. This time with the traffic scenario, the predicate operator is always defined the same with pos_1 and pos_2 and do not allow any adjusting parameters. So we generate the difference at the scenarios. We assume a lane of 1 km distance and that all vehicles could drive 80km/h to 120km/h. Moreover the inter-arrival time of the different cars is adjusted by a parameter of the underlying exponential distribution, so that it is possible that the inter-arrival time is smaller, but nearly not greater. The other parameters for our generator are the number of cars in the scenario and the supremal time of how long a car needs to pass the monitored lane. For generating much more different overlaps and do not have to adjust the supremal time one vehicle needs to pass the lane, we only adjusted the inter-arrival time. Thereby the inter-arrival time is getting much higher than in real scenarios. The point is, that it is the same for generating the overlaps, if we just need adjust the lane length and so the time one car supremal needs for this distance or whether we adjust the inter-arrival time. So for easily distinguish the different scenarios we adjust the inter-arrival time and of course the number of passing vehicles. So we assume scenarios as to see in table 3.3 for the test runs.

The traffic scenario has no cumulative probabilities plot in cause of the not getting a constant number of overlaps. Our assumption of the mostly underlying factor is still provided, because the latency increases even at the traffic scenario with a higher number of overlaps. As to see in figure 3.3, the quantiles of each overlap until 20 thousand are near neighbors. Thus there is no big variance of the latencies at a specific number of overlapping selections. There are some peeks, but we assume that they are the consequence of the scheduling of the running system.

So finally we assume at a result of the analysis that the number of overlapping selections is the decisive factor of influence on the given latency for the splitting part at the splitter. With this assumption we will build an approach in chapter 5, to solve the LoS issue.

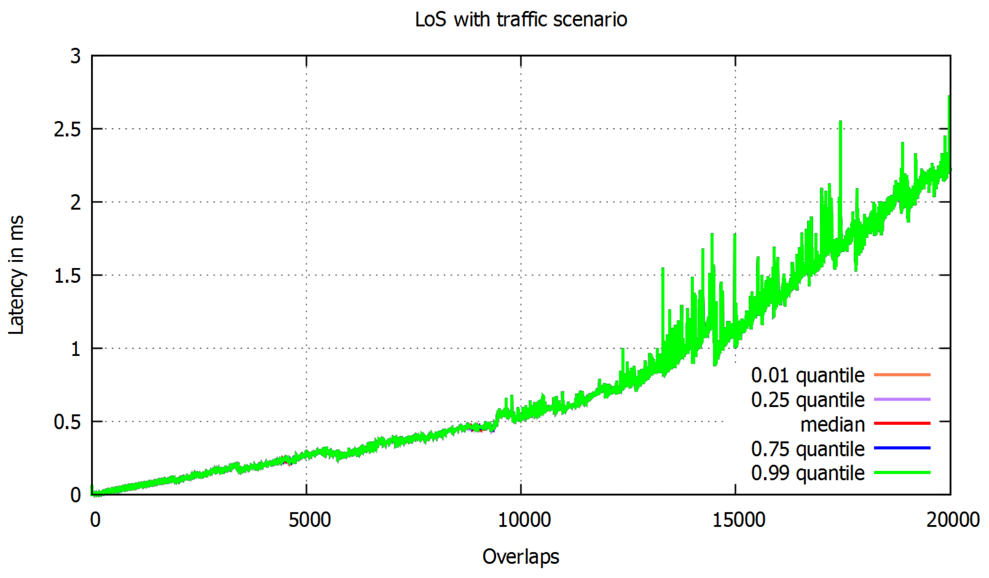


Figure 3.3: Plot of LoS quantiles with traffic scenario.

4 Prediction Techniques

The LoS issue is as mentioned, the issue about the predictive knowledge of the latency of the splitting in future scenarios. As we will invent a solving approach for the LoS issue in the following chapter 5, we need therefor to mention about some prediction techniques. We can separate in two different types of prediction techniques, which we need further on. So on one hand we will need some regression learning to predict the latency in means of the overlap. Therefore regression learning can give us a predictive function with a quite low error. Further on we need some time series prediction, to predict the overlaps in the future, whereby we do not have any real context to make a nice prediction except the last measurements. Time series prediction is therefor the answer, because it predicts a variable in means of their history.

This chapter is further on split in two sections. At first we will talk about regression learning models and beyond about methods of time series prediction.

4.1 Regression Learning Models

As we want to solve the LoS issue by learning a latency function with a regression model, we consider in this section about different regression models. Regression learning works best if the right model is used, because each model interpolates a bit different and so extrapolates different. So the model, which fits best to a specific situation is strongly related to the situation itself. If the model is not chosen rightly, problems at the results of interpolation can occur due over- and underfitting. Whereby overfitting means that the degree of a function is too high in the range of some points, so some interpolated points have a too high error in means of the real original function. Accordingly underfitting means that the degree of a function is too small, whereas again some interpolated points have a too high error ins means of the real original function. A nice example for over- and underfitting can be seen at figure 4.1. Thereby the black points are the data points, the red line fits best for those three and the green and blue line under- or rather overfits the data points. Outside of the range of the input data, the regression learning model has to extrapolate if it is asked for such a point. The results of a extrapolation are strictly related to the calculated regression function. Thus they can be good if the raise of the data is predicted well in the range of the data points. So the goodness of the extrapolation is related to the available data, the distance of the asked point to the available data, the model strategy and to the specific scenario. So for getting a nice evaluation later on, we take two different models to compare the results of them. Note that we assume both

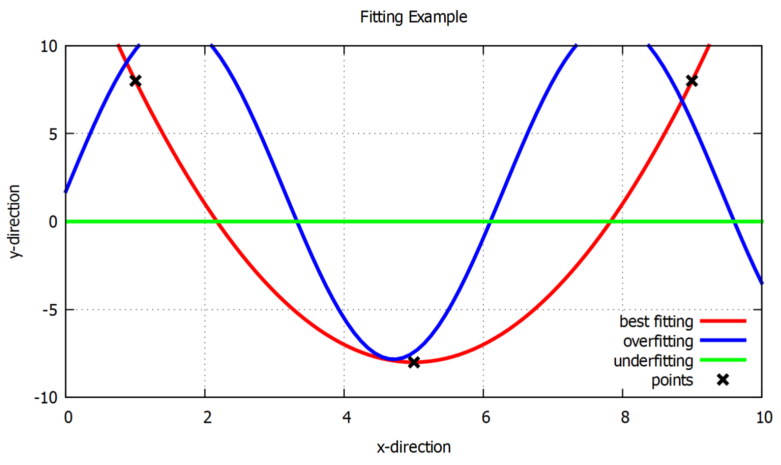


Figure 4.1: Examples for over- and underfitting.

models can have nice results in our evaluation scenario and will see more at the evaluation in chapter 6. The probably most easiest versions of all regression learning methods are the so called linear regressions. Thus it does not take high computational costs to calculate such a linear model. In cause of this, one of the two methods we use for the comparison is a linear regression. A really nice idea of regression learning are the so called Multivariate Adaptive Regression Splines (MARS) by Friedman [Fri91]. It is auspicious for us because it never needs some sort of parameters to reach mostly all function points in a model and is still strongly fitting to the input data.

So at first we introduce the two regression models in this section. Beyond we compare their advantages and disadvantages to know their specified differences.

4.1.1 Linear Regression

We use an ordinary least squares (OLS) linear regression model to take in some easy and simple model. Thereby it not only takes the part of comparison for MARS. The idea of taking a quite simple method for building the regression model is based on the possibility, that an easier structure may not deliver more inferior results as the complexer version. The simplest recommend version of linear regression is the OLS method. In doing so, the OLS model works on the ordinary least square sums of the x and y values of the entranced data. When the square sums are least, the error term of the prediction function to the real values are minimal in each point. In other words, the linear regression with OLS minimize the difference between all points and the model function by minimizing the square sums of all input data. So you can see that the linear regression is in a optimization problem by minimizing all residues of the input data to the regression function. To get a better understanding of linear regression, we will also describe it in a more mathematical way. So for instance we have n data points (x_i, y_i), whereby $1 \leq i \leq n$. Linear Regression tries to solve the regression problem by finding a polynomial function with degree 1 of the form:

$$f(x) = \alpha_0 + \alpha_1 x$$

The residue r_i between the function and the data point (x_i, y_i) is then defined as:

$$r_i = \alpha_0 + \alpha_1 x_i - y_i$$

Linear regression now tries to find the coefficients α_0 and α_1 such that the sum of the squared errors is minimal.

$$\min_{\alpha_0,\alpha_1} \sum_{i=1}^{n} r_i^2$$

We can solve this easily by assuming the sum as a function of the variables α_0 and α_1 and take the input data as numerical constants. So we have to find the minimum variables for:

$$n \cdot \alpha_0 + (\sum_{i=1}^{n} x_i)\alpha_1 = \sum_{i=1}^{n} y_i$$

When we then take the derivation of this function and try to find the root, we have the equation:

$$(\sum_{i=1}^{n} x_i)\alpha_0 + (\sum_{i=1}^{n} x_i^2)\alpha_1 = \sum_{i=1}^{n} x_i y_i$$

So we can calculate the solving as:

$$\alpha_1 = \frac{(\sum_{i=1}^{n} x_i y_i) - n \cdot \bar{x}\bar{y}}{(\sum_{i=1}^{n} x_i^2) - n \cdot (\bar{x})^2} \quad \text{and} \quad \alpha_0 = \bar{y} - \alpha_1 \bar{x}$$

Whereby $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the arithmetic mean of the x-values or rather the y-values.

As an example, the figure 4.2 shows in blue some points in a 2-dimensional room. The red line clarifies the predicted linear function, which fittest the most as possible to all points in the same instance. We will use the class called *simple regression* by apache commons for the implementation. All relevant infos about the library of apache commons or the library itself can be found at [apa].

4.1.2 Multivariate Adaptive Regression Splines (MARS)

As mentioned before the MARS idea was invented by J.H. Friedman in 1991 in [Fri91]. The differences between MARS and the linear regression will be discussed in the next section

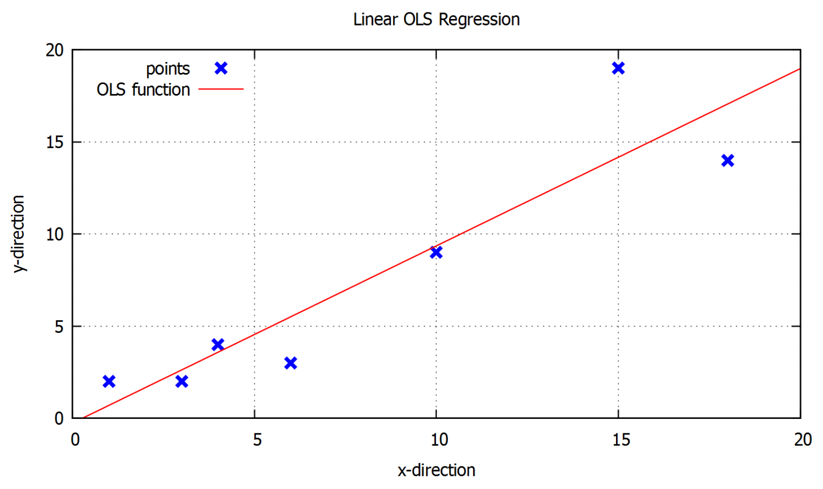


Figure 4.2: Linear Regression using OLS.

(4.1.3). So for now we focus on the approach of MARS and refer hereby strict and directly to [Fri91] and [HTF09].

The only user specified input is the maximum size of the basis functions set for the model. As the name terms, the basis functions are splines and to be more precise the basis functions are linear splines. The basis functions are defined as $(x-t)_+$ and $(t-x)_+$. The "+" means that these functions depict only the positive part and are otherwise zero. So more in detail, they are defined as:

$$(x-t)_+ = \begin{cases} x-t & \text{, if } x > t \\ 0 & \text{, otherwise} \end{cases} \qquad \text{and} \qquad (t-x)_+ = \begin{cases} t-x & \text{, if } x < t \\ 0 & \text{, otherwise} \end{cases}$$

These basis functions are always occur in the mentioned pairs with the same knot t. One example of such a pair of MARS's basis functions is to see in figure 4.3.

The different knots are the distinct input variables x_{ij} for the input X_j. Note that it is possible to learn a model with different input dimensions for predicting one ore more other value-dimensions. For instance, we can input different measurements of the pulse and the moving of a specific person and then predict somehow the sportiness of this person in general. So MARS distinguish these different inputs with the term X_j. As a next step the candidates for being an basis function are collected in the set $\mathcal{C}$, where:

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1j}, x_{2j}, \ldots, x_{Nj}\} \text{ and } j=1,2,\ldots,p}.$$

Note that although each basis function depends on a single input X_j, it is defined as a function over the entire input space $\mathbb{R}^p$.

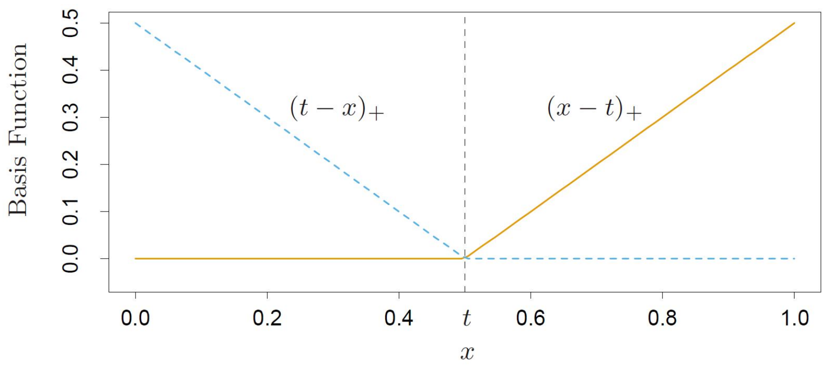


Figure 4.3: A pair of basis functions in MARS with $t = 0.5$. [HTF09]

The idea of MARS is to perform a stepwise linear regression. Thereby it not uses the original inputs, but the functions of the set $\mathcal{C}$ and their products. The resulting model gets the form

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X)$$

where each $h_m(X)$ is a function of the set $\mathcal{C}$ or an product of two or more of them.

Next MARS differences between a forward and a backward algorithm. The forward algorithm is the algorithm, where the basis functions are chosen and the model is established. Thereby the forward algorithm overfits the input data, in cause of two new basis functions or rather one, before called, pair of functions in each step. Hence the backward algorithm is for cleaning up the model and desirably deleting this overfitting.

The forward algorithm start with the constant function $h_0(x) = 1$ in the set $\mathcal{M}$, which contains all basis functions of the resulting model. In every step the term of the form

$$\hat{\beta}_{M+1} h_\ell(X) \cdot (X_j - t)_+ + \hat{\beta}_{M+2} h_\ell(X) \cdot (t - X_j)_+ \ , \ h_\ell(X) \in \mathcal{M}$$

is added to $\mathcal{M}$, which fittest the most. As to see, each part of the new basis function pair is multiplied with one of the h_m in $\mathcal{M}$. The factors $\hat{\beta}_{M+1}$ and $\hat{\beta}_{M+2}$ are minimized trough least squares as in linear regression. The term, which decreases mostly the training error is selected as the best and added to the set $\mathcal{M}$. This procedure is working until the maximum size of $\mathcal{M}$ is reached.

The backward algorithm then has to delete some functions again to get a smoother final result and not such a overfitting as the forward algorithm produces. In every step the term will be removed, which removal causes the smallest increase in residual squared error. So finally the estimated best model $\hat{f}_\lambda$ is produced, where $\hat{f}_\lambda$ has the size of λ, which is the number of terms. Friedman proposes the general cross-validation to estimate the optimal value of λ.

The approach of Greg Amis, which we use in later on to evaluate the final model of our approach, differs there a bit. This approach stops deleting basis functions, when the squared error stops decreasing and when the number of bases is less than or equal to the maximum number of bases. The approach of Greg Amis is to find at [Ami06].

4.1.3 Comparison of Linear Regression and MARS

As you may recognized before, MARS takes in some parts of linear regression and exchanges the model strategy by the basis functions. So there are some differences, but rather you can say, that MARS is somehow an upgrade of linear regression. For instance, we have a cloud of points in 2D like in figure 4.2, but know with a corner, so that one linear function can only predict the half of the cloud with a low error. Therefor MARS will find this corner and take it as a knot. Then it can establish two different linear functions to predict both halves with quite a low error. So on one hand MARS could handle such a situation quite nice, but on the other hand it is much more complex in his work and so have a higher runtime than linear regression.

This makes clear, that we should take both, linear regression and MARS, to evaluate our approach, because two different possibilities could eventuate. The first one is, that linear regression predicts not quite well due the fact of a only piecewise linear slope and MARS fits this error by its complexity or rather its stepwise basis functions. The second possibility is, that MARS needs much more computational power and linear regression suits still well in our approach. Both possibility have their right to exist, but hopefully one will eventuate strictly at the evaluation.

4.2 Time Series Prediction

As mentioned before, we will need to predict a variable without any other parameters, who influence this value, than the history of measurements of this variable. More detailed, we want to predict the count of overlaps at the time, where the next selection closes. Thereby we need to predict the inter-arrival time of new selections and the lifetime of selections by only considering their own measured history. More details on this are to find in the following chapter 5, where we will present our solving approach. Therefor time series prediction is the answer, because we can measure in distinct timestamps the actual value of the variable. With this measurement we are generating a history and therefor the possibility to use time series prediction methods. For evaluating our approach we will take in two different time series prediction methods. As in the last section we will introduce those two methods in two different sections.

For introduction in time series prediction, it is to note, that every time series prediction is based of measurements of the same variable. So it is not like in regression learning, where an inference on a certain variable is made in regard to other variables. Instead times series prediction measures the different values of a variable and trying to make a prediction for the next value of the variable with a small error.

The table II in [HHKA14] brings us to the two methods *Moving Average (MA)* and *Simple Exponential Smoothing (SES)*. The SES is again some sort of upgrading the MA, like MARS upgraded linear regression. To implement the time series prediction models, we will use the OpenForecast library from Steven Gould. Any further information on the Code or OpenForecast can be found at [Gou].

4.2.1 Moving Average (MA)

Moving Average calculates at each new incoming value measurement of the monitored and to be predicted variable the arithmetical mean of the last k measured values. It is possible to specify k and so to fit the predicated result for the next value as fitted as the user wants. A disadvantage is that much saved measurements for calculating the prediction infer a higher workload and more memory space, but a too small count of measurements is not fault tolerant to false measurements or outliers.

To give a small example, we could think about an array $A = \{1; 3; 6; 4; 9; 2; 7; 3; 5; 8\}$ of measurements of the variable t with its ten inputs, whereby the maximum size of last known measurements shall be 10. If we see this array as a stream of the incoming measurements by time, where the first index is the oldest measurement and the biggest index is the newest one, we can establish a MA. So as we have ten measurements and each measurement gets the same average weight, we just use as weight $w = \frac{1}{sizeOf(A)} = \frac{1}{10}$. Further on if we call a prediction at this time, our time series prediction model MA will answer us with the result r:

$$r = \sum_{i=1}^{sizeOf(A)} A[i] \cdot w = w \cdot \sum_{i=1}^{sizeOf(A)} A[i] = \frac{48}{10} = 4.8$$

So we expect as the next measurement a value of 4 by just cutting at the dot. If a new measurement is taken, the array A will delete the oldest measurement at index zero, shifts all indexes by -1 and will then insert the new measurement at the biggest possible index. There by the average of the measurements moves by time and so it is called MA.

4.2.2 Simple Exponential Smoothing (SES)

Simple Exponential Smoothing also calculates the arithmetic mean of the last k measured values, but is distinguished from MA by giving the measurements different weights. Thereby the different weights are exponential distributed and the distribution is smoothed by an user given factor. Thus the older measurements can be fewer weighted than the newer ones, so that the newer ones are more important for the next value than the older ones.

As an example we again think about the array $A = \{1; 3; 6; 4; 9; 2; 7; 3; 5; 8\}$ and the maximum size of last known measurements is again 10. So if we then take a smoothing factor of 0.75, we equip the newer measurements with a bit higher weight than the older ones. If we choose it 0.5 it would be nearly the same as MA, if we would use 0.0 as smoothing factor the newer ones would not even influence the prediction and if we would use 1.0 we older ones would not even influence the prediction. However, if in our example here, the array $w = \frac{1}{18}; \frac{1}{18}; \frac{1}{17}; \frac{1}{15}; \frac{1}{10}; \frac{1}{10}; \frac{1}{5}; \frac{1}{3}; \frac{1}{2}; \frac{1}{2}$ for instance represents the weights, whereby the weight $A[i]$ is at $w[i]$. So the prediction's result r is here:

$$r = \sum_{i=1}^{sizeOf(A)} A[i] \cdot w[i] = \frac{1}{18} + \frac{3}{18} + \frac{6}{17} + \frac{4}{15} + \frac{9}{10} + \frac{2}{10} + \frac{7}{5} + \frac{3}{3} + \frac{5}{2} + \frac{8}{2} = 10.842$$

So if we cut again at the dot, the predicted value will be 10. The next measurement will inserted like in MA to hold the maximum size of last known measurements.

4.2.3 Comparison of MA and SES

So you can see, that the last trend of how the measurements are changing influencing the prediction of SES, whereas at MA the trend is only influencing the prediction if it is also influencing the average mean. Distinct more differences are not really to see between those two different models. As a conclusion of this section, it is to say that the advantage of MA is the simplicity, but SES in turn can better recognize current trends of the predicted value. So we assume, in our context is only a advantage to take SES if and only if, the inter-arrival time of new selections or the lifetime of selections is changing during the scenario. Because if so, SES can provide ot predict on regard to this trends. Else MA should be the better choice in cause of its simplicity and its stable prediction if the values just differ in cause of some faults at the measurement, thus swing around a constant value.

5 Solving Approach

In this chapter we will describe our solving approach of the LoS issue, which we will evaluate in the next chapter. Therefore we will section this chapter in three parts. At first we will invent the main-algorithm to predict the latency of the splitting, thus solving the LoS issue. As we will see in the main-algorithm, we will need some environment to calculate our predictions. We will use the mentioned prediction models and will describe, how we use them in the second section. The last part of the chapter will be built by a description of the workflow of the whole approach at runtime.

5.1 Main-Algorithm

In the analysis of the LoS issue in section 3.3 we concluded a final assumption on the latency of the splitting. The overlaps of opened selections are the mainly influencing parameter of this latency. As a consequence of this assumption, we can introduce our main-algorithm of predicting the latency of the splitting as figured out in algorithm 5.1.

As we wanted to know, where exactly the latency is rising more, we concluded about the two different predicates PStart and PClose. So if we start a new selection in cause of a new event is opening, it is a quite low burden to calculate. Because we only have to allocate a new selection on our memory and moreover schedule it to one of the available instances. But the scheduling is not a part of the splitting. So for PStart there is mostly only the allocation of a new object of the selection class and the adding to our selection set. On the other side, the second predicate PClose is more expensive in computation. Because if we will go ahead with, we need to find the selections, which are closed by this event. Due this, we have to go through the whole set of selections and must find out, if the new arrived event at the splitter, will close this specific selection. So the closing of a selection needs distinctly more computation power than starting a new one. This fact and our assumption of the analysis leads us to our main-algorithm *getPredictionOfLoS()*, which can be found in algorithm 5.1.

So at first we want know the oldest still "living" selection. In our context a living selection is one, which is opened but not closed yet. We take the oldest selection in cause of the highest probability of closing as the next one. Of course, in a more complex scenario than the simple tuple-based or time-sliding window, like the traffic scenario or even more complex, this has not to be the truth. But still in a more complex scenario, the oldest selection should have the

Algorithmus 5.1 Algorithm for predicting the LoS.

```
 1: procedure GETPREDICTIONOFLOS()
 2:     slt ← getOldestSelection()
 3:     rolslt ← getRestOfLifetime(slt)
 4:     iat ← getSelectionsIATPrediction()          // iat is intended to mean inter-arrival time
 5:     #slt ← getCurrentCountOfSelections()
 6:     if rolslt $< 0$ then
 7:         rolslt ← $0$
 8:     end if
 9:     if iat < rolslt then
10:         #slt ← #slt + $\frac{\text{rolslt}}{\text{iat}}$
11:     end if
12:     result ← askForLatencyWithOverlaps(#slt)
13:     return Latency at "$currentTime$ + rolslt" will be around "result" milliseconds
14: end procedure
```

highest probability of closing before all other selections will close. And if we just think about the simpler cases like tuple-based or time-sliding windows, we can see, that the oldest selection will close as first with even a probability of 100 percent. So at all, it is to say, that with the point of needing more computational power at closing and picking up the selection with the highest probability closing next, we should be on the right way. Because it is more necessary to know the peeks of the latency, than other situations. Thus if we know the peeks, we also know when we have to order more instances or rather increasing our parallelization degree to stay in the given latency budget.

As a next step, we need to know the selection's rest of lifetime. This timespan is really important to know, because then we can define a specific timestamp, when the selection should close and when the latency should be as high as predicted. Moreover we need to know the timespan between the predicted end and just right now Because then we can evaluate the count of new selections, which will be open at the time of closing the oldest selection. We need to contemplate about this fact, because at the time of our prediction not all selections, which will be there at the closing of the next one, have to live at the moment of the prediction. So for predicting these new incoming selections, we need moreover the inter-arrival time of new selections. Because if we know the inter-arrival time, we can easily estimate the count of new selections, which will get opened until the oldest selection should close. Further on we can simply update our number of currently living selections by calling the size of the set of opened selections. Then we can add to the number of current living selections the predicted number of selections, which will be get opened in the rest of the lifetime of the oldest selection. This adding is exactly need to be done, when the inter-arrival time is smaller than the rest of the lifetime of the oldest selection. Finally we can ask with the number of the overlapping

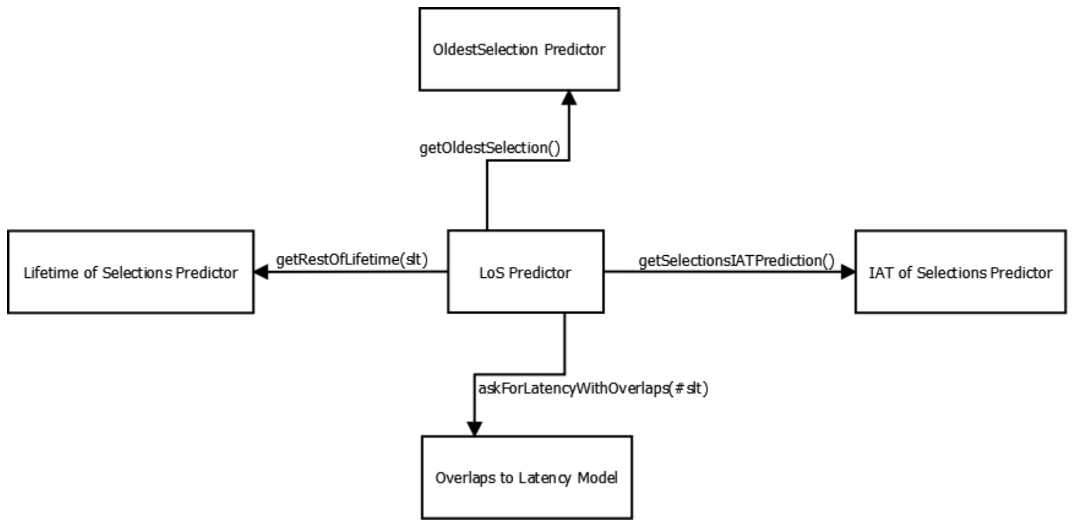

OldestSelection Predictor
getOldestSelection()
Lifetime of Selections Predictor
getRestOfLifetime(slt)
LoS Predictor
getSelectionsIATPrediction()
IAT of Selections Predictor
askForLatencyWithOverlaps(#slt)
Overlaps to Latency Model

Figure 5.1: The LoS predictor and its environment.

selections for the inferential latency and give out a nicely prediction, when this latency should be the case.

So in more details the main-algorithm starts at line 2 with saving the oldest selection in slt. The next steps are the saving of the current predicted rest of lifetime of the oldest selection in line 3, getting the current predicted inter-arrival time in line 4 and saving in line 5 the current number of living selection. Note that the if-statement in line 6 catches all errors, if the actual rest of life time of the oldest selection is lower than zero. Because if the rest of the lifetime is lower than zero, we have to assume that the selection should have closed in the past. Thereby the percentage that it closes just right now is a better assumption for our algorithm, than predicting situations in the past. To make this assumption we just set the predicted rest of lifetime to zero in line 7. However, after this catching, we need to add the predicted new selections, which will be there at the time of closing the oldest one. As mentioned before we evaluate if we have to do so in line 9 and if we have to, we add the number of incoming selections in line 10. A nice fact is, that the division of the rest of the lifetime and the inter-arrival time is by cutting it to a integer number exactly the number of selections, which will start until the next ends. The number of selections is predictively then the number of opened selections, when the oldest one closes. So we can get the prediction of the latency with a number of overlapping selections in line 12 and finally give a result in line 13. It is to see that the main-algorithm is quite simple and should not take a several time to get to a new result, if asked for one. So updating and releasing of any predictions should not influence the latency to much.

5.2 Environment

As we presented the main-algorithm, we need some more environment to implement the algorithm as simple as it is by itself. Because the different predictions and assumptions had to made or rather experienced continuously during runtime of the splitting. So finally we will take four more instances, which will work together with the LoS predictor. These different instances are mostly basing on the different prediction techniques we introduced in chapter 4. Moreover the overview of referencing the environment of the LoSPredictor to predict finally the latency of the splitting and so solving the LoS issue can be seen at figure 5.1.

The *OldestSelection* predictor is as the main-algorithm quite simple and does not need any of the mentioned prediction techniques to find the oldest selection. So may you can say it is not really a predictor, thus it is more a searcher. It is just linked with the set of living selections known by the splitter and always saves the oldest living selection to deliver it to the LoS predictor. The Splitter calls an update of this "predictor", whenever he closes a living selection.

The *Lifetime of the Selections* predictor and the *IAT of Selections* predictor are both based on a time series prediction model. So all predictions are based on the last k observations, and are either made with an moving average model or an simple exponential smoothing model. Whereby the factor k can be defined at the startup of the splitter and is so definable for the query of the whole system. This factor should be defined to establish a well-suitable model on each specific scenario. If the simple exponential smoothing is taken, we assume a smoothing factor of $\frac{2}{3}$, to make the latest observations a bit more weighted in the model than the older ones. But the older ones should not be nearly unimportant for the model, because if they are, the model size defined by k is just a bit too big.
So to predict the rest of the lifetime of the oldest selection, we just need to substitute the actual lived timespan of the oldest living selection from the predicted selections' lifetime. And there plays the prediction of the lifetime its role. Whenever a selection is closed, we can calculate its lifetime, thus have a new observation for our *Lifetime of the Selections* predictor. To predict the actual inter-arrival time of selections we do not have to do anything with the prediction of the *IAT of Selections* predictor. We refresh this model by simply determine a measurement of an inter-arrival time, when a new selection is opened by the splitter, and take it as a new observation for the predictor. Note again that both, the *Lifetime of the Selections* predictor and the *IAT of Selections* predictor, will use always use the last k observations to answer a prediction request.

The last to be named instance is the *Overlaps to Latency Model*. To evaluate the final prediction of the latency we take a regression learning model. Thereby it can be chosen between the linear regression and MARS. While the linear regression is easy to build and so easy to update, MARS needs there obviously more time. So while we build MARS just one time during runtime and then predict with this model, we proceed in the case of linear regression a bit different. We are building the model with the first few observations and will then moreover update it continuously. The update process of linear regression is really simple, thus we can update it with new observations during runtime and be therefore more precise and dynamic than without updating it. All observations are just the same we took as measurements for the analysis with the linked number of overlapping selections. So every single observation is a point in a two-dimensional field, where one dimension is the number of overlapping selections and the other one is the measured latency of the splitting. In the case of MARS we need to define a point from when we going to use the observations for the model and a point where we will build the model and future observations will not influence the model. With these two defined points we can also conjointly define the number of observations we take in to build the regression learning model. For setting up we do not define these points by time, since we can just define the number of first observations, we do not want to take, and as next the size of the MARS model, which means how much observations should influence the model. Note that, also in the case of linear regression the parameter of the number of first observations we do not want to take plays a role. Because in case of setting up the whole framework PACE, we can not be sure that all measurements, are exactly true, because may some of them will need more time than they would need it later on. This fact can appear due a too small space of computing power at the beginning of the process. We can be sure that the operating system will force enough computing power after a several steps. Therefore we also provide to ignore some first measurements at linear regression. Bad measurements at the beginning of the model, can influence the model really badly, in cause of giving the function a negative slope due highly differences between the measurements. When the future observations are quite near to themselves, the modeled function may stay with a negative slope. A negative slope is quite bad, because on one hand the analysis shows us, that this is not the truth and on the other hand predictions of high values may fail and give out negative predicted values. That a negative predicted value can not exist in our case should be clear, in cause of the fact that negative latencies do not make any sense.

5.3 Workflow

To get a better understanding, how our solving approach works during runtime, we focus us in this section on the workflow. This contemplation is quite important, because then it is hopefully easier to understand, how the results in the evaluation emerge.

So at first there is to say, that we gain some more concurrent threads at the splitter for our workflow. Because we do not want make the splitter busy with our prediction calculation. So we will start four new threads for each of our environment instances, two for some connectors of the time series models, which we will explain later on more in detail, and one for calculating the algorithm 5.1, thus being the LoS predictor. These seven threads are working together with the splitter and will finally result predictions of the LoS. Let us start at the splitter and go step by step deeper in hierarchy of our approach.

The splitter computes as usual the events and decides the two predicates PStart and PClose and schedules new selections with the underlying scheduling strategy. If the prediction of LoS is activated at startup, which can be done by a boolean parameter of the splitter, the splitter has to activate the update procedures of all 4 environment predictors at the right step of its process. So if a new event arrives and starts a new selection, the splitter has to some more work. The last inter-arrival time of new selections can be calculated by simply saving the last timestamp of opening a new selection and then calculating the span of time, how long it took since the last one was opened. This new measurement of the inter-arrival time is then delivered to the *IAT of Selections* predictor by calling an update with the new measurement as parameter. The timestamp of the new selection's opening has also then to be saved at the specific selection, to provide a calculation of the lifetime later on.
The splitter goes on again as without LoS prediction until it evaluates PClose. If one selection is closed by an event, the splitter has again to do some short and simple extra work. Therefore it forces an update of the *Oldest Selection* predictor, whereby no parameter is needed. Next the lifetime of the to be closed selection is calculated by the saved startpoint and the current time. This calculated lifetime will be delivered as a parameter of the update call at the *Lifetime of Selections* predictor.
At last it is to mention, when the update of the *Overlaps to Latency* predictor is updated. We therefore just observe the latency of the splitting as in the analysis in section 3.3 by simply measure the time before we handle an event and after we did it. We can calculate then again with these two timestamps the latency of the splitting. Thus after the splitter done is work of one event it calculates the latency of splitting and deliver it as a pair with the current count of overlaps to the *Overlaps to Latency* predictor. This delivering is the call of an update at the *Overlaps to Latency* predictor.

If the *Oldest Selection* predictor receives an update call, it simply updates concurrently the reference of the oldest selection. If the determined reference is not a null-pointer it sets it setup status on false to provide a request of the LoS predictor. As the LoS predictor requests the actual oldest selection, the *Oldest Selection* predictor simply gives the saved reference back.

The *Overlaps to Latency* predictor will not always really make an update if one is called. The decision is related to the underlying model and to the configuration parameters of the *Overlaps to Latency* predictor. If the underlying model is linear regression the activation of the refresh will always call an update procedure with the new observation as input. The only exception of this updating are the first observations, which shall not influence the model due false setup measuring. If the underlying model is MARS, the activation will not really activate a progress. Instead the model will collect the observations, which shall influence the model, until all of them were made. Beyond it will build its model and will moreover ignore following new observations.

The *Lifetime of Selections* predictor and the *IAT of Selections* predictor are working both on a time series prediction model. Thereby they are really similar in the workflow and we always starting them with the same time series prediction model by one parameter of the splitter. At the calling of an update of one of them, the linked connector of those predictors has its role. As we mentioned before each predictor with an time series prediction model has its connector. This means, that the splitter not really calls an update at the predictors itself, but at the related connector. The connector is there to provide an always full and stable set of input data for the linked predictor. So at starting the framework, the connector will start to collect the defined k observations and will only start an "update" of the real model, if it has collected k observations for the first time or a new observations has arrived. Note that if a new observation arrives at a connector and the set of observations has already k elements, the oldest one will be deleted and the new one will be inserted to provide a moving window on the last k observations. So if related connector of one time series predictor starts an "update" of the model, the predictor not really updates the model in cause of no available update function in OpenForecast [Gou]. Rather it builds a new model with the actual set of data. If there is a stable model available the setup the specific time series predictor is set to false. Note that the LoS predictor can only access a prediction, if there is no new model building in process. Otherwise the LoS predictor has to wait until the model building comes to its end.

If one of the time series prediction models where updated or rather new built, they will always activate a new refresh of the LoS predictor. Of course if there is a refresh in progress and a new refresh has been ordered before, they do not have to set the refresh value again on true. Because if the actual refresh of the prediction is done, the LoS predictor will take all actual models or values to make a new prediction. Note also that the LoS predictor will only make a prediction if no instance of the environment is in its setup. If the setup of all instances is fulfilled, we can be sure, that all predictors will give an actual result to the main-algorithm. If this is not the case, the LoS predictor could give results, with no common sense and that would cross all work behind the prediction. Because regardless of what the splitter will then display as an output, as soon as it gives one, the handling of such an output will have errors and mistakes as a consequence. Finally we also provide a petri-net in figure 5.2 to give also a viewable part on the workflow.

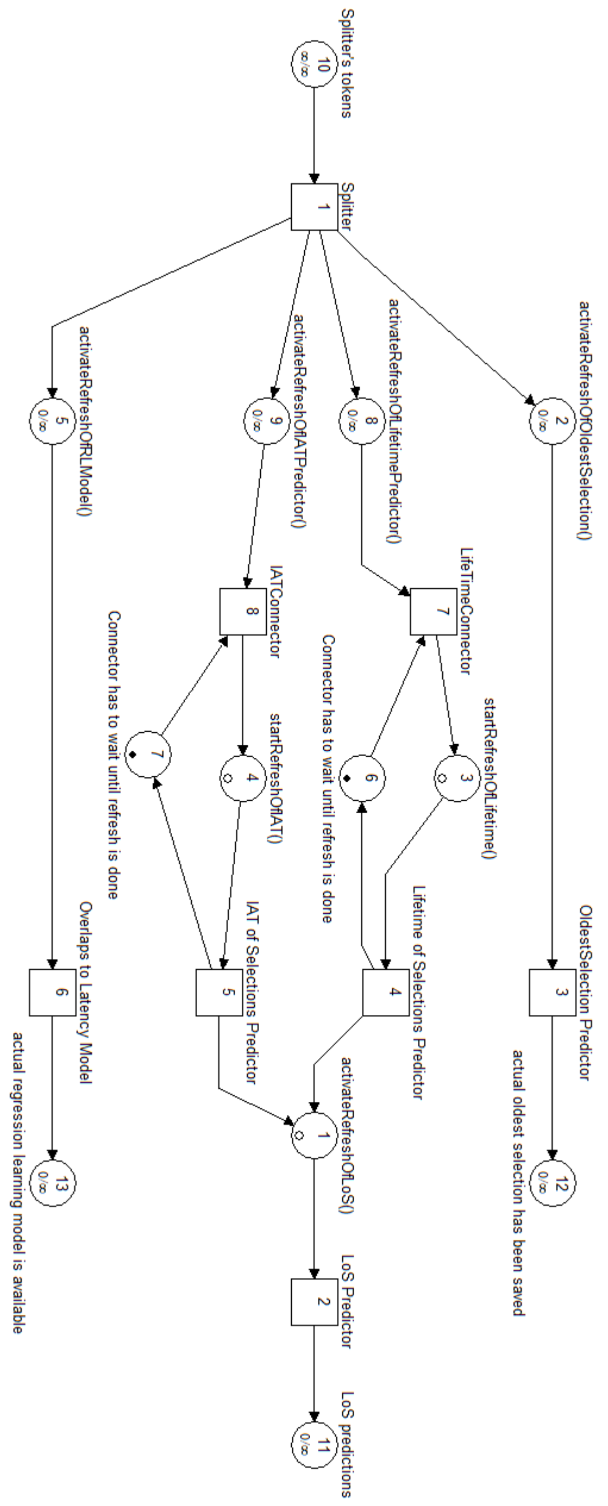

Splitter's tokens
10
∞/∞
Splitter
1
activateRefreshOfOldestSelection()
2
0/∞
activateRefreshOfLifetimePredictor()
8
0/∞
activateRefreshOfIATPredictor()
9
0/∞
activateRefreshOfRLModel()
5
0/∞
LifeTimeConnector
7
IATConnector
8
startRefreshOfLifetime()
3
Connector has to wait until refresh is done
6
startRefreshOfIAT()
4
Connector has to wait until refresh is done
7
OldestSelection Predictor
3
Lifetime of Selections Predictor
4
IAT of Selections Predictor
5
Overlaps to Latency Model
6
actual oldest selection has been saved
12
0/∞
activateRefreshOfLoS()
1
actual regression learning model is available
13
0/∞
LoS Predictor
2
LoS predictions
11
0/∞

Figure 5.2: The workflow of the LoS solving approach as petri-net.

6 Evaluation

So in this chapter we will evaluate our solving approach for the LoS issue. To structure the evaluation, we section this chapter again in three parts. At first we will introduce a moreover naive approach to deal with the LoS issue. This naive approach is there to give a comparison to our solving approach. For the rest of this chapter, we will went on like in section 3.3 the analysis of the LoS issue. So as the second part of this chapter we will describe the setup of our evaluation and the last section of this chapter will be the overview of the evaluation's examination results . In last section we will moreover conclude with further assumptions, which result on the evaluation results.

6.1 Naive Approach

As mentioned before, our approach needs a comparison with some more simpler approach. With this suggestion, we build a quite simple one, which is very naive in its assumptions at the latency function. We contemplate about this approach to see moreover, that our approach certainly tends in the right direction. So the naive approach should be one, which someone would think about, when he did not read this work nor did the research we did before. So at a first glance such a guy would think to solve the problem with the naive approach. Finally it should be clear that this naive approach does not work well and that indeed our approach is needed.

The most simplest way to handle the LoS issue should be to just predict the next upcoming latency with the knowledge of a few last measurements. So our naive approach takes in the last one hundred measurements of the latency to build up a moving average time series prediction model. With these measurements it will predict a forecast value. One prediction will be predicted as the truth for 500 ms in the future from the time stamp at the prediction. This naive algorithm to predict the latency of the splitting can be seen at algorithm 6.1. The

Algorithmus 6.1 Naive algorithm for predicting the LoS.

```
procedure GETNAIVEPREDICTIONOFLOS()
    prediction ← getCurrentForecastOfTSP()            // tsp is here a moving average model
    return Latency at "$currentTime + 500$" will be around "prediction" milliseconds
end procedure
```

workflow of the naive approach is just a simpler way of our one. The splitter will trigger a new observation of the latency to one connector of the time series model. This connector will force a model building of the time series prediction and at each new built model the predictor of the latency will activate the naive-algorithm to make a new prediction.

6.2 Setup

The hardware setup of the evaluation is exactly the same as the hardware setup of the analysis in chapter 3. We have again our two computers. On the one hand there is the main PC and on the other hand there is the second PC, whereby both are the ones of the analysis. So the main PC is equipped with 3x2 GB DDR3 RAM, an AMD Radeon R9 290x and a Intel(R) Core(TM) i7 930 CPU. The motherboard is an ASUS Rampage II Extreme and the operating system is Windows 7 Ultimate x64. Moreover the second PC is again the notebook and more detailed a Lenovo ThinkPad Twist N3C28GE 334728G (Edge S230u). The technical equipment of the second PC can be classified with 8 GB RAM, a Intel Core i7-3517U and a Intel HD Graphics 4000 (IGP). The operating system is Windows 8.1 x64.

As we now know again the PCs, which we use to evaluate, we can further describe the scenario we choose. Of course we examine again as before one whole PACE framework with two operator instances. The splitter will run thereby on the main PC and the rest of the framework will run on the second PC to not overwhelm the main PC with one whole framework. This outsourcing is necessary due the type of our evaluation scenario. Note that we will again use Round-Robin as scheduling method and no adaptive mode for the parallelization degree is enabled. In the analysis the scenarios were quite static in themselves. The tuple-based and time-sliding windows are of course static, because we define a size and a moving rate of them and then they will work with this assumption. A correct end of these scenarios is not really designated for the usage in practice. In cause of their simpleness a real end of such scenarios are only postulated, when the interest of the linked query wanes and is finally gone at the time of the end. So for our evaluation we just assume to run such a scenario several minutes long and assume at the end a end with no cars left at the lane. The traffic scenario of the monitored road to punish transgressors of the actual valid road regulations on this line, is on that account a better base for delivering a not so exactly formed scenario. The scenario has of course also only a real end, when the monitoring should stop, but we can define one by assuming a specific point of time where no more cars will arrive. So finally no cars will be between our two sensors at the locations pos_1 and pos_2.

Moreover the traffic scenario of the analysis can be formed more realistic. Because analysis' traffic scenario was just a scenario, where the inter-arrival time of the cars was defined by an exponential distribution with a specific form-factor. For the evaluation of our approach we take a traffic scenario of around 90 minutes length. Thereby the exponential distribution of the inter-arrival time of new cars will be increased and then decreased again. More detailed the scenario starts with a exponential distribution around 2 seconds and will concurrent increase by time to a exponential distribution around 200 milliseconds. After a short time on the peak of the inter-arrival time, the inter-arrival time will again decreasing concurrently by time to the start distribution around 2 seconds. The cars will need between 600 seconds and 900 seconds, whereby the 900 seconds are the normal case and only 10 percent of the cars will be faster than 900 seconds. Of these 10 percent speed violating cars the exact speed is uniformly distributed. So by these facts, it can be ascertained that all of our models has to perform a relevant work. The overlaps will increase and decrease, so the regression learning model has to perform well for all different predictions. And the both time series predictors are tested due non-static observations. Because the lifetime of the selections fluctuates due the differences of the specific car speed. And the inter-arrival time of the selection fluctuates, since it is calculated by a exponential distribution and this distribution furthermore changes during the scenario. With these evaluation scenario we assume a most realistic one as actual possible for us.

Note that we will take some alteration on the scenario for the run with the naive approach. At this special run, we just could run it with predictions running all time during the run at a smaller scenario. So all parameters of the scenario are the same instead of the length of the scenario. The shorter scenario is just around half of time of the original evaluation scenario and so the number of arriving cars during this scenario are about the half of the longer one. The speed of the cars is the same as before, but on the dynamical change of the inter-arrival time is also some alteration to be made due the shorter scenario length. The increasing and decreasing of the inter-arrival time of cars is the same by the exact exponential distribution, but the specific distributions will used the half of the time as used in the original scenario. We use this special scenario for the evaluation of the naive approach in cause of a unknown issue at the runtime. The point is that the prediction just freezes after around one quarter of the evaluation scenario, but with the shorter or rather tighter scenario it works fine. We can exclude a memory leak, due the fact, that our approach works fine with the evaluation scenario and the naive approach needs significant less memory space. Finally you will see, that we can nevertheless see obvious results on the naive run with the tighter scenario.

6.3 Results

As mentioned in chapter 5, we have a few parameters to define, when we want to activate our solving approach. We appoint that our time series prediction models on a size of one thousand observations. So at each point, the lifetime prediction and the inter-arrival time prediction will give out a forecast, which is calculated on the last one thousand observations. As we have the possibility to choose between two different time series prediction models and two different regression learning models, we started five different runs. Four runs for all possibilities of our approach and one run for the naive approach. To structure the assessments of the evaluation, we go on with four different subsections. At first we will detail the assessments on the runs with linear regressions. Afterwards we take a look on the two runs with MARS and then on the

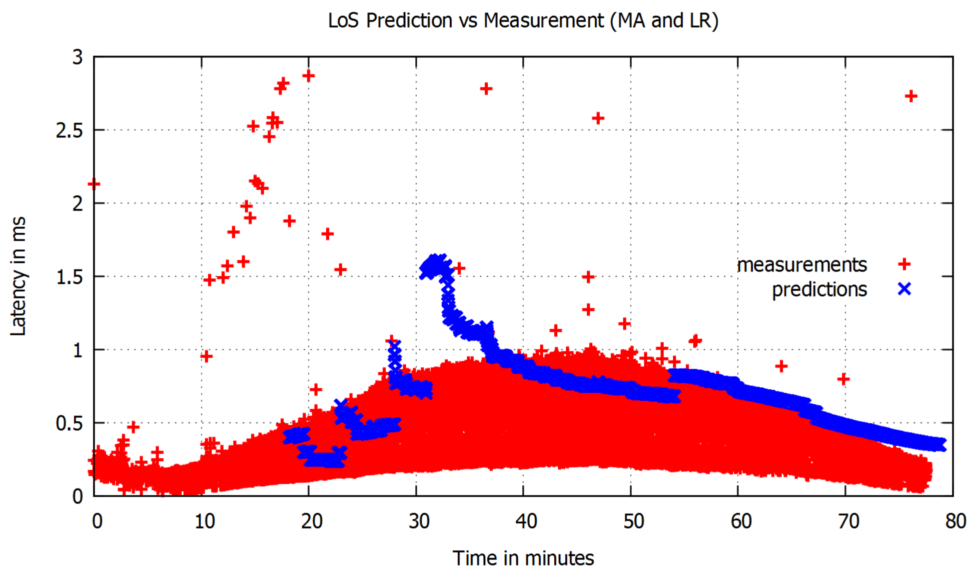


Figure 6.1: Plot of the evaluation run with moving average and linear regression.

run of the naive approach. Finally we will close this section and thereby the chapter with a concluding assessment of the whole evaluation.

6.3.1 Linear Regression Runs

We decided for the linear regression to let the first one hundred observations of the latency pass and not deliver them to the regression model. Thus all needed parameters for the two runs with linear regression are defined. In figure 6.1 it is to see that at the beginning we first need to setup all models. After around 18 minutes, the first predictions are made. We can observe that the observations are mostly smaller than one millisecond except some outliers. These outliers were also to observe in the analysis. In our assumption, these outliers are a consequence of the scheduling of Windows. At around the half of the time the most overlaps occur, thus the measured latency is there higher than at the rest of the scenario. The predictions are jumping some times at the beginning and one more time to the end of the scenario. We assume there two factors, which play a role for these jumps. On the one hand at the first predictions, the linear regression is may not stable enough due the measured outliers. And on the other hand, may also the time series predictions of the lifetime and the inter-arrival time play a role due a little false prediction. The main influence on these jumps should be settled at the fact of the not stable linear model, because when we take a look at the predicted overlaps we can not find a obvious high miscalculation. Remind that the predicted count of overlaps is the result of both

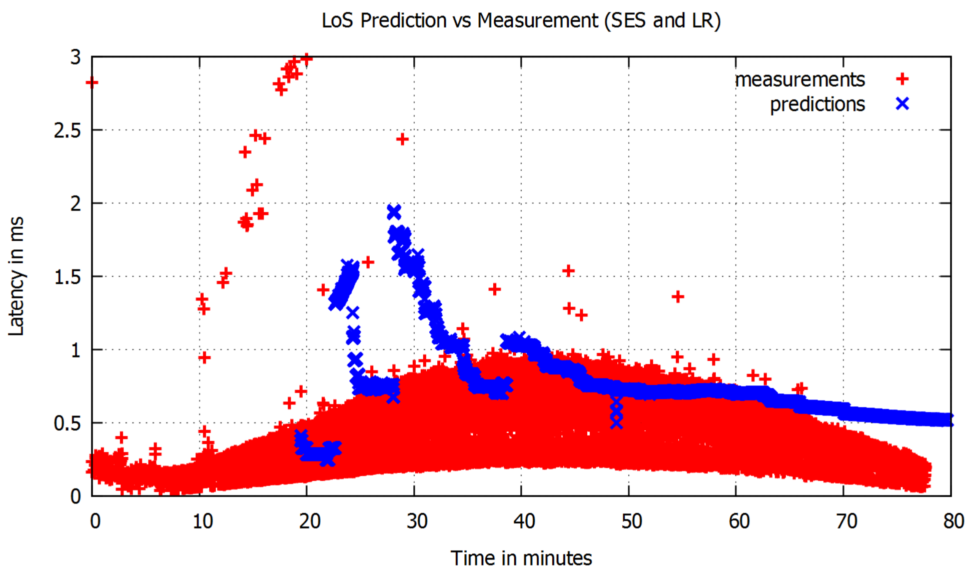


Figure 6.2: Plot of the evaluation run with simple exponential smoothing and linear regression.

used time series prediction models. On a final note, we can observe a quite good prediction field at the end of the scenario.

The second run with linear regression as our overlaps to latency linking model is to see in figure 6.2. At this run we take in the simple exponential smoothing to build our time series prediction models. The measurements are nearly equally distributed as before at the first run with moving average. The biggest difference is to see at the predictions. The predictions start as before around 18 minutes after the start, and at first they are quite similar to the first run. But at the first jump of the predictions we observe two really high jumps. The predicted count of overlaps is again quite fine in the logs. Thus we assume a difference at the linear regression model due other measurements or rather not in the same sequence as in the first run. So we determine that the jumps at the beginning of the predictions or rather at the time of a quite less influenced regression learning model could be quite high. The good thing on this run is the part of the predictions from around 40 minutes. There we can see a quite stable sequence of predictions. As we have no influences in the model from the first one hundred measurements, we see also that to the end the predictions get even higher than the real values.

6.3.2 MARS Runs

So in the two runs with MARS we have one more parameter than at the first two runs with linear regression. Initially we have to remember that we will not update MARS at any time. This is the case due the long building time of a stable MARS model and no presence of a

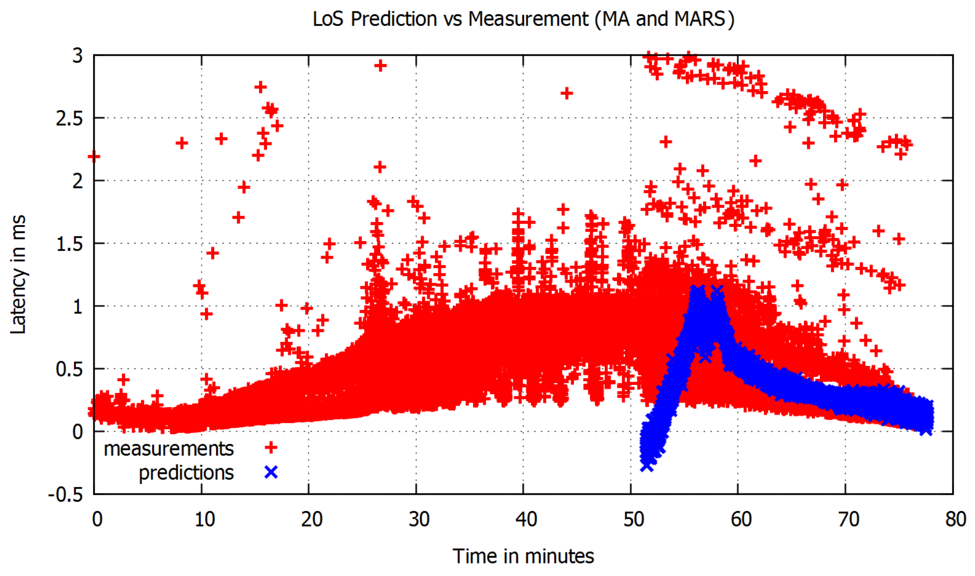


Figure 6.3: Plot of the evaluation run with moving average and MARS.

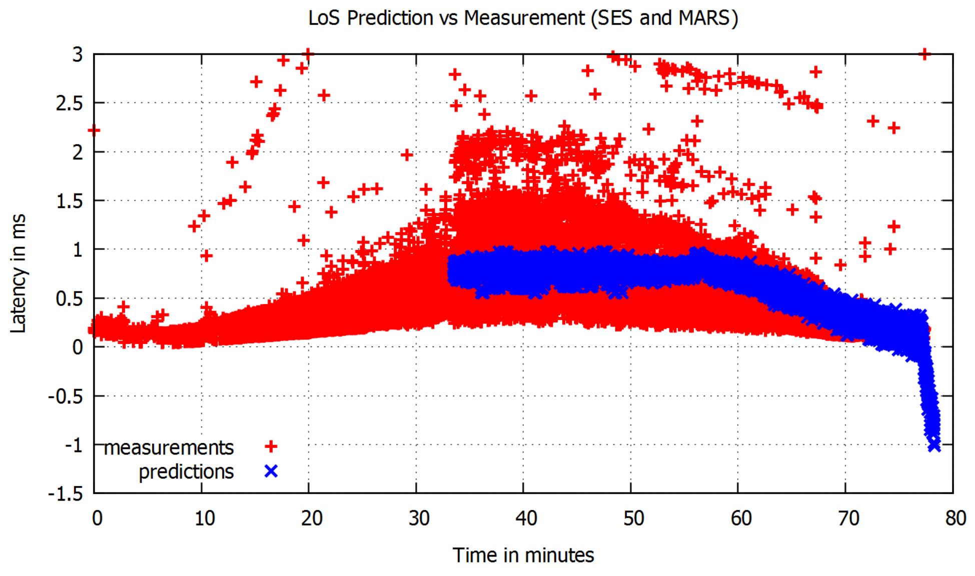


Figure 6.4: Plot of the evaluation run with simple exponential smoothing and MARS.

real update function. So to have a powerful model after building it we need a big count of observations, which function as the training set. Finally we decided due the knowledge of reading the logs of the first two runs and a testing of the MARS model 7000 observations should be enough for this scenario. Moreover, we have the chance to calculate for every upcoming count of overlaps a meaningful prediction, if we start thereby to collect these 7000 observations after letting the first 500 pass. So with a quite big expert knowledge we can define the two remaining parameters for the runs with MARS with 500 for the first passing observations and 7000 for the number of influencing ones. Thereby we should start building the MARS model at around 23 minutes after our starting point.

In figure 6.3 we can see the run with moving average and MARS. We obvious see there more outliers of the measurements than at the runs with linear regression. The predictions are really different to the runs with linear regression. In the plot of the run with moving average the predictions are firstly made really late in our scenario. So the building of the model took there a several sequence of time. The most predictions made are quite good integrated in the cloud of the most observations. But in addition we can also observe negative predictions. These predictions do not make any sense for usage afterwards, but they are calculated at positions where we do not have any observations in our training set.

The second run with simple exponential smoothing and MARS is quite different and to see in figure 6.4. The measurements have as in the first run with MARS more outliers than in the runs with linear regression. But the predictions are really different to the first run with MARS. This difference is again not really influenced by the changing of the time series prediction model, because the count of overlaps is again quite well predicted. The really difference is thereby at the MARS model. One the one side the building time was distinctly faster. On the other side the predictions for a count of overlaps, which is higher than the biggest count in our training set is not negative. At least for the counts, which we did not observe for the building and which are smaller than the smallest observed one, have really high negative values in the prediction. So we see that MARS is not really stable at extrapolation but stable at interpolation. In other words, MARS is really good at the range where observations are in the training set. But for values of the independent value, which are not in this range, MARS could be ok, but also really awful.

6.3.3 Naive Run

As mentioned before, the naive approach has a not solved issue with the evaluation scenario on the two PCs. This issue may not appear on other machines and so we use the tighter version of our evaluation scenario for this run. So we have now about a total time of 45 minutes. Figure 6.5 and figure 6.6 plotting both the same logs. To see more details on the run we just plotted it two times with different scales at the dimension of the latency. Of course the setup time of the approach is really small. It is so small that it is may not even to mention as a setup time. We only need 5 observations and afterwards we can predict 500 milliseconds in the

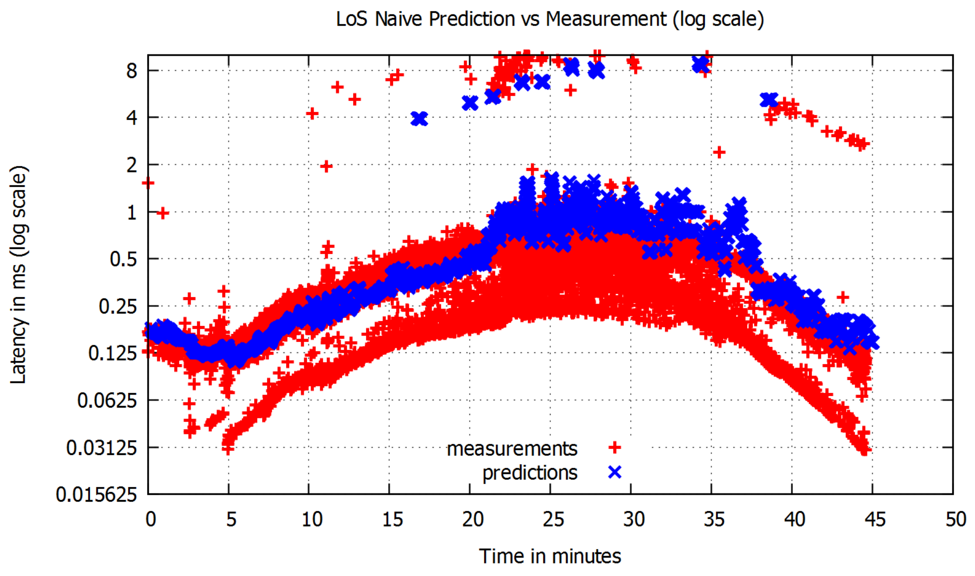


Figure 6.5: Logarithm scale plot of the naive approach run.

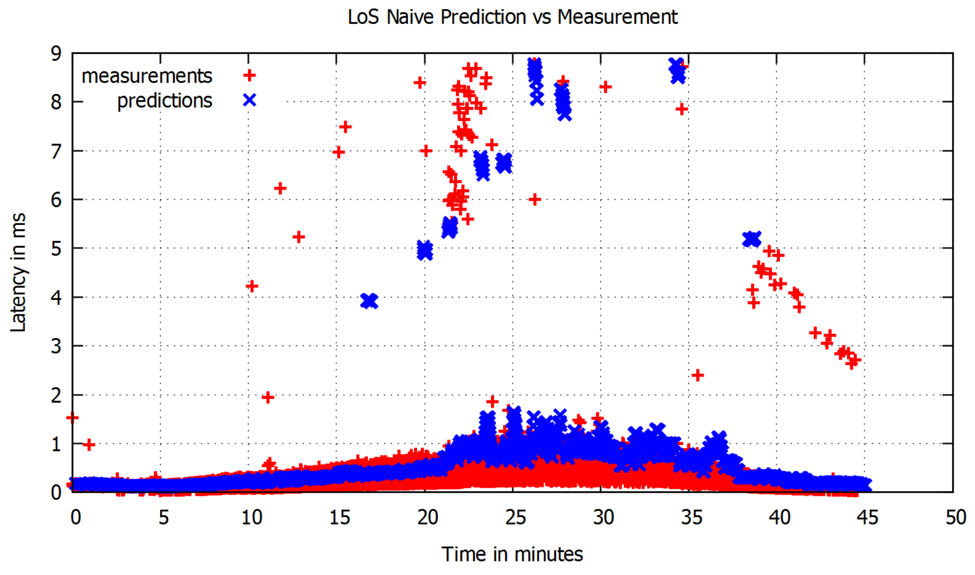


Figure 6.6: Plot of the naive approach run.

future. Remind that there will be not more than 100 last observations to forecast the next latency. With this fact we can see in the both figures that the prediction is never so stable such our solving approach. Especially the outliers in the measurements are a plague for the naive prediction, because the influence of those is to high. So if there are any outliers observed, the naive model is easily teared. And with the high scaled figure 6.6 we can also observe big outliers in the prediction. Such outliers in the predictions are not desirable, because they could have miserable consequences. For instance, if we have a distinct high prediction, we assume as a consequence that we need more operator instances to deal the upcoming event rate. We assume this in cause of better be prepared for the worst case than not fitting in the given latency budget. If it then demonstrates that we not need this more computation power, we are not achieve the goal of never have more operator instances at work as needed to handle the scenario in the given latency budget.

May it seems to be so that the naive approach performs better than our solving approach. It is the truth, that the naive approach performs really well in this scenario, if we filter the outliers. We want to be as near in practice as possible, but if we assume a scenario with a high fluctuating count of overlaps, the naive approach will really fail. Because if for instance some events closes more than 100 selections at once, the naive approach will result in many outliers. If there are many outliers, which we then want to filter, the results will be only sparsely distributed over the whole scenario. Our solving approach will there be more stable, because it just performs on the knowledge over the overlaps and not only on current trends.

6.3.4 Final Assessments

So we observed with the evaluation that our solving approach does not influence the latency to much. Because we can compare the measured latencies with the latencies of the analysis in section 3.3 and all over the latencies are always in the same magnitude. As we contemplate on the plots, we can conclude a equality in the usage of moving average or simple exponential smoothing for the time series predictions. So our recommendation is the usage of the moving average due a bit smaller workload in cause of the same weights for each influencing observation. An other assessment on the evaluation is the fact, that all runs of our approach are quite better than the naive one, in cause of being more stable in the predictions.

Our approach does not predict really perfectly due setup or rather neither doing any prediction at the setup time. Thereby we see some differences, where the two different regression learning models could be the better one to deal with the solving of the LoS issue. Which model is auspicious to take for a specific scenario can be different concerning the superior interest, which defines the needed information and so the query for the whole CEP system. The linear regression model could be better for a scenario, where it is needed to make updates on the regression learning model. Moreover the linear regression will probably achieve better predictions for values, which were not in the range of the observed ones. This holds on the one hand in cause of the single basis function used in linear regression and the observed probably linear increasing of the latency at a higher overlap. MARS can achieve good predictions outside of the observed range but the probability is linked to the used splines in the model and they may do anything outside the range of observed values. We also observed this in the plots above. On the other hand the linear regression will probably make better predictions outside the observed range in cause of the possibility to update the model. Thus in practice it is a small probability to need a prediction far outside of the observed range. And even if, after the next update the distance between the needed prediction and the model is perhaps smaller than before.

MARS in contrast will probably deliver better results for values inside the range of observations, because the different basis functions shrink the standard error to all observations compared with the linear regression. Linear regression tries to keep the error low to all observations with the limitation of just one linear basis function. We can observe in the analysis of the LoS issue that the latency increasing respectively to the count of overlaps nearly linear, but at a higher workload with much more overlaps this does not have to be the truth. And there has MARS with its different basis functions a advantage to deal with. Moreover we observed better results in the range of observation, when we used MARS. So we can assume that MARS will always give better results, when the to be predicted value is inside the observed range. MARS could thereby be also the better solution for specific scenarios. If we could include for instance in a MARS model all upcoming traffic of a road over two weeks, the built model should be large enough to deal all upcoming traffic. Note that this assumption can also be broken by the fact of more traffic at holidays.

So you see that there are use-cases, where MARS is the better choice, and there are cases, where linear regression is the better choice. In any case, we need obvious more expert knowledge to perform a prediction with MARS. So if you do not have any expert knowledge, which you can use, our recommendation is to use linear regression. But if we can observe nearly the whole range of possible overlaps and have a profound expert knowledge and have time to built with the observations a model, MARS is definitely the better choice.

7 Conclusion

This work did a first step in the direction to stay with a parallelized CEP System always in real-time computing. More in detail, we can now easier stay in real-time computing with a CEP system, which implements the PACE framework. At first we showed the essential need of knowing the latency of the splitter and the merger at a specific CEP or rather PACE node. With this knowledge we can easily adjust and prepare the node for upcoming scenarios, thus the correlation calculation will stay in the specific latency budget for this node. Thereby we will not have to care, whether much events arrive at nearly the same time or not. Because the prediction shall be automatically and concurrently calculated. Further on we detailed the problem, of how to predict the latencies of the splitter and merger in less complex issues. Of course we can first split at the point of the different instances. So we need a different prediction algorithm for the splitter than for the merger. By contemplating more in detail about the splitter we marked again to subparts. The splitter's latency can moreover be split in two influencing parts, which it has to calculate. On the hand there is the scheduling of the actual opened selections, which will especially influence the latency as more complex the scheduling method is. A complexer scheduling can have many more advantages to use than the probably simplest version. Because instances can be distributed over the available instances such that nearby no communication overhead exists. Thereby it has to be regarded that also no operator instance should be overwhelmed by to much workload, because that was the fact we want to circumvent by parallelizing the CEP nodes. On the other the splitting is also influencing the latency of the splitter due on every event has to be decided if it starts a selection and or or closes one selection. In this work we investigated more in detail on the *Latency-of-Splitting* (LoS) issue. Therefore we first defined this issue and further analyzed influencing parameters of the LoS. The definition is to find at section 3.2 and concluding we can say, that the LoS issue is the problem of have a knowledge how to predict the latency of the splitting.

At the analysis of the LoS issue we make different runs of a PACE node with a mostly simple scheduling approach to get moreover the best fitting how the prediction could be established. By running different scenario of different complexity we finally assume after the analysis that the most influencing part of the splitting is the count of overlapping selections. This is the size of the set of all currently opened selections and is saved by the splitter for a easier scheduling and the communication with the merger by closing one selection. The assumption seems for us very legit due the calculation of the predicates PStart and PClose. These predicates state the decision upon a specific event starts and or or closes one selection. The calculation of these are quite simple. But to decide whether a selection should be closed by this event or not, the splitter has to go trough the hole set of opened selections. Thereby it has to determine PClose for every pair of opened selections and the specific event. Based on this final assumption we invented a solving approach which is introduced in chapter 5. At the evaluation with a scenario of a traffic monitored lane, our approach did quite well. It was shown that our approach needs some setup time to predict stable latencies but the error seems to be all right. Also the new workload for the splitter by our extensional approach is not important to mention due the fact, that the latency was not noticeable increased. We also showed in the evaluation, that a naive approach would definitely predict more worse than our approach. We also mention that it is equal, which time series prediction model we use for our approach. By contrast the use of a specific regression learning model is direct linked to the superior interest or rather query of the user or application.

Future work can focus in many different things. On the one hand there are still many parts left up till the point of knowing the hole latency of the splitter and the merger is reached and getting a really chance to handle all upcoming situations within the latency budget. On the other hand the prediction of the LoS issue could maybe optimized a bit. The big expert knowledge to start our approach should be getting smaller by time. Because we have many parameters, thus expert knowledge is always needed and not only at the usage of MARS, but of course by using MARS we need distinct more expert knowledge. May we can also optimize our approach due other underlying models than our suggested ones. If MARS will anytime be the state-of-the-art to use, when a regression learning model is needed, or is the best for the scenarios, which could happen in the future processing of the CEP system, FastMARS [CSF93] by Friedman could also be the answer. We assumed at first to just provide easy models, because FastMARS goes a head with some heuristics to build the model faster and FastMARS will also may not be needed anyway due no query where we need a MARS model and a new built model after some time. Because we saw that if we can have a learn period for MARS quite everything should be fine for any event-load upon this specific node. Moreover we do not have a good exampling scenario, where we need FastMARS instead of MARS. But maybe it could be a idea to handle the hole prediction more perfectly.

Bibliography

[AGPS15] C. Athuraliya, M. Gunasekara, S. Perera, S. Suhothayan. Real-time Natural Language Processing for Crowdsourced Road Traffic Alerts. 2015. URL http://www.researchgate.net/profile/Cd_Athuraliya/publication/281208029_Real-time_Natural_Language_Processing_for_Crowdsourced_Road_Traffic_Alerts/links/55db4caf08aec156b9afe73a.pdf. (Cited on page 12)

[Ami06] G. Amis. MARSplines.java, 2006. URL http://www.amisworks.com/MARSplines.java. (Cited on page 36)

[apa] Apache Commons. URL http://commons.apache.org/. (Cited on page 33)

[BDWT13] C. Balkesen, N. Dindar, M. Wetter, N. Tatbul. RIP: run-based intra-query parallelism for scalable complex event processing. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 3–14. ACM, 2013. (Cited on pages 14 and 15)

[BK09] A. Buchmann, B. Koldehofe. Complex event processing. *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 51(5):241–242, 2009. (Cited on page 11)

[BMK+11] A. Brito, A. Martin, T. Knauth, S. Creutz, D. Becker, S. Weigert, C. Fetzer. Scalable and low-latency data processing with stream mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 48–58. IEEE, 2011. (Cited on page 14)

[CCA+10] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, R. Sears. MapReduce Online. In *NSDI*, volume 10, p. 20. 2010. (Cited on page 14)

[CSF93] S. U. L. for Computational Statistics, J. H. Friedman. *Fast MARS*. 1993. URL http://www.milbo.users.sonic.net/earth/Friedman-FastMars.pdf. (Cited on page 58)

[EB09] M. Eckert, F. Bry. Complex event processing (CEP). *Informatik-Spektrum*, 32(2):163–167, 2009. (Cited on page 11)

[Fri91] J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pp. 1–67, 1991. URL ftp://ftp.uic.edu/pub/depts/econ/hhstokes/e538/Friedman_mars_1991.pdf. (Cited on pages 32, 33 and 34)

[Gou] S. Gould. OpenForecast. URL http://www.stevengould.org/software/openforecast/index.shtml. (Cited on pages 37 and 45)

[HHKA14] N. R. Herbst, N. Huber, S. Kounev, E. Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and Computation: Practice and Experience*, 26(12):2053–2078, 2014. (Cited on page 37)

[HTF09] T. Hastie, R. Tibshirani, J. Friedman. *The elements of statistical learning*, pp. 321–328. New York: Springer, 2nd edition, 2009. (Cited on pages 6, 34 and 35)

[KKR10] G. G. Koch, B. Koldehofe, K. Rothermel. Cordies: expressive event correlation in distributed systems. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pp. 26–37. ACM, 2010. (Cited on page 11)

[KMR$^+$13] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, M. Völz. Rollback-recovery without checkpoints in distributed event processing systems. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 27–38. ACM, 2013. (Cited on pages 11 and 13)

[KORR12] B. Koldehofe, B. Ottenwälder, K. Rothermel, U. Ramachandran. Moving range queries in distributed complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pp. 201–212. ACM, 2012. (Cited on pages 11 and 13)

[KTKR10] G. G. Koch, M. A. Tariq, B. Koldehofe, K. Rothermel. Event processing for large-scale distributed games. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pp. 103–104. ACM, 2010. (Cited on page 12)

[Luc02] D. Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002. (Cited on page 11)

[MBF14] A. Martin, A. Brito, C. Fetzer. Scalable and elastic realtime click stream analysis using streammine3g. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 198–205. ACM, 2014. (Cited on page 14)

[MKR14] R. Mayer, B. Koldehofe, K. Rothermel. Meeting predictable buffer limits in the parallel execution of event processing operators. In *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 402–411. IEEE, 2014. (Cited on pages 11 and 14)

[MKR15] R. Mayer, B. Koldehofe, K. Rothermel. Predictable Low-Latency Event Detection with Parallel Complex Event Processing. *Internet of Things Journal, IEEE*, PP(99):1–1, 2015. doi:10.1109/JIOT.2015.2397316. (Cited on pages 6, 11, 12, 14, 15, 16 and 17)

[MTKR15] R. Mayer, M. Tariq, B. Koldehofe, K. Rothermel. PACE: Bandwith-Efficient Real-Time Scheduling in Parallel Complex Event Processing, 2015. UNPUBLISHED. (Cited on pages 10, 16, 18 and 19)

[NRNK10] L. Neumeyer, B. Robbins, A. Nair, A. Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pp. 170–177. IEEE, 2010. (Cited on page 14)

[OKR$^+$14a] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, U. Ramachandran. MCEP: a mobility-aware complex event processing system. *ACM Transactions on Internet Technology (TOIT)*, 14(1):6, 2014. (Cited on page 12)

[OKR$^+$14b] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, U. Ramachandran. Recep: Selection-based reuse for distributed complex event processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 59–70. ACM, 2014. (Cited on pages 11 and 13)

[OKRR13] B. Ottenwälder, B. Koldehofe, K. Rothermel, U. Ramachandran. MigCEP: operator migration for mobility driven distributed complex event processing. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 183–194. ACM, 2013. (Cited on pages 11 and 12)

[SGLN$^+$11] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, V. Nanayakkara. Siddhi: A second look at complex event processing architectures. In *Proceedings of the 2011 ACM workshop on Gateway computing environments*, pp. 43–50. ACM, 2011. (Cited on page 14)

[SKRR13] B. Schilling, B. Koldehofe, K. Rotherme, U. Ramachandran. Access Policy Consolidation for Event Processing Systems. In *Networked Systems (NetSys), 2013 Conference on*, pp. 92–101. IEEE, 2013. (Cited on pages 11 and 13)

[sto14] Storm, 2014. URL `http://storm-project.net/`. (Cited on page 14)

[VKR11] M. Völz, B. Koldehofe, K. Rothermel. Supporting strong reliability for distributed complex event processing systems. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 477–486. IEEE, 2011. (Cited on pages 11 and 13)

All links were last followed on Ocotber 19, 2015.

Declaration

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

place, date, signature