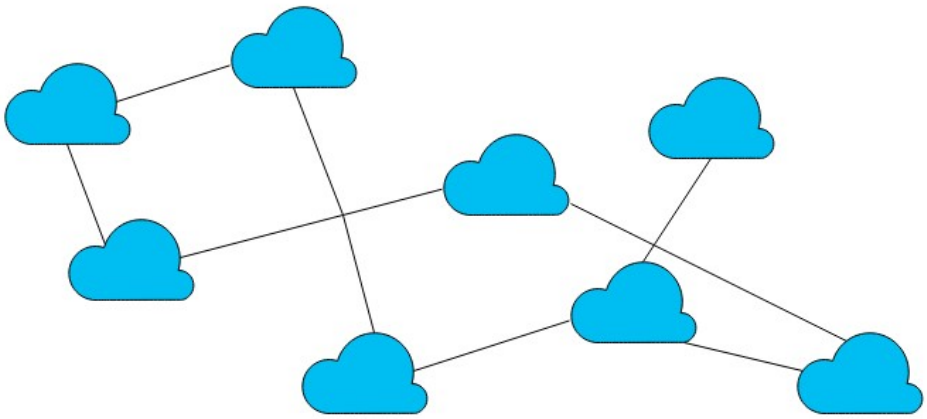

Evaluating Different Approaches for Predicting Task Execution Time - A Case Study in a Distributed Production Environment

Master's thesis in Computer science and engineering

JESPER CARLSSON
ERIK FORSSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Evaluating Different Approaches for Predicting Task Execution Time - A Case Study in a Distributed Production Environment

JESPER CARLSSON
ERIK FORSSTRÖM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Evaluating Different Approaches for Predicting Task Execution Time - A Case Study in a Distributed Production Environment
JESPER CARLSSON
ERIK FORSSTRÖM

Supervisor: Vincenzo Gulisano, Department of Computer Science and Engineering
Advisor: Viktoria Nilsson and Otto Bergström, Recorded Future
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Evaluating Different Approaches for Predicting Workflow Execution Time - A Case Study in a Distributed Production Environment

JESPER CARLSSON
ERIK FORSSTRÖM
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This project details the evaluation of several machine learning models used to predict the required processing times in a a complex distributed system used to analyze large amounts of data. Specifically, the system is owned and developed by the company Recorded Future, who specialize in sifting through vast amounts of textual data acquired from a variety of online sources in search of threat intelligence they can provide to their clients. The input is pre-processed in several stages before it arrives at the main analyser process, where natural language processing and other tools are used to perform a threat analysis of the text. Our primary goal is to determine the time needed to analyse one of these texts. The ability to predict the time required to process a given set of input can be used to design scheduling algorithms in cloud computing environments [1]. It is of extra interest to Recorded Future as they use the size of message queues, which might grow large when processing takes too long, in order to decide when to start additional servers. As servers take some time to go online, being able to start them proactively based on estimated queue size, which can be inferred from the required processing time and the available computing resources, can alleviate problems with bottlenecks and other performance issues. RF has defined a maximum error within one order of magnitude compared to the actual time to be acceptable for the purposes of workload estimation.

To accomplish this goal, we have developed, trained, and tested several prediction models based on neural networks. Each network considers a different set of input features that may affect the processing time - information extracted from the input data, system performance at the time of analysis, total server workload in terms of input processed in parallel, and past processing times. For evaluation, the prediction error from two naive algorithms that predict the mean and median value of the task execution times in each data set is compared to each models error.

Our results show that all but one of the prediction models achieve a lower error than using the naive approach, and all models perform better than the maximum error specified by RF. There is a trade-off between how feasible it would be to implement and use a model in the real system, and the achieved accuracy. The model that considers system performance achieves an error that is half that of the one based purely on input information. Considering the total workload of each server reduces the error by a negligible compared to the first model, and using previous task execution times is shown to provide fluctuating results, indicating it is not a suitable model to use for prediction in this system.

Keywords: distributed system, processing time prediction, EC2, task execution time, neural network, time series prediction.

Acknowledgements

We would like to thank our supervisor, Vincenzo Gulisano, for providing guidance and advice throughout this project, as well as helping us make this report as good as possible. We would also like to thank everyone at Recorded Future who have aided us - primarily our supervisors Viktoria Nilsson and Otto Bergström, but also all of the people that have taken time out of their busy days to help us with everything from improving our machine learning models to make our data retrieval processes not crash the production environment. A big thank you to Karl Bäckström for helping us with Machine Learning theory and helping us tweak and finalize our models.

Finally, we would like to thank Devdatt Dubhashi for being our examiner.

Jesper Carlsson and Erik Forsström, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1

Introduction

The advent of *big data* has in many ways revolutionized large parts of our society [2]. Information has become highly valuable - everything from where someone lives to what their preferences are when it comes to certain products can be used to conduct directed marketing, estimate consumer demand, or even try and predict the outcome in political elections. Today, information from just about any source is potentially interesting and exploitable for some practical goal like improving an existing product or service, or gaining increased business insight. However, being able to make sense of all this information is far from easy [3, 4]. Often, treating huge volumes of information requires a complex system consisting of many components that each handle some small part of the overall processing chain [5].

As an application grows in terms of features and users, increasing the complexity, e.g. by adding more interconnected processes, and the size, in terms the number of servers running concurrently, in the underlying system is necessary to handle the increased load. A prominent example is the architecture powering Google's services, consisting of an estimated one million servers [6]. While these large-scale systems enable developers to create complex software that can serve millions or billions of users without sacrificing application performance, building and maintaining such systems presents a number of challenges [7, 8].

One pertinent issue developers and users of such a system can have is that, due to this complexity, it can be very hard to know in advance what effect on the system a scheduled operation will have. During the execution of a task, it can be hard to know exactly which of the interconnected processes will be involved, and transmission delays or even process failures introduce uncertainty as to *when* the input will be processed. In this context, "effect" can be anything from which processes may be involved to how a specific database will be modified [9]. A related aspect to this is the processing time required for a task. Investigating the time it would take to perform a specific operation is interesting not only because it is generally important to know when a result will be available, but also because it can help in detecting bottlenecks and other problems, should a certain operation take far longer than what is considered reasonable.

In addition, the longer an operation runs for, the longer it takes for the computing resources it uses to become available for other operations. In distributed systems using automatic scaling more resources are allocated should the system detect that operations are not being executed quickly enough. The ability to handle this pro-

visioning of resources based on fluctuating workload is known as *elasticity* [10]. A system with a high degree of elasticity is less likely to run into the problem of input being processed at a sub-optimal rate, but the ability to automatically start additional servers introduces new challenges. One of these is caused by some operations being far less interesting than others in terms of how relevant the result is to the users of the system. Thus, a scale-up of the system - a costly procedure, as many companies rent their cloud infrastructure from providers like Amazon Web Services (AWS) where cost is determined by usage - could be caused by tasks that are considered low-priority. All of this results in the deceptively simple question *"how long will this operation run for?"* being highly relevant, but also highly complex to answer. This project attempts to do just that, in a computing system used for threat intelligence.

1.1 About the Problem

Recorded Future (RF) is a company specializing in real-time threat intelligence, providing its clients with aggregated information about potential cyberattacks that may target them [11]. Over one million web-based sources provide between 10 to 50 million new text files, henceforth referred to as *documents*, each day. During analysis, these documents are processed in multiple stages including natural language processing, search indexing and duplicate removal, requiring 15 billion database look-ups. The system used to perform this extraction consists of a combination of asynchronous queues and processes that run in parallel. Hundreds of concurrently running servers provide the performance required to provide this insight in real-time to their clients. RF runs their service in the cloud, renting infrastructure from AWS. To handle fluctuating amounts of workload, the system is configured to scale up, i.e., start additional instances, when a monitoring process detects that certain document queues grow beyond an acceptable length.

In addition to this automatic document retrieval, called *polling*, the teams working with data science may sometimes order a *re-analysis* of some existing data for various reasons. These jobs often consist of hundreds of thousands of documents and they often constitute a large part of what is analyzed at any point in time, and they are thus a very large source of system workload. However, they are usually not considered as important as standard jobs - the reasoning being that brand new information can provide more and better intelligence to clients than new conclusions drawn from existing data. RF would like to be able to forecast how long one of these re-analysis jobs will run for.

Unlike the standard gathering and analysis of documents, a re-analysis has to be ordered manually, meaning the data scientists often have some idea of what the gain will be in terms of what new and relevant information it would produce. However, this might not be worth it if the jobs running time would necessitate a large up-scale of the system in order to maintain Quality of Service (QoS) since the monthly costs are based on total usage.

1.2 Goals and Contributions

The goal for this project is to evaluate the accuracy of several machine learning based approaches to predicting the required processing time of input in RF's system. The *accuracy* of the predictions will be measured in terms of mean absolute prediction error (MAE). This was chosen as the error metric as it is intuitive and easy to interpret - a MAE of x means that the model is, on average, x milliseconds off from the real value. Depending on the use case, the amount of prediction error that is acceptable can differ significantly. For RF's aim of gauging the amount of workload entering the system at a given point in time, an error within one order of magnitude (0% - 100%) has been deemed acceptable.
Additionally, the validity of implementing each model in the real system will also be evaluated. Thus, the achieved error will be compared to how complex it would be to use in practice.

1.2.1 Challenges

Machine learning has been around for some time and there are now plenty of tools and libraries that can be used to build prediction models without requiring much overhead [12, 13, 14]. In addition, there are vast amounts of information available - RF has tens of billions of documents in their databases, and detailed information about hundreds of metrics related to system performance is available. However, there are many challenges that will have to be overcome in order to build a prediction model that is of any practical use, i.e achieving a satisfying prediction accuracy.

1.2.1.1 Access to data and detailed information

While there is plenty of document-specific information available - both in terms of number of data points and level of detail - there exists no staging or testing environment for RF's system. Data cannot be retrieved without affecting the performance of the production system that is shared with the clients. For this reason, only a comparatively small amount of data can be studied.
Another consequence of this is that different models require different data sets. Data related to the various aspects investigated are stored in different databases, where they are available for varying periods of time after the analysis is complete, and in varying quantities. Thus, it is impossible to perform completely fair comparisons between the models. The approach taken to obtain inter-model comparable errors are described in the methodology.

1.2.1.2 Problems when investigating a system used in production

Somewhat related to the above problem is the chaotic nature of a system used in production - there is no way to conduct controlled experiments as parts of the system cannot be shut down, and it is not possible to isolate specific processes to study them in more detail. Furthermore, results yielded from any such controlled study would likely not be representative of the production environment.

1.2.1.3 Understanding system state

RF maintains huge volumes of metrics from nearly all parts of their system - as will be discussed in theory, information about everything from message queue length to individual server CPU-usage is available. However, this does not mean that it will be easy to find which parameters (or combination of parameters) that are key to understanding processing time fluctuations. This is discussed further in theory.

1.2.2 Contributions

While the end result will ideally be a functioning and accurate model that RF can use to predict analysis times in their production environment, findings during the development process in combination with a continuous evaluation of the models performance will provide insight into the following areas:

- Which features in the document metadata (information gained from analysing the input) affect processing time the most?
- In a complex, distributed system like the one used at RF there are a plethora of parameters related to system performance that constantly fluctuate. How much does this "system state" affect task execution time? Among these parameters, is it possible to isolate the ones that are most relevant for the processing time?
- The document analyser process is multi-threaded. If the other threads on the machine are analysing large documents, to what extent does this affect the task execution time in the thread where the investigated document is being processed?
- Considering the volatile nature of the system, is it possible to use previous analysis times to predict future ones?
- What happens to the prediction error if input is aggregated into larger batches, and the model is trained and tested on these?

While these findings are of great interest to RF, they are far from the only ones using a large, complex system to perform some type of data processing. Hence, the methodology in this project can act as a guide to others who are interested in performing similar investigations. Also, the results from evaluating model performance when trained on different features functions as a practical examination of existing methods used to predict task execution times.

1.3 Limitations

The system used by RF consists of a large number of processes, each performing one part of analysis. To reduce the scope of the problem, this investigation will focus on just one process: the *document analyzer*. This tends to be the process documents spend the majority of their time in, as it performs the majority of the computationally intensive natural language processing used to identify risk.

In addition, some of the models developed will make use of documents that have already been analysed at least once. This is since detailed information about the documents, which will be used as input features to these prediction models, is only known once they have been analysed. Hence, some of the prediction models can only be used for predicting the time required for a re-analysis job rather than the more general purpose of estimating the time required for *any* input to the system.

Also, while this project will rely on machine learning based predictions as a tool to analyse the system and to investigate which features seem to have an impact on processing times, the focus will not be on creating elaborate and complicated neural networks and the actual implementation details will only be briefly presented in an appendix. While spending time optimizing the prediction models and fine-tuning parameters would likely yield better results, this thesis focuses on obtaining an overview of how well different approaches perform. The obtained results can serve as a guideline for which approaches provide the best prediction accuracy, and further work can then investigate these in more detail.

1.4 Report structure

In the next chapter, an overview of related research and its relevance to this project is presented. The subsequent Theory-chapter presents an overview of the system used at RF as well as brief introductions to various concepts used throughout the project. In Methods, an outline of how the work was conducted, and why those approaches were chosen is given. This is followed by the results from the investigations. The report is concluded with a discussion and final remarks including future work.

2
Related Work

Another thesis was done in collaboration with RF in 2018 by S. Westlund and S. Strandberg [15]. They tried to predict the size of document queues across RF's system, using a recurrent neural network trained on historical data from various metrics related to the queues. The intention was to use these predictions for preemptive provisioning of the computing resources. If the model could predict when queues would grow beyond a certain size, more servers could be started to deal with the imminent spike in workload. Their results were promising in that historical data could indeed give an indication of how the queue size would change over time. However, the prediction model output too many false positives, resulting in expensive resource over-provisioning, to be put into use by RF. The conclusions from their work is one of the reasons the scope of this project was narrowed to focus on just one process.

A previous study done in 2011 by N. Roy, A. Dubey, and A. Gokhale [16] involved predicting workload in a similar environment as the one used at RF. It was managed in the Amazon Elastic Cloud Compute system and hosted the FIFA 1998 soccer world cup website. They built a resource allocation algorithm which predicted the upcoming workload as a function of the number of connected clients/customers. Their results show that this approach can predict workload and benefit a cloud user when looking at the workload as a function of some input that affects it. However, they approach the problem by using model predictive control techniques. Our models do not rely on control theory, and instead we have opted to use recurrent neural networks for our time-series predictions, due to the available support from machine learning professionals at RF.

S. Venkataraman et al. [17] built Ernest, a framework for performance prediction for Large-Scale Advanced Analytics. Much like in this thesis, their goal was to accurately predict the performance of a given analytics job. They approached the problem by finding relationships between input variables and the target output variable. They use a system modeling approach where they built what they call a "high-level end-to-end model for advanced analytics jobs". Furthermore, they focused on minimizing the amount of training data needed, having similar challenges of accessing data as is the case in this project. While they focus on predicting processing times on a specified hardware configuration, our predictions cover the more generic case in order to handle more variable hardware configurations.

In their study "Analysis, Modeling and Simulation of Workload Patterns in a Large-

Scale Utility Cloud" [18], S. I. Moreno et al. investigated workload characteristics in a cloud production environment. Their work included identification of model parameters for simulating workload, as well as the quantification of behavioural patterns for users and tasks. In their own words: "Tasks are defined as the basic unit of computation assigned or performed in the Cloud, and a user is defined as the actor responsible for creating and configuring the volume of tasks to be computed". In this case, task could translate to a re-analysis job, and a user would be the team scheduling these jobs. Although there will be no focus on the user behaviour part, some important and relevant conclusions were still drawn here, which can be used when approaching some problems in this thesis. For example, they state that "Describing Cloud analyses is an important first step, but providing the parameters and characteristics derived from these analyses is critical.". Thus, a great deal of focus will be on identifying these characteristics and finding the suitable parameters in order to accurately describe the system's behaviour.

T.P Pham et. al developed an approach for workflow task execution time prediction based on a two-stage machine learning approach. Here, the first stage would use historical data about the task to estimate the running time. The second stage combines the outcome from the first stage with the specifics of the considered task as well as information about the Virtual Machine performing the operation [19]. Their work has many similarities with this project, with several different techniques being combined in order to achieve a better task execution time prediction accuracy. The resulting machine learning model was tested using workflows produced by four different applications - an image generator, a tool for performing computations on solid materials, a computer graphics ray-tracer simulator, and a 3D-suite for modeling. Their results are very promising, obtaining an average prediction error of 10%. However, none of the applications tested were focused on performing data analysis or text processing. In addition, the university-developed tool ASKALON was used to to simulate the cloud computing system the tasks were run in [20]. The behaviour of that system may have been more controlled and theoretical in nature compared to the system used at RF, consisting of hundreds or even thousands of servers, billions of database accesses, and the general problems faced in a large-scale real-world system [21].

A study on estimating execution time of workflows in the cloud was conducted by I. Pietri et al. [22]. The goal of this project was to accurately predict the makespan for a given number of resources. Their model took into account the structure and characteristics of a task related to a scientific workflow in order to estimate the execution time. They argue that their model is able to reach good prediction accuracy. However, they did not consider any runtime metrics for prediction, something that can be used in combination with input characteristics when predicting execution time.

3

Theory

This chapter describes the background information necessary to understand the work carried out during this project. First, the threat intelligence product and its underlying analysis system used at RF will be presented. This part is concluded with a summary, for quick reference. Then, information about various theoretical concepts used when developing the prediction models will be presented.

3.1 Overview of the analysis

In this section, an overview of this project's application specific information - RF's document analyser and its underlying system architecture - will be presented. It is important to have a basic understanding of what the input to the system looks like, what happens during an analysis of a document, and how the underlying architecture is designed, to better understand the choices made in the methodology.

3.1.1 The Document

A *document* is a term RF uses for the input information that is fed into their system for analysis. At its core, a document is some kind of information available from the web that contains text. Forum posts, tweets, PDF's, and code repositories are all different types of documents.

3.1.1.1 Entities and references

When text is analyzed in the system, one of the goals is to aggregate information from this text into *entities* and *references*. This is done in order to make sense of what the text is actually dealing with. An *entity* can be inferred to as a noun, which for example can be a person or a company. A *reference* is a more generic term for any information the analysis process determines is interesting - a sentence, a timestamp, or a noteworthy event mentioned in the document.

Figure 3.1: A typical example of a document - a tweet

Figure 3.2: An example of what the analysis process might extract from the previously mentioned tweet. Here, it is also shown how the *entity* Elon Musk is also considered an *author*, another type of aggregated information, by the system.

3.1.1.2 Document metadata

Once a document has been analysed, aggregated information about what the document analyser found is stored in a database. Some examples of what this information can include are:

- What *language* the document was written in
- When the document was posted on the web
- Which entities were found
- Which references were found, and why they were considered interesting

- Who the author of the document was - for example, the Twitter account

It is important to note that depending on the document, the analyser might not be able to identify all of these fields. For example, the document might not contain a sentence the analyser can correctly interpret as a timestamp, in which case the date the document was posted is unknown.
This aggregated information is referred to as the *document metadata*, and it will be used extensively in this project.

3.1.2 The analysis system

RF's system consists of multiple processes, each responsible for one part of the analysis. For brevity, only the process studied in this project - the *document analyser* - is described in this section. An overview of the most relevant other processes are illustrated in Figure 3.3, and more detailed descriptions of their functionality is available in Appendix A.

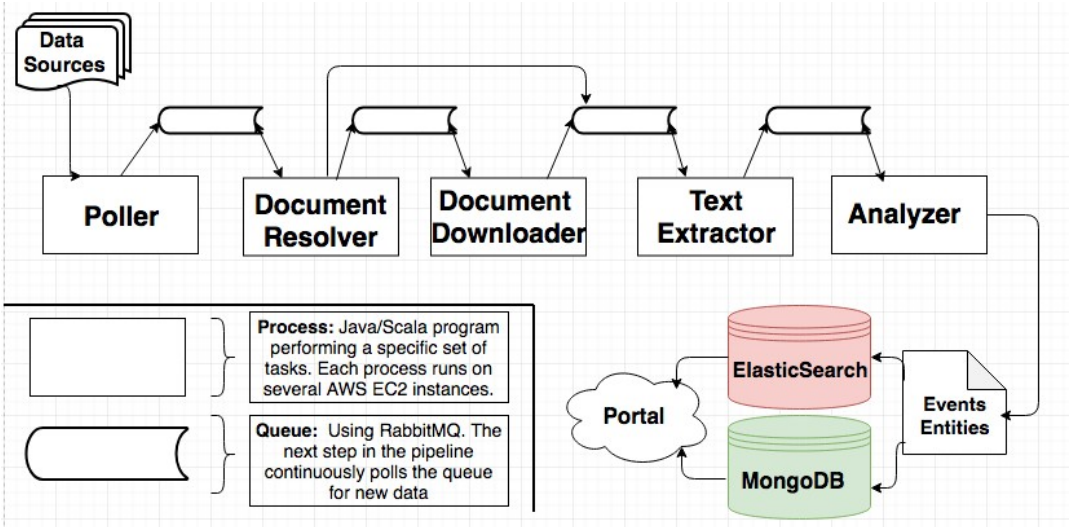

Figure 3.3: An overview of the system. The analyzer process, which is shown to the far right, is the process focused on in this thesis.

3.1.2.1 Document Analyzer

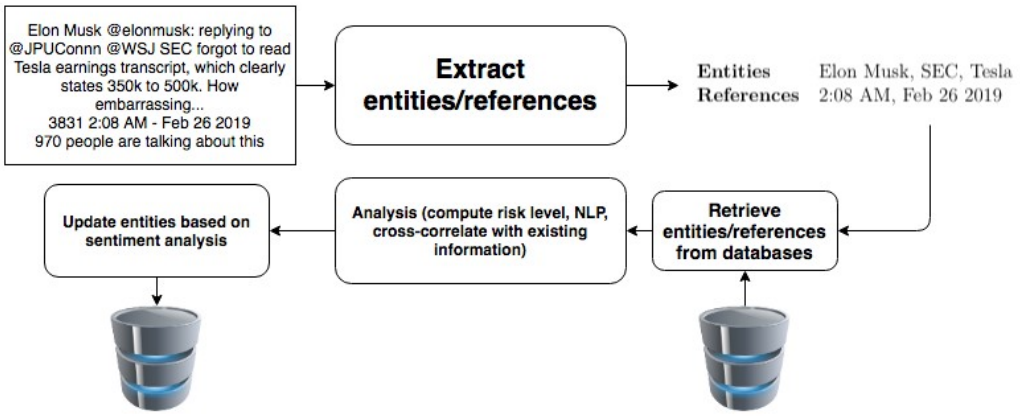

Figure 3.4: An overview of the different steps taken by the document analyzer when processing a document

The document analyzer process performs the main analysis - by identifying entities and references in the extracted text and cross-correlating them with already existing information it is able to draw conclusions about the level of risk associated with those entities. The bulk of this work is done via natural language processing, some of which is developed internally at RF while some is provided by third parties.

Compared to the other processes, the analyzer performs far more computationally intensive operations which is why it was chosen as the focus of this project. It is also the component responsible for the majority of calls to the databases and indexes - other than often need to read multiple entries for each piece of potentially interesting information it finds, the practical implication of a finished analysis is a set of new entities as well as old entities being updated due to the new information discovered.

When an analysis is performed on any given document, information about the time it took for that analysis is stored by the system. Further in this thesis, this will be denoted as the DPT, *Document Processing Time*.

3.1.3 The re-analysis operation

One of the primary reasons RF were interested in developing a task execution model was to be able to make informed decisions about a special type of analysis performed in their system. This is referred to as the *re-analysis*.

A re-analysis job is a manually ordered processing of a large number of documents that already exist in the system. As described in the system overview, documents become *references* and *entities* when they have passed through the full analysis chain. These are then used to calculate the *risk level* associated with certain entities. For

example, if an IP-address (entity) is linked to multiple DDoS-attacks (references) then it could be classified as high-risk. This is then reflected in the portal, which is what the end users see. Thus, references and entities are intricately linked meaning new information about one entity could possibly make the analysis conclude that another entity should have a higher risk level.

Also, new and improved functionality is continuously added to the product meaning new and more interesting information can be gathered from analyzed documents. Because of this, re-analysis jobs are run almost constantly as they may in some cases provide just as much relevant information for RF's clients as analyzing new documents.

Other than the documents being known in advance - a property that will be used extensively throughout this project - an additional difference from conventional documents is that re-analysis jobs are put on a special queue before they are allowed to enter the analysis pipeline. This queue is throttled, meaning that if the system is currently under too much stress no re-analysis documents are put through. In this sense, re-analysis jobs are deprioritized.

The ability to determine prior to analysis how long one of these jobs would take to run is highly valuable to RF. The longer a job runs for, the more computational resources are used, and the more RF has to pay in server costs.

3.2 The distributed architecture

Each day, between 10 to 50 million new documents enter the system for analysis. Around 15 billion database lookups are performed when extracting and updating the identified entities in these. In addition, hundreds of re-analysis jobs are ordered, adding even more workload.
To deal with this load, all of RF's processes, including the document analyser, are run in a distributed system. As described in the introduction of this report, this introduces multiple challenges when trying to examine the time required for a certain task, and it has some implications for what a prediction model is able to consider as input features.

The document analyser is run on 300 to 500 servers, depending on the current workload. RF does not have its own infrastructure of machines. Instead, it rents computing capacity from Amazon Web Services using their *Elastic Compute Cloud* (EC2). A plethora of server types are available, depending on what is required of the application, as well as system monitoring tools that can be useful when running diagnostics. It should be noted that unless a dedicated host is used, any single EC2 instance (server) can run software from multiple clients simultaneously. Hence, it is more accurate to think of it as renting pure computing capacity rather than physical servers the client has full control over.

3.2.1 Automatic scaling

One of EC2's main strengths is the flexibility provided by *auto scaling*. This allows for the dynamic start and stop of individual instances while the service is running. Users can design *auto scaling groups* where they define the minimum capacity that should always be available to the system as well as the maximum number of machines the group is allowed to scale up to.

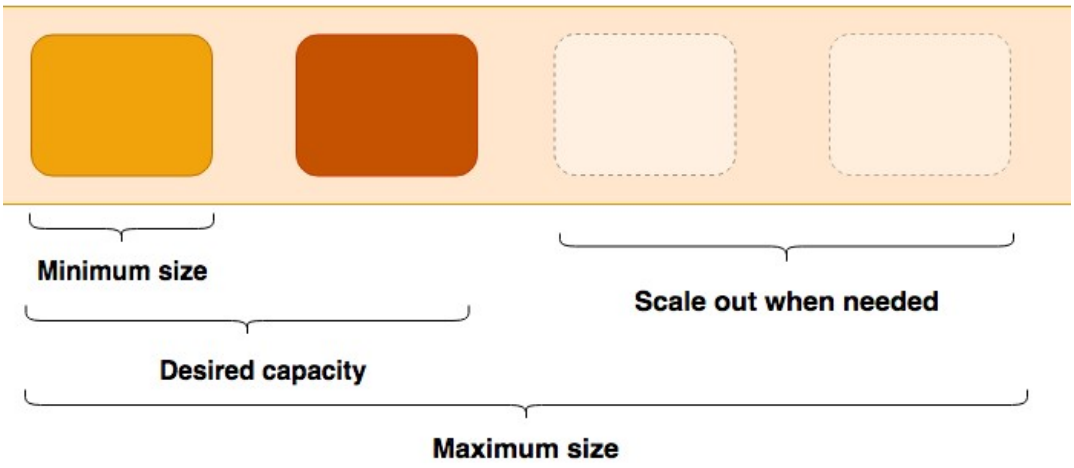

Figure 3.5: Simple illustration of an auto-scaling group in AWS EC2

The user then creates *auto scaling policies* that dictate what should trigger a scale-up or scale-down, and how much capacity should be added or removed. It is possible to scale based on internal metrics in EC2 itself - for example, keeping track of the CPU usage across running instances and triggering a scale up when it rises over a certain threshold. It should be noted that these newly started instances do not become available immediately, but require some time before they are able to start performing work. This means that there can be large build-ups in the queue size during this initiation phase. The additional instances can then be stopped once the usage goes below the queue size limit for a sufficiently long period of time, to reduce costs.

Another option is to use the **CloudWatch API**. With this, users can base the scaling on external metrics. For example, RF continuously keeps track of the length of RabbitMQ-queues in the system. Using a monitoring tool, an alarm is triggered if the number of messages in a queue preceding a certain process becomes too large. If this happens, an alert is sent to CloudWatch, which in turn starts additional instances according to what is specified in the corresponding policy.

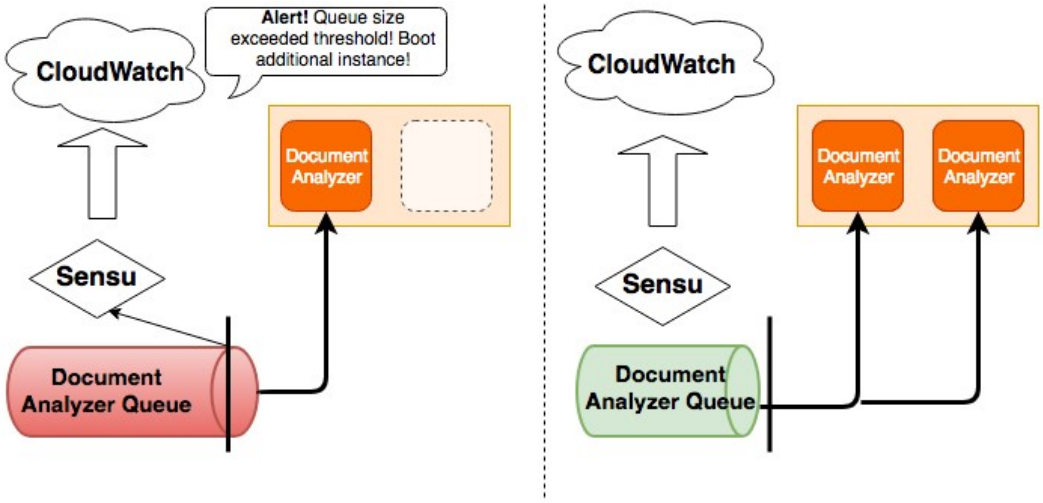

Figure 3.6: Example of a scale-up triggered by a custom event. Once the new instance has booted, the queue size decreases as messages are now being processed twice as fast.

3.2.1.1 Task-execution time prediction and its relation to automatic scaling

The processing time required for a task can be equated to the amount of computation required to perform that task - the longer an operation runs for, the longer it occupies computing resources. Thus, if the required processing time for a task can be predicted, then it is possible to forecast how much computational load it will incur on the system.
In a system using automatic scaling, the ability to predict the time required for a task can alleviate the problem of newly started instances taking a while to start up, leading to increasing queue sizes. The scaling can be based on the predicted time required for the tasks entering the system in combination with the current amount of available resources, ensuring additional instances are started and available should large increases in incoming workload occur.
Scale-down triggers can be set up in the same way, notifying CloudWatch to stop instances once some metric indicates the provided capacity is more than enough to handle current workload. It is also possible to make scale-downs conditional - for example, requests to stop instances may be ignored if the CPU usage across all instances is high. This aids in preventing erroneously configured alerting rules from causing the system to go into a loop of scale-downs followed by immediate scale-ups.

3.2.2 The java virtual machine

The document analyzer process is written in Java/Scala and consequently runs in the Java Virtual Machine (JVM) [27]. A useful side-effect of this is the ability to conduct performance monitoring on a per-JVM basis. As there is one JVM running per EC2 instance, combining the metrics from the JVM with the EC2-instance

metrics available from CloudWatch it is possible to get a detailed view of how a given document analyzer is performing during any interval at any time.

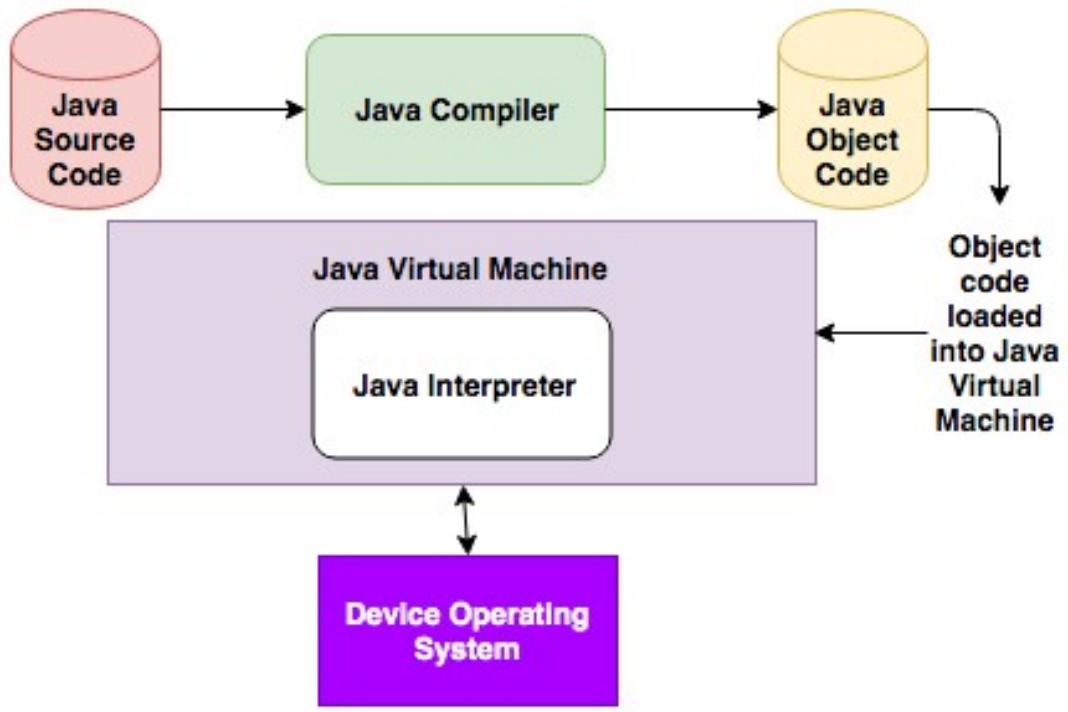

Figure 3.7: An overview of how Java code is compiled and run on a computer

3.2.2.1 Memory usage

Various metrics related to how the memory is used are available. For a full list of these, see Appendix A. Running out of memory can force a process to use the hard disks for short-term storage resulting in a massive performance drop or to crash. Hence, investigating the memory usage can be highly relevant when determining why a process might be performing poorly.

3.2.2.2 Thread count

A thread created in Java maps directly to an operating system-level thread. In the case of the document analyzer, up to four or eight threads depending on instance type run concurrently and each thread is configured to analyze one document at a time. While the number of active threads should always be four or eight, deviations from this value could indicate that something is wrong with the instance.

3.2.2.3 Garbage collection

As opposed to languages with manual memory management like C - where the programmers explicitly has to assign memory to new variables and then return it at some suitable time - in Java the JVM automatically frees up unused memory. This is known as *garbage collection*, and depending on which algorithm is used for this the

performance of the application can be affected to various degrees [28]. Therefore, considering when the JVM was performing garbage collection could be helpful when studying variations in document processing time.

3.3 System performance metrics

RF maintains many metrics related to the performance of the document analyser, and of the surrounding infrastructure. In addition, users of AWS can access detailed logs of the performance of their rented instances. Many of these metrics will be used as input features for the prediction model.

3.3.1 Internal system performance metrics

RF collects various metrics about their system from different sources, which can be anything from RabbitMQ-logs to collecting interesting information in the code of the processes themselves. These are then all sent to Graphite [29] for storage, and then become available for download and further analysis using an API. Graphite is also able to provide simple graphs of the metrics, and it supports integration with a visualization tool called Grafana [30]. RF stores thousands of metrics in Graphite, for example:

- **Queue information** - Number of messages, egress and ingress rate (incoming/outgoing rate of messages to/from queue), and number of consumers (threads listening) of each queue.
- **Monitoring logs** - Contains information about when various types of auto scaling was triggered.
- **Java virtual machine metrics** - Information collection from each java virtual machine such as memory usage, virtual thread count, and garbage collection logs.

3.3.2 CloudWatch - EC2 instance performance metrics

CloudWatch is a service offered by AWS [31]. CloudWatch provides real-time monitoring of the EC2 infrastructure rented by any customer. The CloudWatch Agent can be run on each instance in the cloud, aggregating metrics such as CPU utilization, network traffic and disk read/write rates. Metrics from individual EC2-instances are available as well as aggregated metrics across auto-scaling groups, providing both detailed information as well as an overview of how a specific process is performing - at RF, each auto-scaling group is responsible for one process in the analysis.

3.3.3 The Checkpoint Database

At RF, checkpoints are saved in a so called *Checkpoint Database* for each document passing through the system. These checkpoints are data explaining which process the document passed though, at what date and time it entered and how long it spent there. If the document belonged to a re-analysis job, the ID of that job is also stored

in each checkpoint. These checkpoints can be of use in many cases, for example when finding processing time for a document, investigating causes of delays, finding bottlenecks, following the path of a document or finding all documents related to one specific re-analysis job.

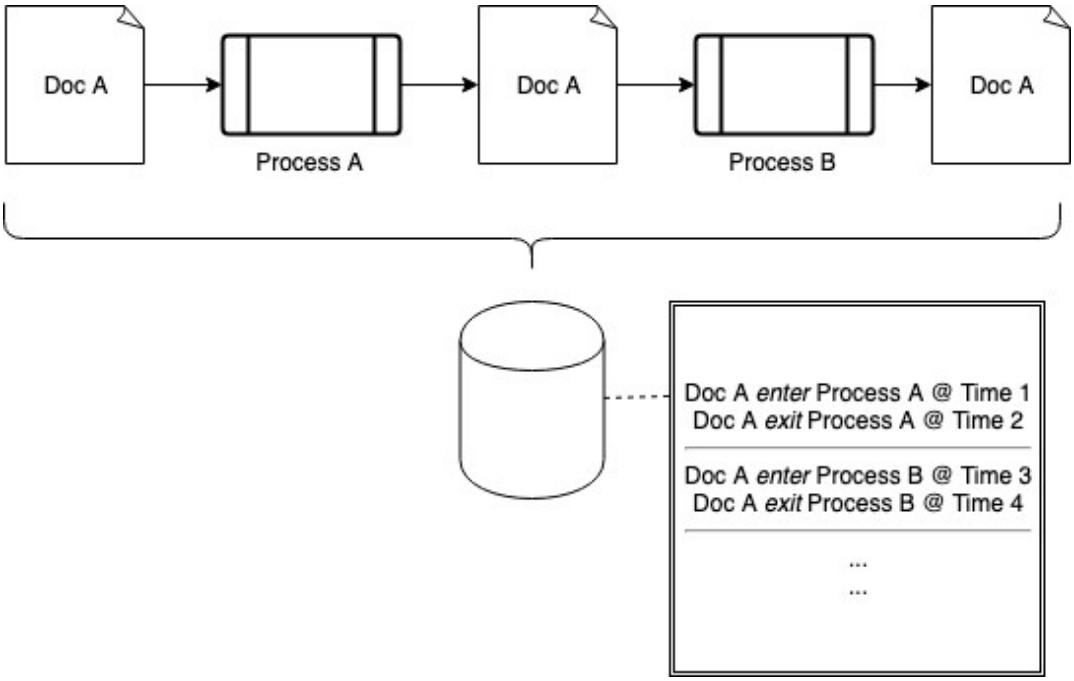

Figure 3.8: An overview of how the checkpoints for documents passing through the analysis system are stored

This information is highly useful for two reasons. For one, it enables easy detection of potentially interesting documents by investigating jobs that either timed out or spent a significant amount of time in a certain process. Also, aggregating all available information about a particular job can give insight into how the documents in that job passed through the system, which in turn provides a relatively simple way of comparing different re-analysis jobs and looking for patterns.

Last but not least, the checkpoint database is crucial for this project as the DPT found in each entry is the target variable for the prediction.

3.4 Summary of the system

To conclude, the most relevant application specific background information presented in the previous sections will now be summarized.

3.4.1 Analysis summary

RF has a system used to analyse data. The input is called *documents* and consists of text. Various forms of pre-processing are applied to the documents before they arrive at the main *document analyzer* process. The analyser extracts interesting information - *entities* and *references* - from the documents. It then uses a variety of techniques to perform a sentiment analysis - a *risk score* is computed and assigned to each entity. This information is then made available to RF's clients through an API and a web interface.

3.4.2 Infrastructure summary

The document analyzers are Java/Scala-processes that are deployed on the Amazon EC2 Cloud Computing platform. The system is distributed, and the number of active servers (also called *instances*) vary depending on current system workload. Each active server runs one document analyzer process. The processes are multi-threaded, and configured such that each thread analyses one document at a time. While doing this, it performs multiple database writes and reads, to retrieve and update information related to the identified entities. Once a thread is finished, it retrieves a new document from the *document queue*. If the size of this queue exceeds a certain threshold, additional servers are started to deal with the increased load.

3.4.3 Metrics summary

The length of time a document spends in the document analyser is logged in the *checkpoint database*. Information about the document metadata such as the identified entities, references, and language is also available. A plethora of information about the systems performance can be found in *Graphite* and *CloudWatch*.

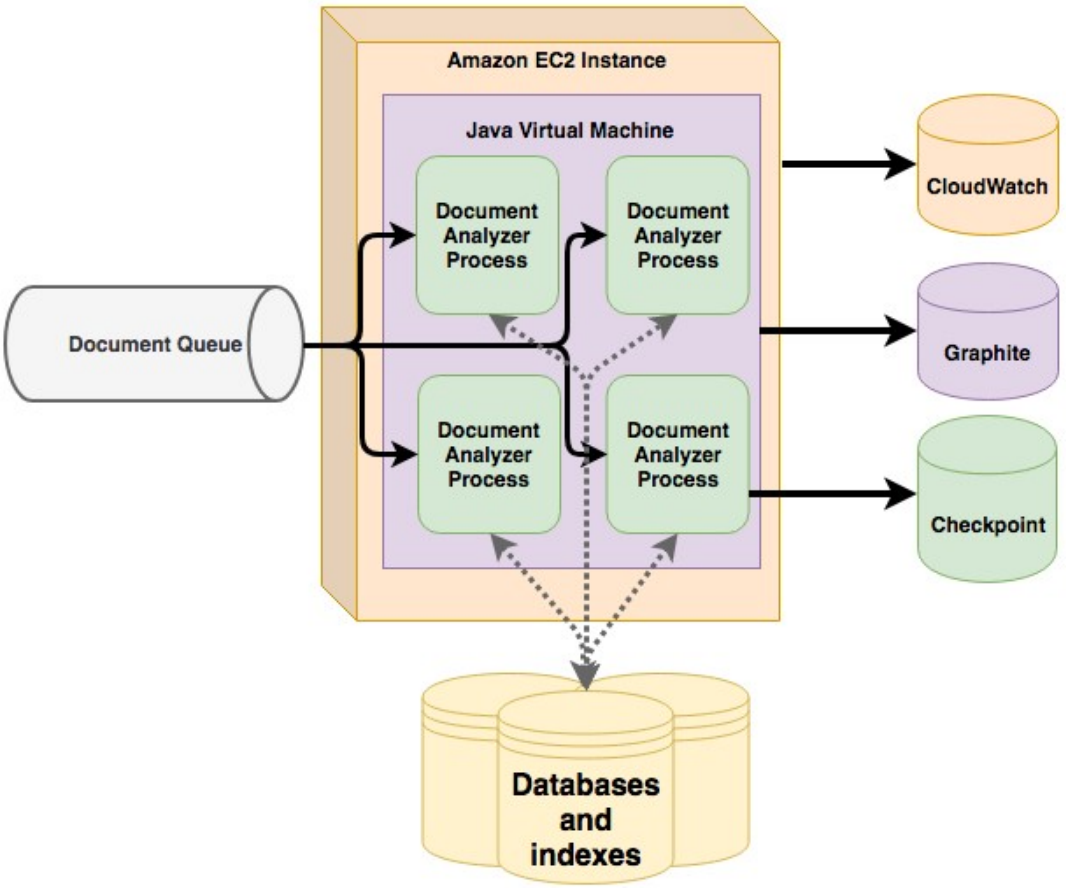

Figure 3.9: An overview of the architecture used for the analyzer process, as well as where the metrics related to the different components are stored

3.5 Neural Networks

In this project, all of the prediction models will be based on neural networks. A Neural Network is a computing system with a graph-like design. The network is structured in layers of nodes and connections between the layers, resembling a simplified version of the neurons in a brain. Each node can forward input values through connections to other nodes, and these connections corresponds to weights which multiply the input by a factor. A Neural Network can have a different amount of layers, with a different amount of neurons in each layer, often depending on the specific application. The first layer is called the *Input Layer* and the last layer is called the *Output Layer*. All layers in between the Input and the Output Layers are called *Hidden Layers*.

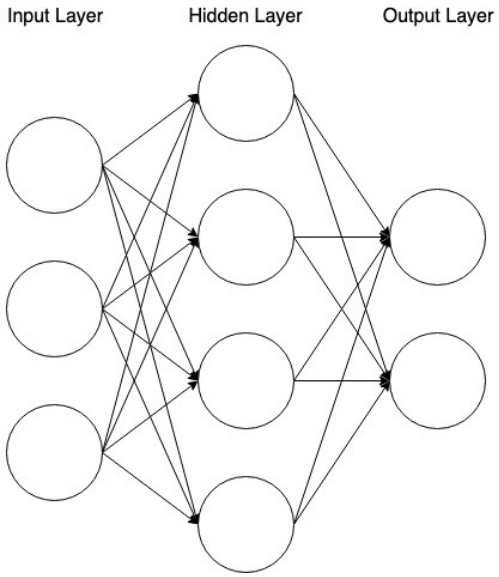

Figure 3.10: A basic structure of a Feedforward Neural Network, having one hidden layer.

The decision to use neural networks was made due to their ability to identify complex and non-linear patterns in input data, unlike more basic approaches like linear regression. Since many of the input features considered for the different prediction models are non-linear in nature, such as the metrics about system performance, it was hypothesized that neural networks would be suitable. Neural networks can learn which of the potentially many input parameters are of importance, using a technique called backpropagation. Backpropagation lets the network propagate errors back through the different layers and adjust the weights based on the error.

In supervised learning, labels with the actual values are needed together with the input in order to train a network. These labels are used as a reference when calculating the error in each step of the training. Thus, having high quality input data for training can increase the chances of having accurate predictions when the network looks at completely new data later on.

3.5.1 Feedforward neural networks

A Feedforward Neural Network (FNN) is a type of neural network where no cycles are formed within the connections between the neurons. It is the most basic structure for a neural network where the flow is in one direction only. For example, the illustration in Figure 3.10 denominates an FNN. This type of network is what will be used in most of the models in this thesis.

3.5.2 Recurrent neural networks

Some of the models evaluated in this thesis will make use of a Recurrent Neural Network (RNN), which is a type of neural network where each neuron has an internal

memory that can store past values. This enables it to take into account not only the current set of input but also previous data when making decisions, making it aware of temporal relations in the input. Because of this ability RNN's are frequently used for sequence prediction problems, where previous values can have large implications of how current values should be interpreted. A classic example is translation - words can have different meanings depending on which context (sentence) they are used in.

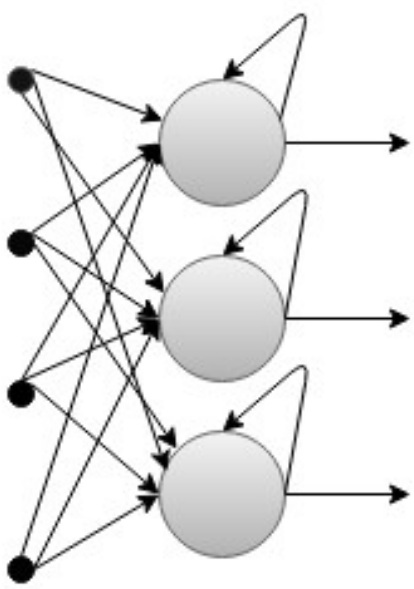

Figure 3.11: An example of a simple RNN. The arrows originating and ending in the same neuron represent the networks ability to use internal state (memory) as input

3.5.3 Evaluating prediction performance

The Mean Absolute Error (MAE) was chosen as the error metric. The primary reason for this being ease of interpretation; it measures the average distance between the predicted and the actual values. For instance, in this project a MAE of e means that on average, the models predictions are e milliseconds off the actual document processing time.

3.5.3.1 Mathematical definition

The MAE is given by:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

for a series of n points, where y_i denotes the predicted value for point i, and x_i denotes the actual value of point i [32].

3.6 Time Series Prediction

The concept of time series prediction suggests that, by looking for trends and patterns in historical data related to some interesting variable, it can be possible to predict future values of said variable. For instance, it might be possible to forecast the price of a share of a company after a certain time by looking at how the price of that share has varied over time up until that point. Time series prediction differs from other prediction models in that, unless manually used as an input feature, there is usually no temporal relationship between the data points in the latter case. That is, standard models do not consider the *order* of the data points when computing their predicted values. This property means that more care has to be taken when constructing the data sets - for example, randomly sampling data points means there is a high chance many of them will not end up in the correct order.

A benefit of using time series prediction is the ability to treat the problem as a black box. For instance, there are a myriad of variables that may or may not affect the price of a share in the stock market, and modelling this is incredibly hard. However, if the future price can be estimated by studying how the price has varied over the last few days, weeks, or months, the problem of modelling the problem is circumvented.

4
Methods

In this chapter, the methodology used throughout this work is presented. An overall view of the different approaches used when trying to predict the processing times is given at first. Subsequently, each approach will have its own section, detailing that particular model.

4.1 Project plan

The aim of the project is to evaluate how well a prediction model is able to predict processing time when trained on different input features. Five models have been developed, each considering a different set of input features related to the following:

- M1 - Document (input) metadata
- M2 - Document metadata, combined with various metrics related to system performance at the time of analysis
- M3 - Considering metadata from all documents processed concurrently in the same document analyser
- M4 - Time-series analysis, where future processing time is predicted based on trends and patterns in previous processing times. This approach is the only one to use a recurrent neural network, as a RNN is able to determine temporal relationships between a set of consecutive data points.
- M5 - Combining document metadata with previous processing times. Unlike the time-series analysis, this will use the standard feed-forward neural network. Each data point in this set includes previous processing times as input features, rather than having the data set consist of a consecutive series of document analyses.

The models are presented in order of increasing implementation complexity. Higher-numbered models do not necessarily consider a greater amount of input features than prior ones, but they may require information from additional sources for their predictions, or the input features themselves may be harder to compute.

An additional approach is also going to be evaluated, where documents will be batched together to form jobs. These will then be used to investigate how the model that only considers document metadata performs when it is trained and tested on very large sets of input. It is however not considered a separate model, as the input features are the same as in the model that considers only document metadata.

4.2 Retrieving data for the different models

As the models focus on different aspects, and in turn consider different input features, the data used for training and testing each model differs in a variety of ways. Not only in terms of what the data itself represents, but also the quantity of available data, where the data is stored, and how difficult it is to obtain.

As a consequence of this, some models will have access to far more data than other ones. This is primarily because of restrictions in how much data can be retrieved from the corresponding sources without causing performance problems in the production environment, as well as some sources occasionally missing data due to collection errors.
The most important implication of this is that all of the models will use *different data sets* for training and validation. This means that the comparisons made between the models are inherently unfair, and the prediction results of the different approaches have to be considered with this in mind. The approach taken to be able to compare the models is described in section 4.3.
Appendix C contains a detailed description of the data retrieval process on a per-model basis.

4.3 Evaluation methodology

The Mean Absolute Error (MAE) was chosen as the metric used to measure accuracy. Since different sets of data were used throughout the project, the error with respect to the mean (and median) value was computed (MAE / mean), allowing for easier comparison of results between the different models. In order to evaluate each model individually, a naive prediction algorithm was used. This algorithm took the mean (and median) value of the data set used, and consistently predicted that same value for each point. The MAE and MAE / mean was used to compare the naive prediction algorithm with the trained model. An illustration of the actual DPT and predicted DPT, both by the trained model and the naive approach, is shown in Figure 4.1.

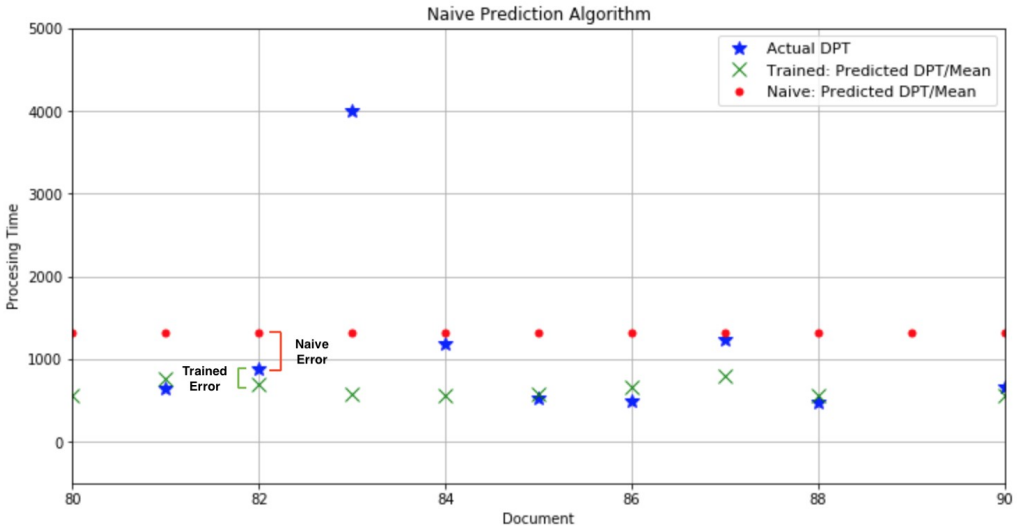

Figure 4.1: A plot of 10 documents and their corresponding DPT, together with a trained model's predictions and naive model's predictions. Values of y-axis are in milliseconds. In this plot, the naive error and trained error is compared for document number 82.

4.4 Model one - Predictions based on document metadata

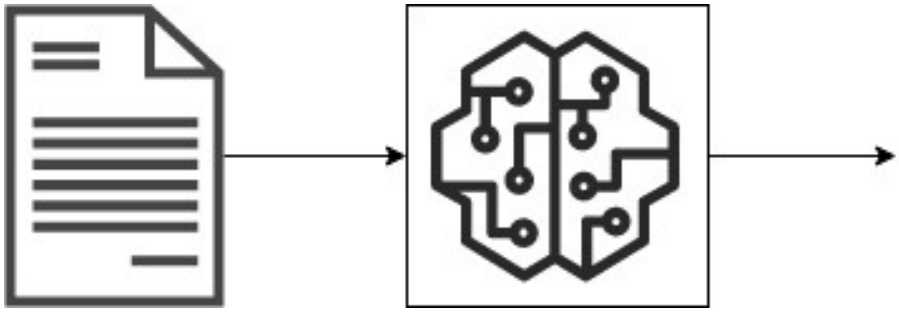

Figure 4.2: Input characteristics are fed into the model for DPT prediction

As a reminder, *metadata* refers to document information found by the analyser process when that particular document is used as input. Developing a model that only considers features related to the actual input to the system is desirable for a number of reasons:

- Firstly, it serves as a good base for the more advanced models discussed in this report.
- A model that only needs information about the input is relatively simple to implement and actually use in a real environment compared to other models.

For example, other models that make use of performance metrics or historical data have added complexity because of the need to aggregate and arrange data from other sources.
- Identifying which input features that seem to have the largest effect on document processing time is helpful to RF, as it might be possible to resolve these bottlenecks.

Since there are a huge number of possible features to look at when considering document metadata and most of these are specific to RF's product, as a first step the team responsible for the document analyser process was consulted. Based on their advice, the majority of these features were discarded, with 11 candidate features remaining. These are detailed in Appendix B.

A large number of documents was retrieved from RF's system and the candidate features were extracted from each document. The model was trained and tested in an iterative process where one candidate feature was removed at a time and the models prediction error was evaluated.

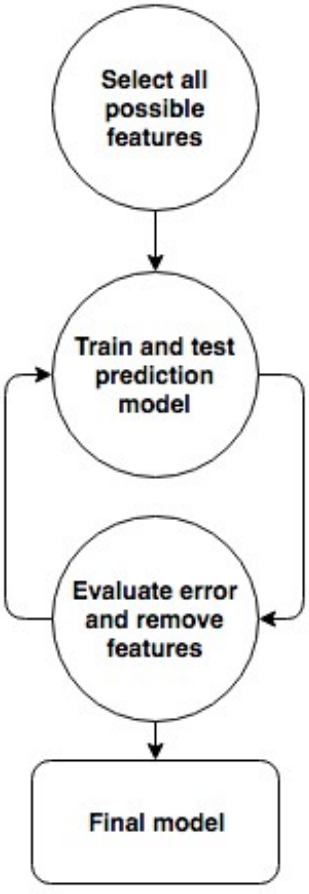

Figure 4.3: Illustration of the iterative approach taken when performing feature selection for model one.

The results of this iterative process - which features proved to be most important when trying to predict processing time - and the final models prediction results are available in Section 5.

4.5 Model two - Using system performance

One hypothesis that was formed at the onset of the project was that just using the content of the document might yield usable results, but likely not optimal ones. The primary reason for this would be the complex nature of the system used. In an ideal setting, the environment would be far more stable and configurable depending on the needs of the current investigation, but as noted earlier in this report a large part of the project's contribution is the fact that the system studied is used in a production environment. Intuitively, if the system used to perform a computation is currently performing poorly for some reason then any operation performed on it will take longer to complete. Indeed, anyone that has used a computer or smartphone is probably familiar with what is usually called "lag" where actions that usually complete instantly suddenly take significantly longer, or end up freezing the users device.

With this as a motivation for creating a more complex model, an investigation into how the current performance of the system affected document processing times was conducted.

4.5.1 Collecting the metrics and engineering input data

All metrics over system performance that was available was collected by constructing a series of programming scripts that would:

- Take an entry from the Checkpoint-database as input
- Using the timestamp and resulting processing time found in this entry, identify the time interval that document was analysed during
- Fetch the interesting metrics during this interval from Graphite and Amazon CloudWatch

All data points available during these intervals were used to create a data set for this model. As a start, the full set of performance metrics were stored before the process of reducing them to a sufficient amount.

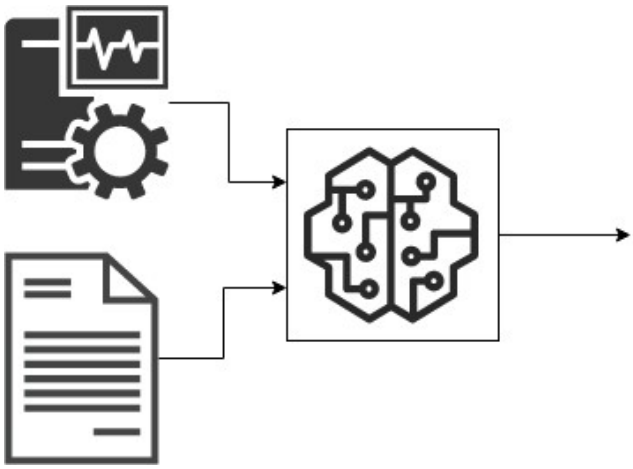

Figure 4.4: A combination of input characteristics and machine performance metrics was used as input to the model

4.5.2 Feature selection and optimization

Unfortunately, there exists not single metric which could accurately describe the health and behaviour of a large distributed system. Therefore, all system performance metrics available were initially considered. In order to exclude metrics that are not related to affecting the DPT, the process of feature selection, illustrated in Figure 4.3, was used again. Furthermore, advice from RF's engineers was used to decide which metrics should be considered at first. All metrics involved are detailed in Appendix B.

As mentioned in section 4.4, the feature selection process included training a model, in this case an FNN, and evaluating the error after each training session. Features were removed from the input set and if the error did not increase, these features where no longer considered in future training sessions.

4.6 Model three - Considering documents analysed in parallel

A metric related to system performance, such as the CPU load of a server, depends on what that server is currently doing. Hence, looking at the metrics can reveal *how* the system is behaving but not *why* it's currently behaving in that way. Due to the multitude of interconnected processes, databases, indexes, and queues used in a distributed system of this size it can become very hard if not impossible to isolate which individual components - and by extension which system metrics - are responsible for fluctuating processing time. However, by instead considering the causes of increased system load - the input - it might be possible to avoid this problem altogether.

The document analyzer process is multi-threaded - each thread processes one document at a time, and fetches a new document from the queue upon completion. Each EC2-instance has four or eight virtual cores depending on server type, meaning that for any analysis of a document up to eight other documents are being processed simultaneously on the same EC2-instance.

The following approach was taken when conducting this new investigation:

- Take an analysis entry from the Checkpoint-database
- That document was then called the *target document* and its corresponding processing time the *target processing time*
- Fetch metadata for all documents that were processed in the same interval and on the same EC2-instance as the target document
- To avoid skewed data, for each of these other documents the number of references, entities, and text length was multiplied with how much that document overlapped with the target document. For example, if a document running for 20 minutes only overlapped for 1 minute, the majority of that documents data was assumed to be processed before the target document
- Then, the "weighted" metadata for each other document was summed up giving a total number of references, entities, and text length.

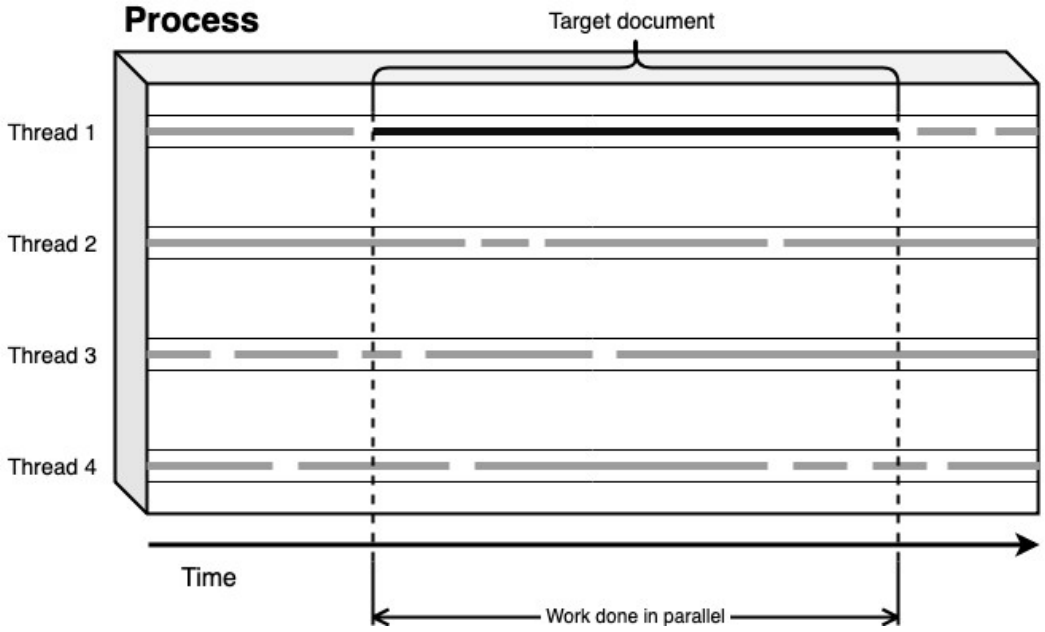

Figure 4.5: Documents being processed sequentially within each thread, and in parallel between threads

Rather than use all available metadata, only the input features that provided the best performance in model one were chosen: the document text length, the number of entities, and the number of references.

Since a single document can be processed inside the document analyzer process for anything from just a few milliseconds up to half an hour, there could potentially be thousands of other documents being processed in parallel during this interval. Due to the limitations, mentioned early in this report, about the system's sensitivity to extracting too much data, there had to be limits in the amount of data points created for a data set used for this model.

As in the previous cases, an FNN was trained and tested on the new data. One data point used as input for the model consisted of the following features:

- Target document number of references
- Target document number of entities
- Target document text length
- Number of references, total
- Number of entities, total
- Text length, total
- Target document processing time

4.7 Model four - Time series prediction using previous document processing times

A series of consecutive documents was retrieved from the Checkpoint-database. A recurrent neural network (RNN) was trained to predict the processing time of a document based on the processing times for a varying number of previous documents. As a follow-up, the processing time was divided with the corresponding document text length, giving a "normalized" time. Three series of varying length, from different EC2-instances and during different intervals, were studied.

As each machine has multiple threads, many of the documents considered in a time-series may overlap. To be consistent when creating data sets for a time-series, the timestamp of which any document was considered *finished* by the document analyzer process was used in the consecutive ordering. Thus, any document i would be considered before document j if the finished timestamp of i is smaller than that of j, regardless of whether i or j entered the process first.

This was followed by a time-series investigation using average prediction times across *all* document analyzers (EC2-instances). A series of one- and ten-minute intervals and the corresponding average DPT was retrieved. Again, an RNN was trained but this time its task was to try and predict the *average* processing time during the next interval, based on previous averages. The purpose of this investigation was twofold:

- Firstly, finding out how much the processing time fluctuates across the entire system provides some interesting information in and of itself.
- Second, which instance (document analyzer) a given document is processed in is unavailable right up until that analyzer fetches it from the queue. Thus, a

practical implementation of this model would not have access to previous document processing times as described above. However, the *average* processing time of the past minutes/hours across the entire system is available.

A more formal definition of time series prediction in this case is that for a given lookback length l, and a set of DPTs, look at the previous l DPTs in order to predict the upcoming DPT using some function, f, computed by the neural network. This was done considering the DPT and the normalized DPT of documents on the same instance, as well as the average DPT across all instances of the system.

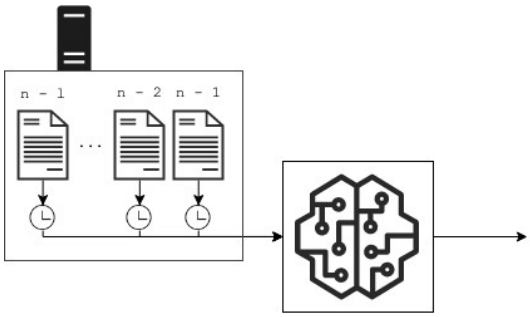

Figure 4.6: Using a time-series of documents being processed on the same instance

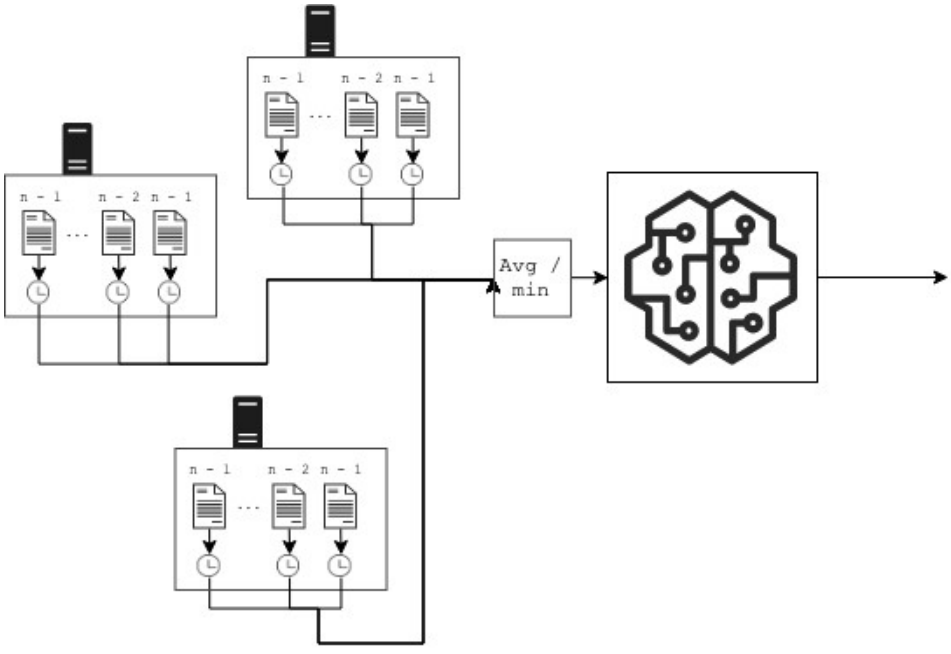

Figure 4.7: Using a time-series of documents being processed across the whole system

4.7.1 Per-instance

Let $i = 1, 2..n$ denote a series of documents analyzed consecutively on the same document analyzer/EC2-instance, and let T_i be the processing time required for document i. The time required for the next document, $i + 1$, can then be predicted with:

$$T_{i+1} = f(T_i, T_{i-1}, ..T_{i-l})$$

where l is the value of the look-back parameter - how many previous analysis times the model should consider when making predictions about the next one.

4.7.2 Per-instance, DPT divided by text length

Let T_i denote the processing time required for document i. Let W_i denote the text length of document i.

$$T_{i+1} = f(T_i/W_i, T_{i-1}/W_{i-1}, ..T_{i-l}/W_{i-l})$$

Dividing processing time by text length results in the speed at which documents are being processed. Hence, this approach tries to predict future processing times based on how fast the analyser was working during the last few seconds or minutes, depending on which lookback is used.

4.7.3 Predicting the average DPT across all instances

Here, let i denote a consecutive series of intervals. Let AD_i denote the average document processing time across all document analyzers for interval i. Assuming $AD_1..AD_i$ are known, AD_{i+1} can then be predicted:

$$AD_{i+1} = f(AD_i, AD_{i-1}, ..AD_{i-l})$$

4.8 Model five - Combining previous processing times with document metadata

This model takes into account the combination of document metadata and previous DPT's within the same machine. Similar to the third model, not all metadata available about documents was used - only the features found to give the best results when predicting only on the documents themselves were chosen. Both the processing time and the processing time divided with document text length were added as features. One data point now consisted of:

1. Document text length
2. Document number of references
3. Document number of entities
4. Processing time of previous document run on that analyzer
5. Average processing time of previous 5, 10, 20, 35 and 50 documents that were ran on that analyzer

6. Average processing time of previous documents, divided by their corresponding text length, giving the time required per character of text.

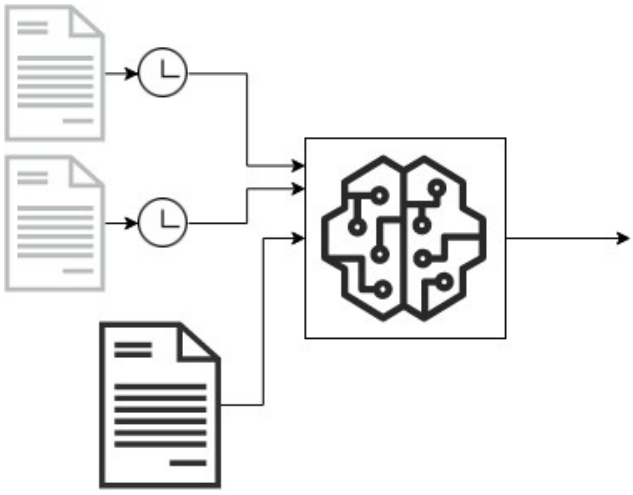

Figure 4.8: Combining input characteristics with previous DPT's as features into the model

4.8.1 Adding system-wide time-series data as feature parameters

As described above, per-instance metrics are unavailable when the document enters the system. Here, the network was trained on selected features from the document metadata and what the average document processing time across all document analyzers was during a number of previous time intervals.

1. Document text length
2. Document number of references
3. Document number of entities
4. Average processing time during previous minute
5. Average processing time during previous 10 minutes

4.9 Predicting the time required for a re-analysis job

Ultimately, being able to accurately predict the processing time for any given document will yield the possibility of adding up prediction times of all individual documents in a given job, indicating the total processing time for that job. On the contrary, if the prediction accuracy is low and consistently undershoots or overshoots, and if a job consists of a large amount of documents, the job duration prediction will also overshoot or undershoot.

Being able to accurately predict processing time for individual documents is desired for several reasons, such as being able to scale the predictions easily and having a detailed understanding of the effects that document metadata have. Nevertheless, stepping up one abstraction level and considering complete jobs could potentially also abstract away some of the features causing larger errors. For example, a document that has been analyzed multiple times and has a somewhat spread out interval of different processing times for each analysis, could potentially have this spread because of uneven resource distribution. It could simply be the case that the OS decides to give the thread, that is analyzing that document, a different amount of resources every time, causing the spread in processing time. The reason for this could be that when the document enters the machine, other threads, processing other documents, are given more or less resources than the last time.

Thus, accurately predicting the processing time for a single document is hard if the resource management by the OS is not completely understood, but it needs not be considered when predicting for a large job with many documents, where the resource distribution is even on average.

4.9.1 Combining documents to form jobs

A job, or a re-analysis job, is essentially a set of documents which are scheduled to enter the system for analysis in roughly the same time. A job could therefore be described as a set of documents, or even as one very big document where the metadata is the sum of the metadata for all the documents in the set.

For training of a model, these "big documents", denoted *artificial jobs*, could be created from a large set of documents. These jobs are artificial in the sense that the documents they are composed of do not necessarily come from the same original job. This approach had to be used due to limitations in how much data could be retrieved from the system - asking for all documents related to even a small number of jobs would require millions of database queries, and this would incur an unacceptable amount of extra workload.

In order to get training data for the model in this case, artificial jobs were simply created by sampling documents from the original set and defining the artificial job as the sum the parameters from the documents in this sample. The sample size was randomized within a predefined interval so that the model could train on both small artificial jobs and larger ones.

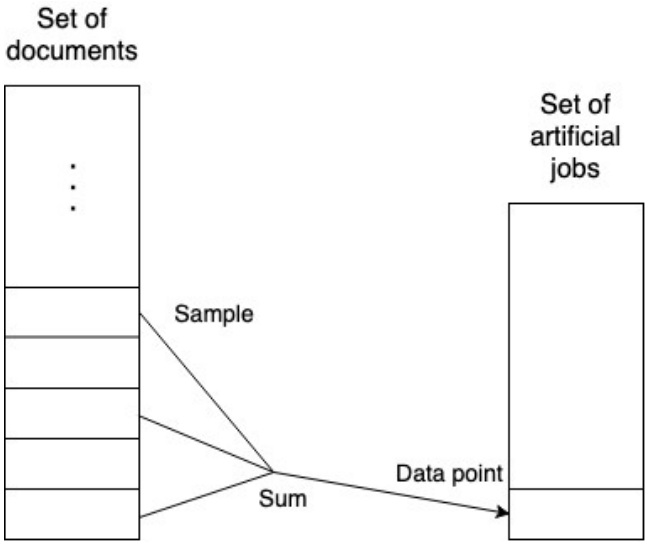

Figure 4.9: Combining document metadata from a sample into an artificial job, which is used as a data point for the training

As the size of a real job can be much larger than the set of documents at hand, the same documents was sometimes reused in the creation of artificial jobs in order to get both enough data points for training and also have them be a representative enough size. While this removed the complete independence between the data points in the different data sets, this trade-off had to be made to ensure sufficient amounts of training data was available.

4.10 Examining prediction error on documents analysed when system is manually over-provisioned

One of the main contributions and aims of this project is to investigate how well existing methods for predicting task execution time perform in a complex system used in a production environment. As mentioned earlier in this report, it is difficult to perform controlled experiments, such as decreasing the number of running servers or to treat processes other than the document analyser as black boxes. Furthermore, even if it was possible to manually reduce the complexity or the "noise" in the system, the results from such an experiment may not be representative of the real environment, which contradicts part of the purpose of the work in the first place.

However, it is possible to manually start more EC2-instances running document analysers than what is required. Due to the associated cost, this can only be done for a brief interval. RF's system designers believe that if there are more computing resources available than what is ever required, the system and consequently the processing times might become more stable, as the maximum load on individual instances should decrease.
Since this investigation was carried out at the behest of RF and does not strictly

relate to the project plan, details on the methodology and results can be found in Appendix F

5
Results

In this chapter the results and findings from the conducted work are presented. As in methodology, each model has its own section. Since each model used a separate data set the distribution and mean value of the processing times differed. It is important to keep this in mind when studying the obtained results, and also Because of this, each section contains some statistical information about the data the model was trained and tested on. Following this is a summary of the features used, the models prediction error, and a short analysis of the results. The prediction error is measured as Mean Absolute Error (MAE) divided by the mean Document Processing Time (DPT) of the validation set. As a reference, the results from using the naive prediction algorithm that, for each data point, guesses the mean and median value of DPTs of the corresponding data set is also included.

Subsequently, the results from training and testing two of the models on aggregated document metadata, the *artificial jobs*, is presented. The chapter is concluded with a summary of all results obtained.

5.1 Terminology

For easier interpretation of the results, a short list of terminology used throughout this chapter is available in the following list:

- **DPT** - Document Processing Time
- **MAE** - Mean Average Error
- **MedAE** - Median Average Error
- **Mean** - The mean DPT of the data set used when training and validating that model
- **Median** - The median DPT of the data set
- **MAE / mean** - The mean average error with respect to the mean DPT of the validation data, giving a percentage. Lower is better.
- **MedAE / median** - The median error with respect to the median DPT. Lower is better.

5.2 Model one - Predictions based on document metadata

This model has the least complexity in terms of structure and input, and is purely trained on document metadata in order to predict DPT. The first version of the model considered all features the designers of the document analyser believed could have an impact on processing times. These are detailed in Appendix B.

5.2.1 Feature selection

At first, a smaller dataset of 21,000 documents were used for the feature selection process. At first, all features were included in the input. Iteratively, features were removed and MAE was used as a comparison between iterations of training. Finally, when no more feature could be removed without having the error rise, the input set ended up having the following features:

- Number of references
- Number of entities
- Text length
- Max Offset

5.2.2 Prediction results

A new data set containing 136,000 documents was retrieved. These were sampled at random from an interval of thirty days. This set had a better distribution than the previous set, thus better representing the "average" behaviour of the processing times in the document analyzer. Furthermore, the feature *number of entities* was divided into two - *light* and *heavy* entities. This division was done because for light entities, a more lightweight key-value database lookup is performed, whereas a lookup for heavy entities requires additional computation. Statistics about the DPT for this data set is listed in Table 5.1.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Total	136,000	31	1,779,948	12,384	1,302	48,222	53,449
Validation	27,000	38	1,777,442	6,264	964	29,099	22,425

Table 5.1: Statistical information about the DPT in the data set used for training and testing model one

The following prediction error was observed:

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	4,690	**0.7503**	661	0.6710
Naive	8,684	**1.3945**	629	0.6367

Table 5.2: Model One prediction errors. Values are in milliseconds.

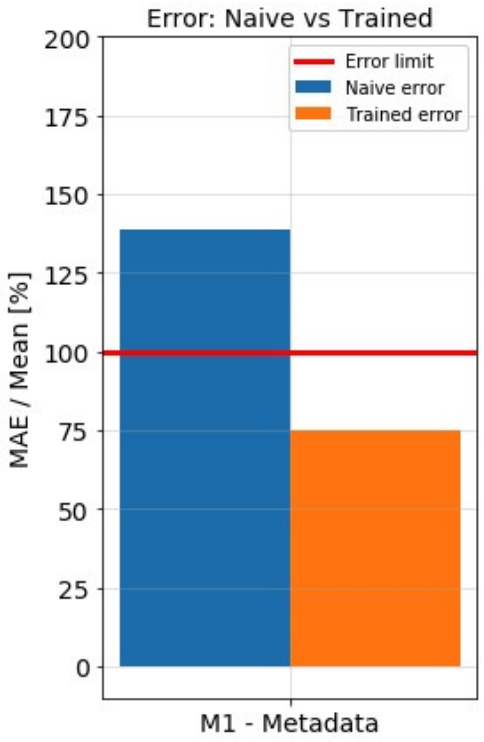

Figure 5.1: The error for model 1 compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.2.3 Analysis

An average prediction error of 75% is relatively high. In the case of a re-analysis, which is the only time the document metadata is known and could be used as input to a prediction model, one job can consist of hundreds of thousands or even millions of documents. The accumulated error over the entire job then becomes significant, and the prediction model might be of little use depending on what decisions its supposed to influence. However, the error is still half that of the naive prediction algorithm, and the findings from the feature selection are useful for the other models developed in this project.

5.2.4 Investigating fluctuations in document processing time

In order to get a more intuitive sense of how single categories of the metadata might affect the DPT, scatter plots were constructed for the metadata versus the DPT.

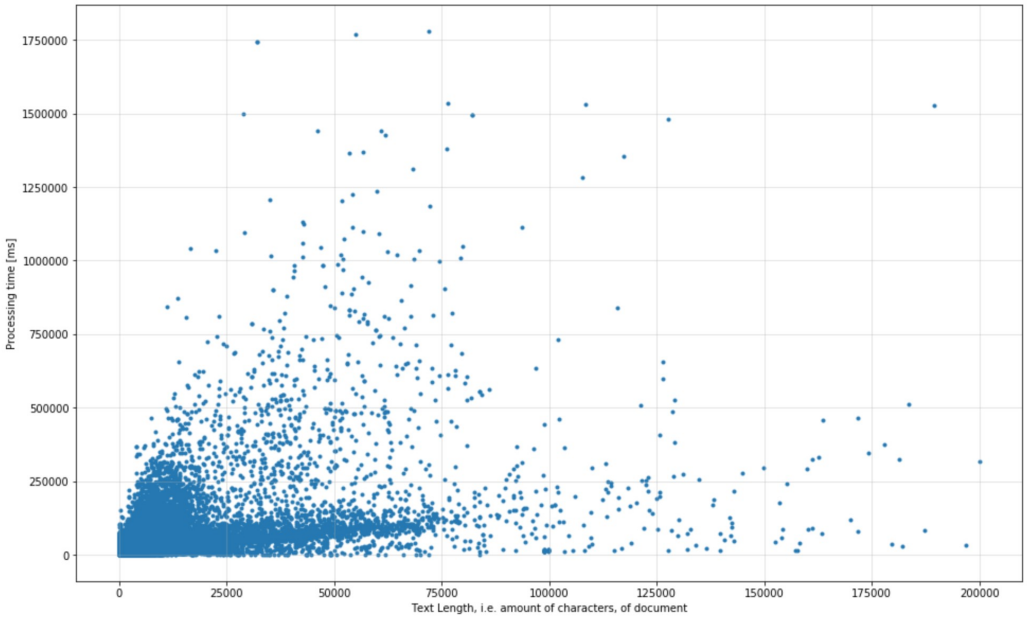

Figure 5.2: Scatter plot of document text length versus how long it took to analyze. Sample size: 136,000.

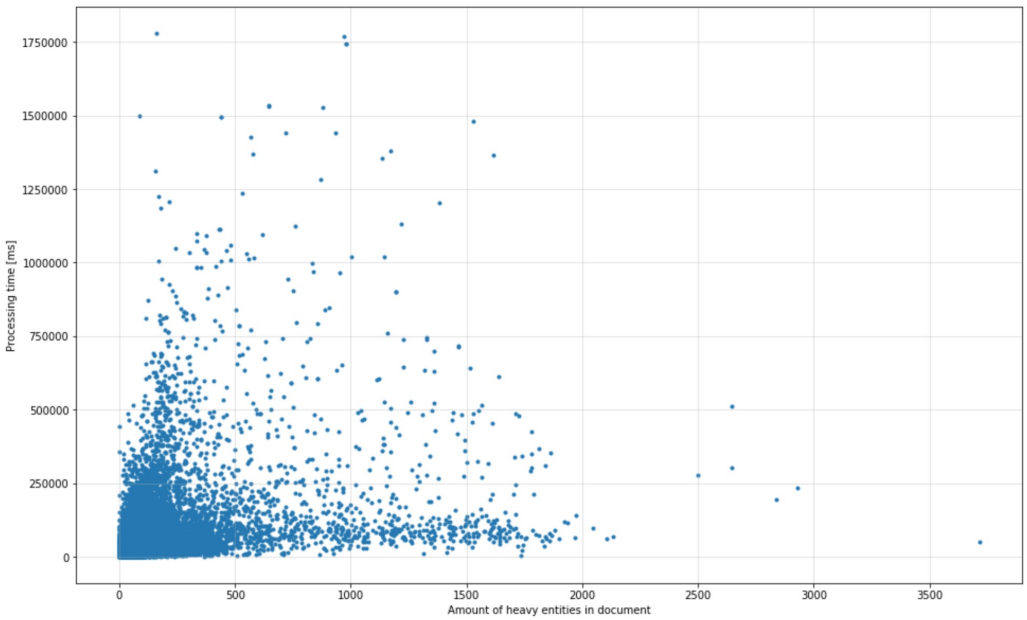

Figure 5.3: Scatter plot of the amount of heavy entities in a document versus how long it took to analyze. Sample size: 136,000.

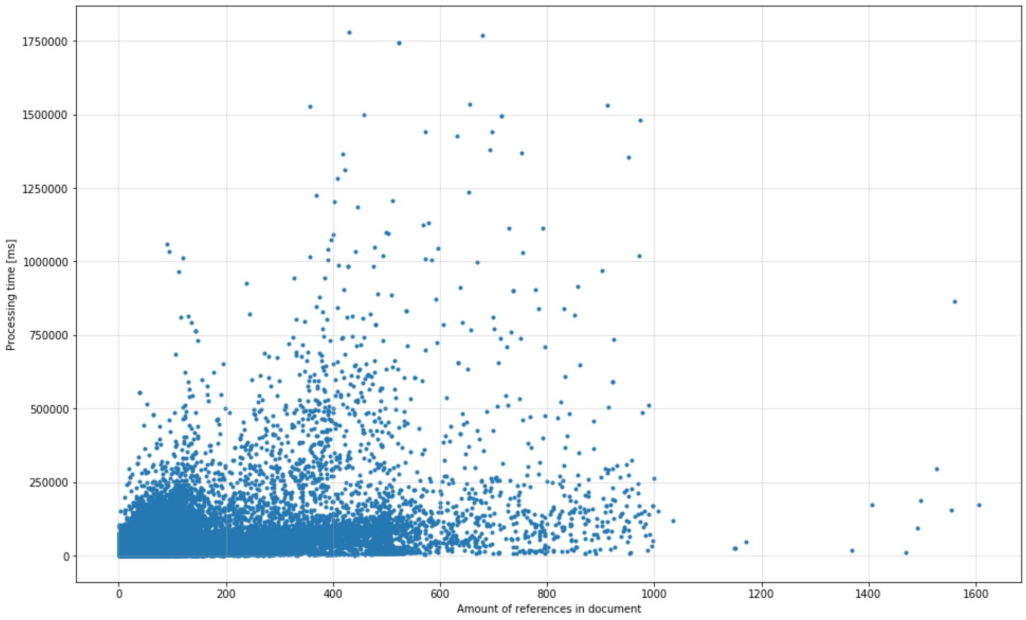

Figure 5.4: Scatter plot of the amount of references in a document versus how long it took to analyze. Sample size: 136,000.

As can be seen in Figure 5.2 there is no clear relation between a document's text length - arguably the most intuitive and general feature in the input - and the resulting processing time.

The same can be seen when plotting the duration versus the number of references and the number of heavy entities, in 5.3 and 5.4 respectively. Both of these metrics give an indication of how much interesting information the analyzer found in a certain document, and by extension how much computation other than the basic text parsing was required.

To gain further insight into these variations, a document was selected at random and ordered for re-analysis 400 times. The resulting processing times are depicted in Figure 5.5

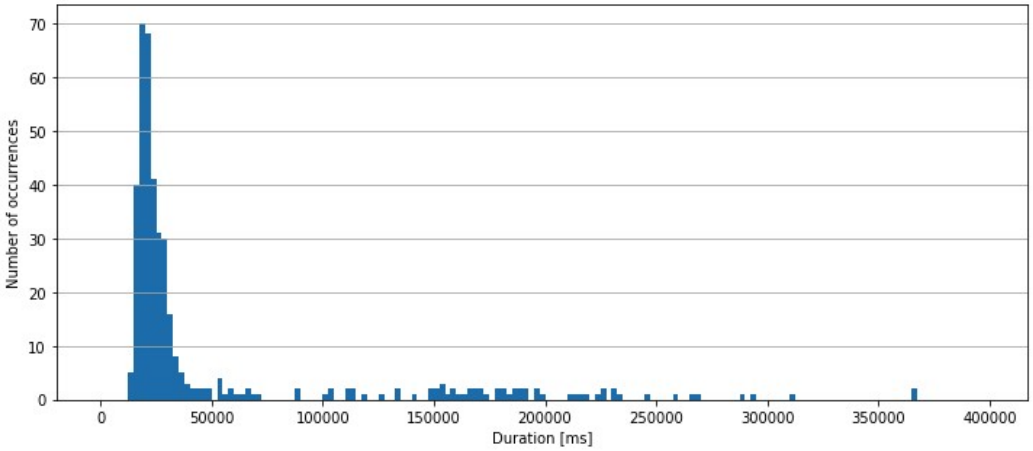

Figure 5.5: Histogram depicting processing times of 400 analyses of the same document

While the majority of the analyses take roughly the same time, there are a large number of outliers - indicating that the systems health at the current time of analysis plays a role in the final duration of the processing time.

5.3 Model two - Predicting using document metadata and system performance

First, the feature selection process will be presented, where the different combinations of features considered and the corresponding prediction errors (MAE / Mean) are shown. The results from the final model, along with statistical data about the data set used for testing and training follows.

5.3.1 Feature Selection

Engineers at RF suggested considering all available metrics from the individual EC2 instances as well as the corresponding Java Virtual Machine were considered. In addition, the features related to the document metadata (DMD) were used here as well. These 65 features were used as the base feature set. To determine the optimal input feature set, an iterative approach was used where a set of features were removed and the models prediction error was evaluated. A higher error indicated that those features had an effect on the DPT, and should be kept.

Feature set	**MAE / Mean**
All system performance and DMD	**0.375**
Only system performance	0.52
Only DMD	1.01
DMD and garbage collection	0.39
DMD and memory usage	0.48
DMD and all metrics related to the EC2-instance	1.00
DMD and database response times	0.47
DMD, CPU-usage, and thread count	0.54
DMD, CPU-usage, thread count, and database response times	0.44
DMD, CPU-usage, thread count, database response times, and garbage collection	0.36

Table 5.3: Input features considered and the corresponding prediction error with respect to the mean DPT of the data set

The base feature set, containing all available features (listed in Appendix B) provided the best results. Therefore, all these features were considered in the final model. However, the feature selection process revealed that it was possible to consider far fewer features and still achieve a similar prediction error.

5.3.2 Prediction results from final model

After the features that provided the lowest error had been identified, the model was trained and tested on the retrieved data set. Information about the DPT in this set, as well as the results from the final model are presented blow.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Total	5,700	300,058	1,803,149	539,739	442,524	267,115	1,125,507
Validation	1,150	300,196	1,742,121	507,804	427,419	233,566	990,492

Table 5.4: Statistical information about the DPT in the data set used to train and test the model considering selected features from the document metadata, and current system performance

The following prediction error was then observed:

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	372,571	**0.3632**	349,963	0.4073
Naive	461,798	**0.8109**	392,430	0.7550

Table 5.5: Prediction error when considering selected features from the document metadata as well as current system performance metrics. Values are in milliseconds.

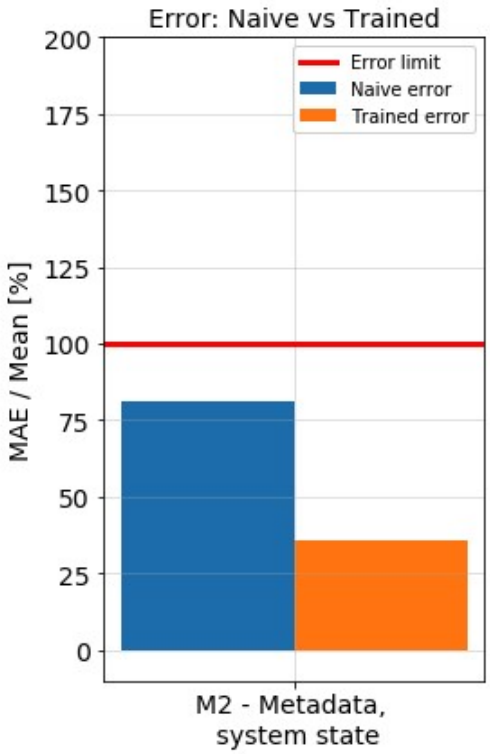

Figure 5.6: The error for model 2 compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.3.3 Analysis

Other than being a good result in and of itself, the achieved error of 35% is nearly half that of the error from the first model. It is clear that considering system performance at the time of analysis is important when trying to predict the DPT required.

However, it is important to note that this model would not be possible to use in the real system. Not only does it depend on metrics during the time of analysis (which obviously are not available beforehand), but all of the metrics except for database performance are on a per-instance level. Since the specific instance a document will be processed on is not known until that document is fetched from the queue by a document analyzer thread. Thus, it is not possible to use this model for more long-term predictions, such as determining how long a set of documents will take to process as soon as they enter the system.

A number of observations can be made when looking at the change in error during the feature selection process. Perhaps most interesting of all these is the fact that considering only system performance features gives a lower error than when considering the document metadata. Thus, for the sake of prediction accuracy the current system performance is more important than what the actual input to the system looks like. Also, considering all metrics from CloudWatch i.e the metrics

related to the EC2-instances performance gives the same error as only considering metadata. This is likely due to the majority of these metrics - network traffic and disk operations - not being correlated with DPT.

5.4 Model three - Considering documents analysed in parallel

Below are the results presented from the model that considered parallel work within the same machine. Here, the definition of parallel work is the sum of the metadata for each document overlapping the interval of which a target document was being processed (on the same instance). Each overlapping document had its metadata multiplied by a factor of how much it overlapped with the measured interval, assuming that the processing was evenly distributed. Together with the target document metadata, the "overlapping metadata" was used as input.

Data set	**Sample size**	**Min**	**Max**	**Mean**	**Median**	**STD**	**95 perc**
Total	2,800	5,002	1,800,003	26,274	15,018	54,769	81,548
Validation	568	5,005	531,948	25,399	15,929	39,714	83,204

Table 5.6: Statistical information about the DPT in the data set used for the parallel document investigation

The following prediction error was then observed:

Model	**MAE**	**MAE / Mean**	**MedAE**	**MedAE / Median**
Trained	9,244	**0.7079**	6,493	0.4640
Naive	10,517	**0.9206**	6,980	0.6156

Table 5.7: Prediction errors for the model which considers parallel work on the same instance. Values are in milliseconds.

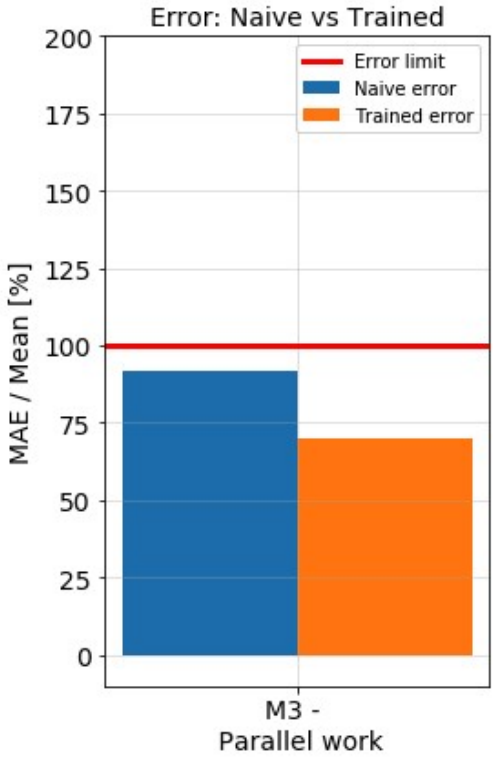

Figure 5.7: The error for model 3 compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.4.1 Analysis

While the prediction error is 5% lower than in model one, which is similar in the sense that it only requires information about the input, this model is more computationally intense as it requires metadata from all documents processed in parallel by that EC2-instance. If the target document took several minutes to analyse, information about hundreds of other documents could be required for the training data. The results indicate that the cores on each analyser are largely independent of each other.

5.5 Model four - Time series prediction

In this section, the results from making prediction based on trends and patterns in the processing times are presented. There were three stages of this investigation:

- Predicting DPT based on a number of previous DPT on the same document analyzer
- Dividing the DPT with the document text length, and then predicting the time required per character
- Predicting the average DPT across all analyzers using a series of one- and ten-minute average DPT.

5.5.1 Time series prediction, Per-instance

A recurrent neural network was designed and then trained and tested on five separate data series consisting of consecutive document analyses. Each series was obtained from a different EC2-instance and from a different interval.

5.5.1.1 Evaluating the look-back parameter

A set of evaluation runs with the look-back parameter - how many previous values are considered when predicting the target - when set to different values were conducted on one of the series.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Total	8,197	3	770,150	6,033	933	26,390	20,550
Validation	1,640	50	101,114	2,668	759	8,213	9,649

Table 5.8: Statistical information about processing times in the first data series used for per-instance time series prediction [ms]

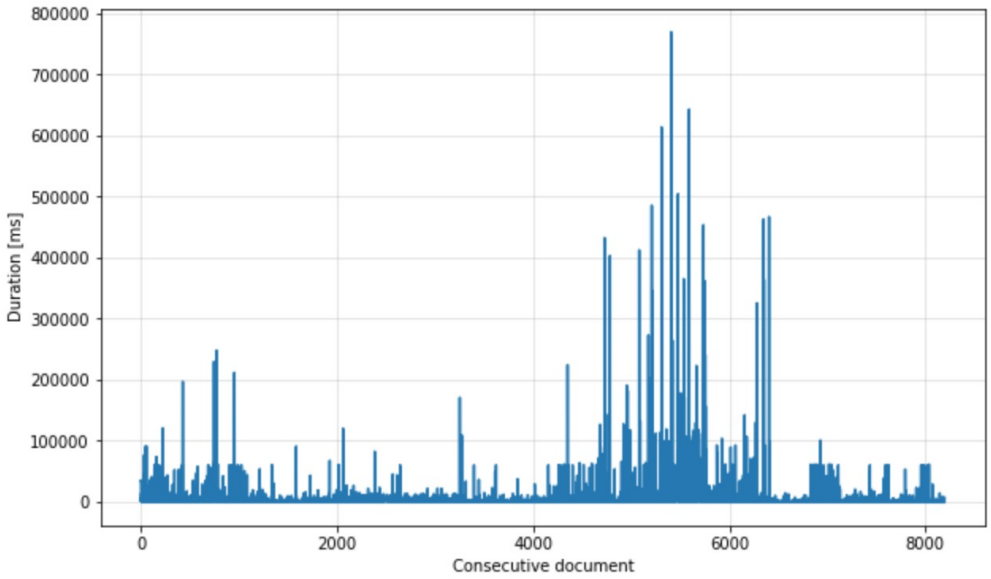

Figure 5.8: Plot of 8197 consecutive documents and their corresponding processing times [ms]

Look-back	MAE	MAE / Mean	MedAE	MedAE / Median
1	346	0.12	107	0.55
10	347	0.13	111	0.61
20	346	0.12	107	0.55
35	349	0.13	110	0.61
50	364	0.13	132	0.83

Table 5.9: Prediction error when considering different numbers of previous document processing times

The results revealed that the number of previous document analysis times considered had a negligible effect on the models error. Because of this, a look-back of 1 was used when evaluating the other four data series.

5.5.1.2 Prediction results

The results from the first series were very promising, with the model achieving a mean average error of just 346 milliseconds at best. The model was trained on the other four data series, and detailed information about each series, plots, and model prediction error are available in Appendix E. A summary of these results is presented in the table below.

Series	Mean (validation)	Median (validation)	MAE	MAE / Mean	MedAE	MedAE / Median
One	2,668	758	346	0.12	107	0.55
Two	8,812	812	5,047	0.57	319	0.29
Three	8,757	987	3,290	0.37	308	0.59
Four	6,021	1331	4,302	0.71	624	0.46
Five	4,258	681	3,920	0.92	412	0.61

Table 5.10: Summary of time series prediction results

Finally, the average of each series and the corresponding prediction error is available in Table 5.11

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	3,381	**0.538**	354	0.5
Naive	8,732	**1.388**	634	0.66

Table 5.11: Prediction error when using instance-level previous document DPT to predict the future DPT

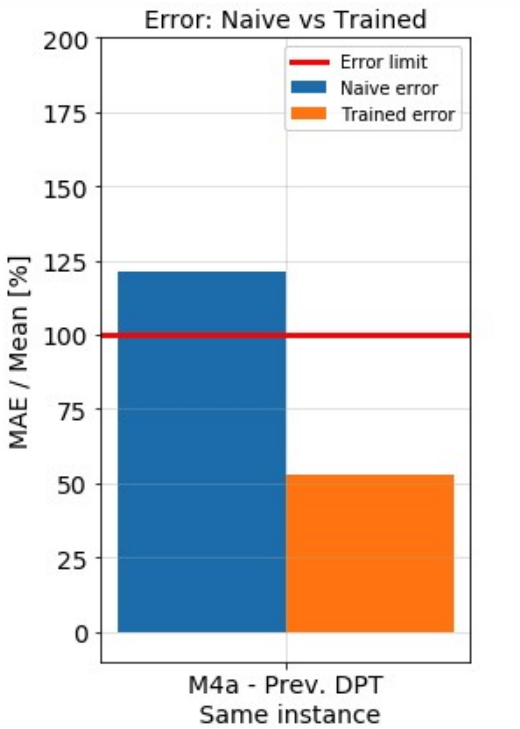

Figure 5.9: The error for model 4a compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.5.1.3 Analysis

An average error of 53% is achieved, which indicates that sometimes, previous DPT can give an estimation of future DPT. However, the results are inconsistent and

there are two caveats to this:

- Since there is no way to know in advance which EC2-instance a document will be analysed on, this model does not work in practice
- The error varies significantly between the series. This may indicate that the DPT is volatile, and a low prediction error is observed when testing a specific series is because that data set had more uniform DPT in the validation set. The fact that changing the look-back had negligible effect reinforces this hypothesis.

Figure 5.10 depicts this quite well - the actual DPT varies extensively between even two consecutive documents, and the model is unable to account for this in an accurate way.

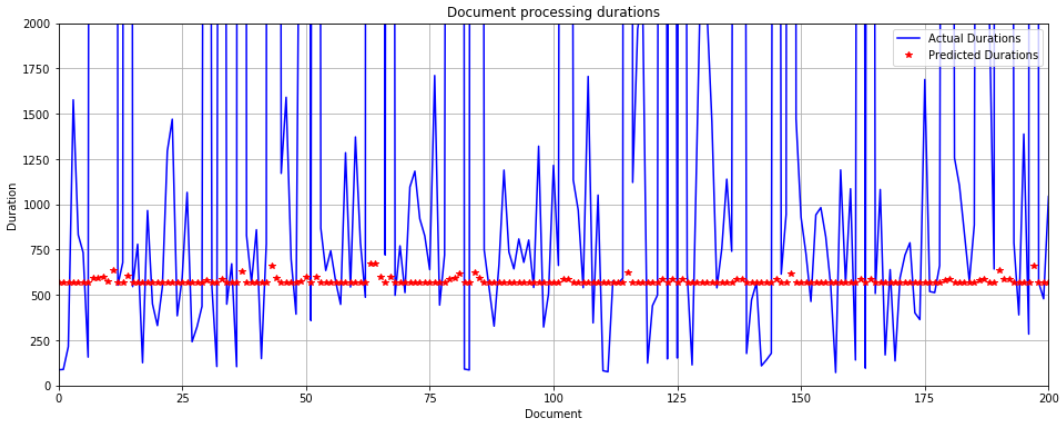

Figure 5.10: Zoomed-in plot of the predicted DPT versus actual DPT of the first 200 documents in a series

5.5.2 Time series prediction per instance, with weighted DPT

Three of the existing data series were selected, and the processing time of each document was divided with the text length. Effectively, this means the model was now trained on antecedent DPT per character, rather than processing time.

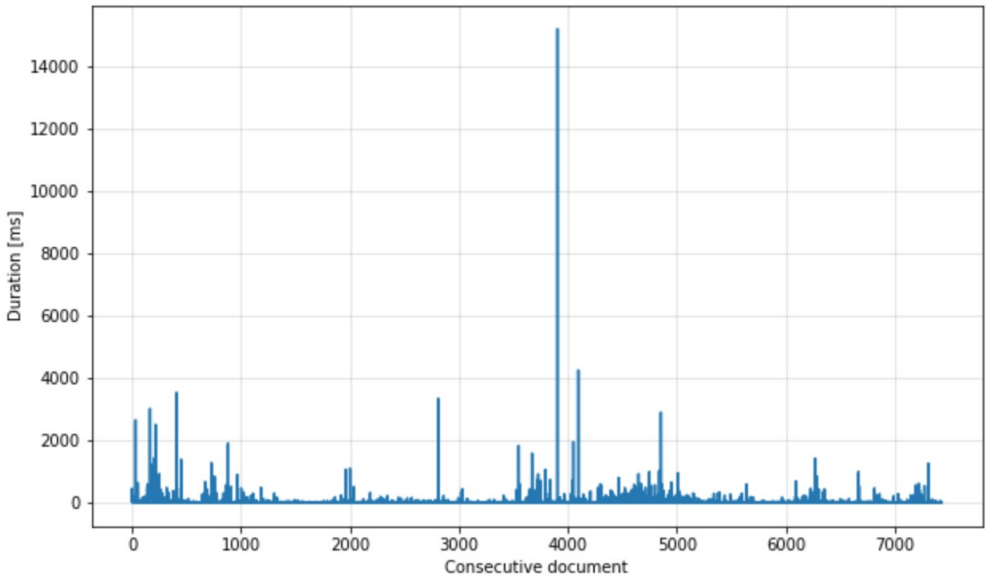

Figure 5.11: Plot of 8197 consecutive documents and their corresponding weighted processing times [ms]

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	7,434	0.0046	15,209	31.42	5.83	224	110
Validation	1,487	0.013	1,422	19.72	5.08	81	51

Table 5.12: Statistical information about DPT per character from the first set

The model was trained to predict processing time per character of the next document.

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1.64	0.08	0.38	0.07
Naive	25	1.27	3.63	0.71

Table 5.13: Model error when predicting time per character

Since this models target variable is DPT / text length (in characters), the results were multiplied with the average text length of the validation data set to obtain the MAE in DPT.

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,216	0.45	287	0.37
Naive	18,594	6.69	2,693	3.55

Table 5.14: Weighted series one results, in milliseconds

The model was trained and tested on series two and three as well. Plots and detailed information is available in Appendix E. A summary containing the mean of each prediction value across the three series is presented in Table 5.15.

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	7	0.352	2	0.275
Naive	61	3.32	12	2.12

Table 5.15: Average error across all three series when predicting processing time per character

Converting the error in DPT/text length back to DPT:

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	2,803	**0.42**	689	0.76
Naive	22,758	**4.211**	4,756	5.40

Table 5.16: Average error across all three series, in DPT

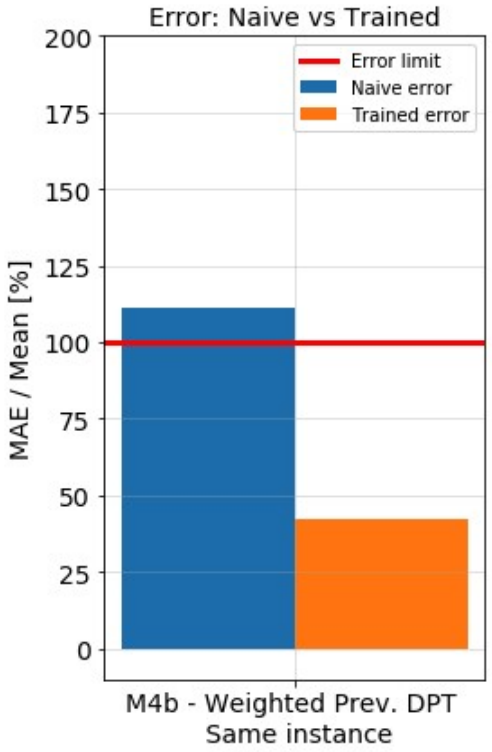

Figure 5.12: The error for model 4b compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.5.2.1 Analysis

The error is lower than in the previous model. This is to be expected, since the DPT is far more uniform due to being normalized with respect to the input size. However, the error still varies significantly and this model would be more complex to use in a real setting as it also needs information about the input, rather than just the processing times. The spikes that can be observed in the plot of the data set indicate very large outliers - documents that even when considering their size take significantly lower to process compared to the average. These outliers are also what cause the huge (compared to the other models) naive algorithm prediction error, as they cause the mean value of the DPT to increase significantly, leading to

5.5.3 Time series prediction - Average DPT across all document analyzers

The results of training the model to predict future average processing times on a one- or ten minute basis are available below.

5.5.3.1 One-minute intervals

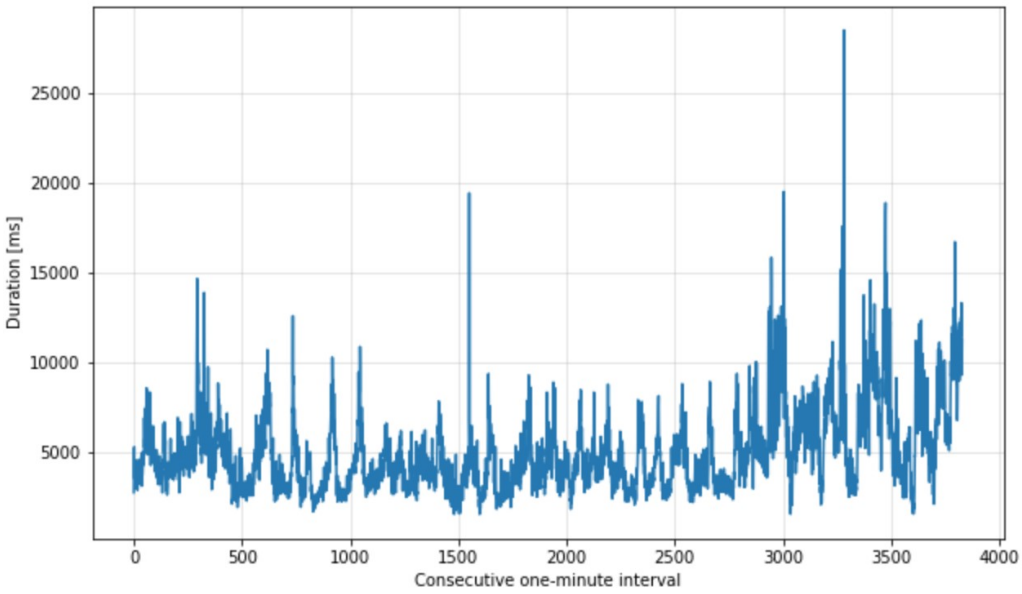

Figure 5.13: 3829 consecutive one-minute intervals and their average document analyzer processing time

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	3,829	1,504	28,523	4,903	4,294	2,322	9,565
Validation	766	1,530	28,523	6,934	6,588	2,867	11,579

Table 5.17: Statistical data about the series of one-minute intervals

Due to limitations in how much data could be retrieved, only one series was available for study. The model error when using five different values for the look-back parameter is presented in Table 5.18

Look-back	MAE	MAE / Mean	MedAE	MedAE / Median
1	2,806	0.41	2,414	0.36
10	2,670	0.38	2,534	0.35
20	2,788	0.40	2,391	0.36
35	2,802	0.40	2,399	0.36
50	2,772	0.39	2,372	0.36

Table 5.18: Model prediction error when considering a varying number of previous one-minute intervals

Changing the look-back has negligible effect on model prediction error. At best, the model achieves an error of roughly 40% with respect to the mean - a promising result, considering this data could be used in a production model.

For reference, an altered version of the naive algorithm which guessed the mean value of the intervals average processing time was tested. The results are shown in Table 5.19

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	2,769	**0.39**	2,410	0.36
Naive	2,217	**0.31**	1,809	0.27

Table 5.19: Prediction error when considering the average DPT of an increasing number of previous one-minute intervals

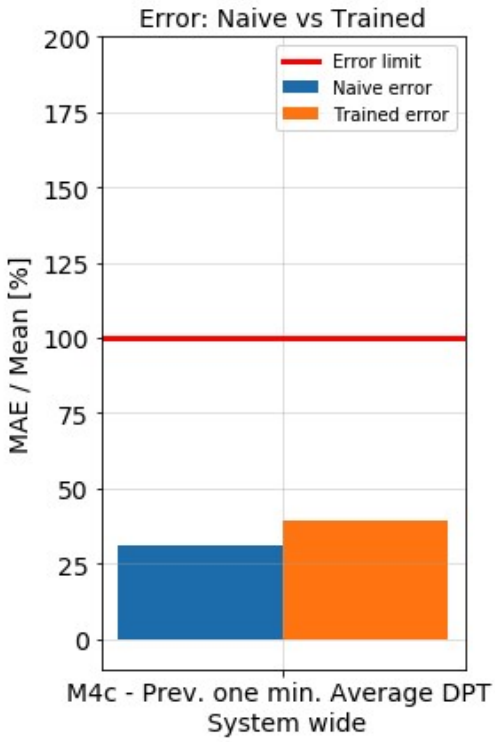

Figure 5.14: The error for model 4c, using one minute intervals, compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.5.3.2 Analysis

An error of 39% indicates that the average DPT does not vary significantly from minute to minute. However, considering the results from the prior models that considered instance-level DPT this can simply be due to luck, and more data would be required to verify the results. In addition, the naive mean-guessing algorithm performs better.

5.5.3.3 Ten-minute intervals

Considering the series of average processing times across ten-minute intervals instead.

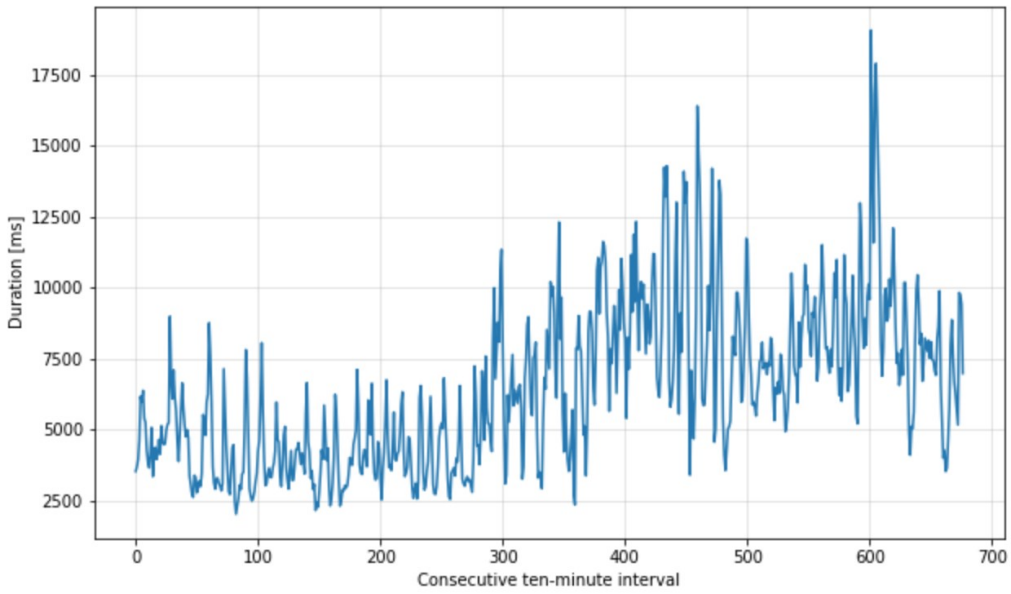

Figure 5.15: 677 consecutive ten-minute intervals and their average document analyzer processing time

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	677	2,014	19,071	6,420	6,054	2,863	11,390
Validation	136	3,517	19,071	8,512	8,176	2,530	12,351

Table 5.20: Statistical data on the set of ten-minute intervals

Due to technical limitations only one consecutive series of document analyses was retrieved. The model error when using five different values for the look-back parameter is presented in table 5.21

Look-back	MAE	MAE / Mean	MedAE	MedAE / Median
1	5,659	0.66	5,327	0.65
10	4,971	0.58	4,635	0.56
20	5,199	0.61	4,850	0.59
35	5,314	0.62	4,810	0.58
50	5,428	0.63	4,795	0.58

Table 5.21: Prediction error when considering the average DPT of an increasing number of previous ten-minute intervals

A look-back of 10 was found to be optimal. The results compared to the naive algorithm (which guesses the mean value) are as follows:

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	4,971	**0.58**	4,635	0.56
Naive	1,809	**0.21**	1,281	0.15

Table 5.22: Prediction error when trained and tested on the average DPT during the 10 previous ten-minute interval

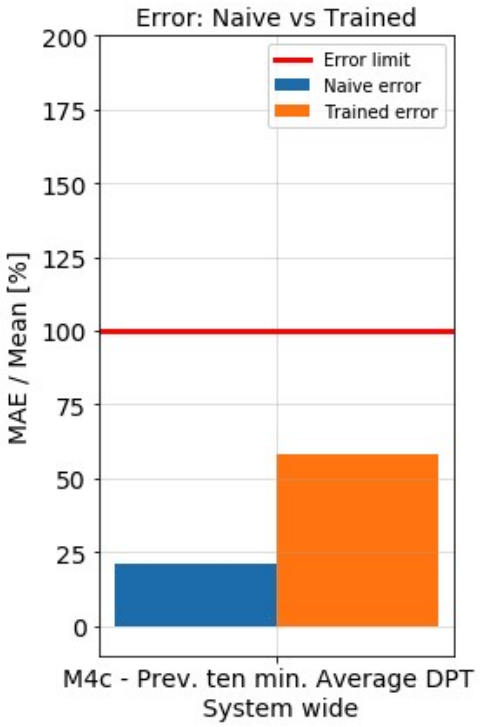

Figure 5.16: The error for model 4c, using ten minute intervals, compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.5.3.4 Analysis

As the naive algorithm performs far better than the trained network, the results are poor. Considering the volatility of the DPT in the system - which can be observed in the plots available in Appendix E - it is not that surprising that considering ten-minute intervals gives too poor of a resolution to make accurate predictions.

5.6 Model five - Combining previous processing times with document metadata

This model uses the same metadata as model one does, with the addition of using aggregated DPT from precedent documents on the same machine. Both average DPT and weighted average DPT of a set of documents from a different look-back was considered, in order to give the model a better picture of the past behaviour in the machine.

5.6.1 Previous DPT on same document analyzer/EC2-instance

The following features were used as input for training the model:

- Text Length
- Amount of References
- Amount of Light Entities
- Amount of Heavy Entities
- Max Offset
- Previous 1, 5, 10, 20, 35 and 50 Document Average Processing Times (6 features)
- Previous 1, 5, 10, 20, 35 and 50 Document Weighted Average Processing Times (6 features)

A new set of documents was retrieved for study. For each document, the DPT of a number of previous documents analysed on the same EC2-instance and the average DPT of the previous one-and ten-minute intervals was also computed.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Total	37,000	34	770,150	4,244	742	20,472	16,680
Validation	7,400	42	746,383	4,673	746	23,143	16,850

Table 5.23: Statistical data about DPT for the data set used when predicting based on document metadata and per-instance previous DPT

The prediction errors are displayed in Table 5.23

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	2,909	**0.7018**	387	0.5502
Naive	5,903	**1.4348**	481	0.6999

Table 5.24: Model prediction errors when considering document metadata and per-instance previous DPT [ms].

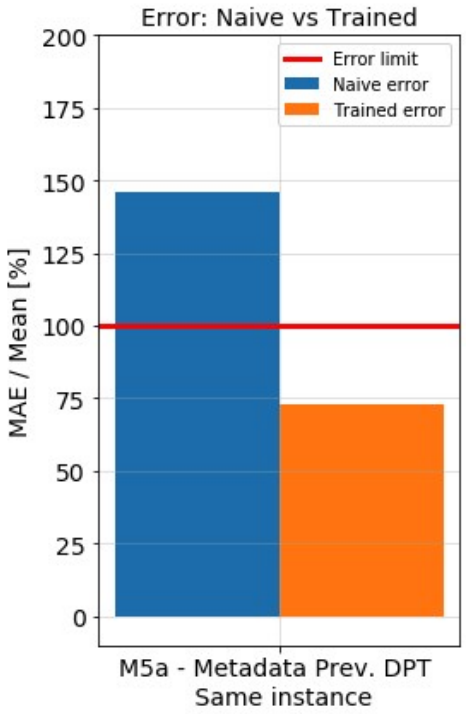

Figure 5.17: The error for model 5a, using one minute intervals, compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.6.2 Analysis

Just like model three, this model performs roughly 5% better than model one when looking at MAE / Mean. Aggregated DPT from previous documents seem to add little value to the predictions, which indicate that not much can be said from looking backwards - documents are independent from past processing times.

5.6.3 Average DPT across all analysers from past intervals

Two new consecutive series of documents were retrieved from the system, and the average DPT across all document analysers was computed on a per-minute basis. Each document was then paired with the average DPT of the system during the minute prior to its analysis. The resulting input feature set for the model then consisted of the following:

- Text length
- Number of references
- Number of light entities
- Number of heavy entities
- Max offset
- Average DPT across the system during the minute prior to the analysis of this document

5.6.3.1 First series

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	4,561	51	574,427	4,961	848	21,078	23,008
Validation	913	51	243,183	3,933	803	15,613	18,908

Table 5.25: Statistical data about the DPT in the first series used for predictions based on document metadata and previous average DPT

The model was trained and tested on the first series, and the prediction error is available in table 5.26

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,755	**1.03**	396	1.08
Naive	2,389	**1.42**	249	0.62

Table 5.26: Average prediction error when considering document metadata and previous average DPT across the system, series one [ms]

A plot of the average DPT during the (consecutive) intervals:

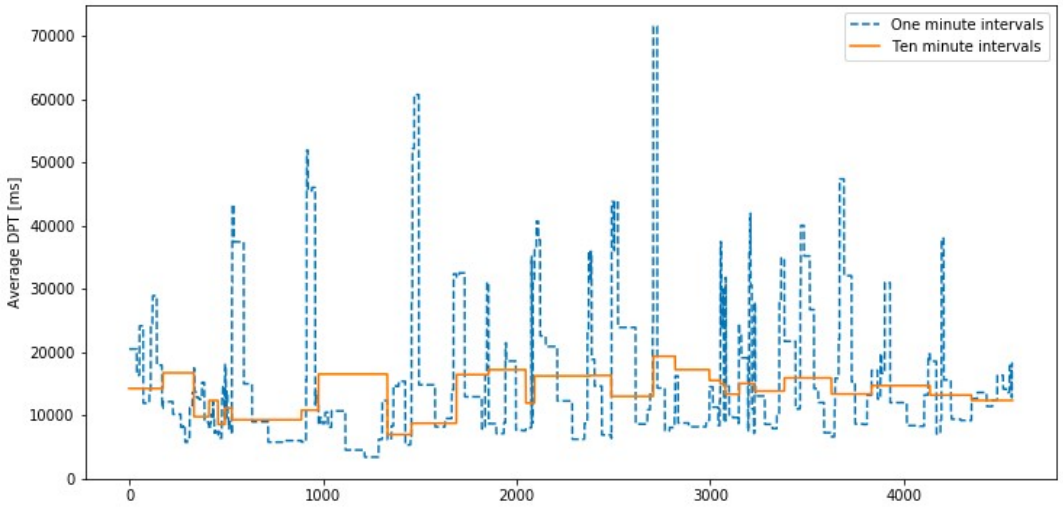

Figure 5.18: Average DPT across all document analysers in the first series considered when training and testing M5, in one- and ten-minute intervals. Note the large fluctuations in average DPT, particularly on a minute-to-minute basis.

5.6.3.2 Second series

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	6,945	42	770,150	5,571	833	24,852	19,411
Validation	1,389	50	101,141	2,530	725	8,046	9,636

Table 5.27: Statistical data about the DPT in the second series used for predictions based on document metadata and previous average DPT

The results from the second series were slightly better than the results from the first.

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	339	**0.88**	100	0.55
Naive	461	**1.25**	97	0.52

Table 5.28: Average prediction error when considering document metadata and previous average DPT across the system, series two

A plot of the average DPT during the (consecutive) intervals:

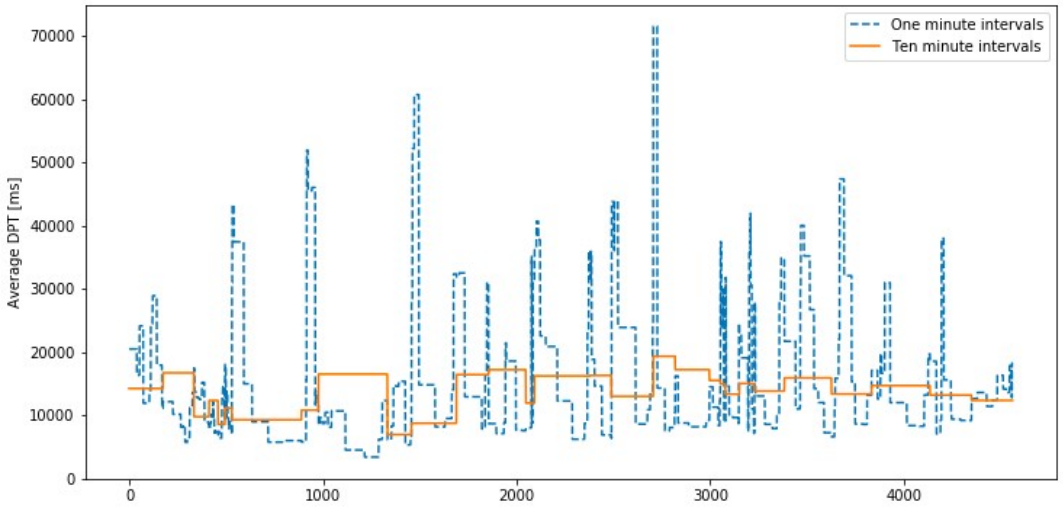

Figure 5.19: Average DPT across all document analysers in the second series considered when training and testing M5, in one- and ten-minute intervals. Note the large fluctuations in average DPT, particularly on a minute-to-minute basis.

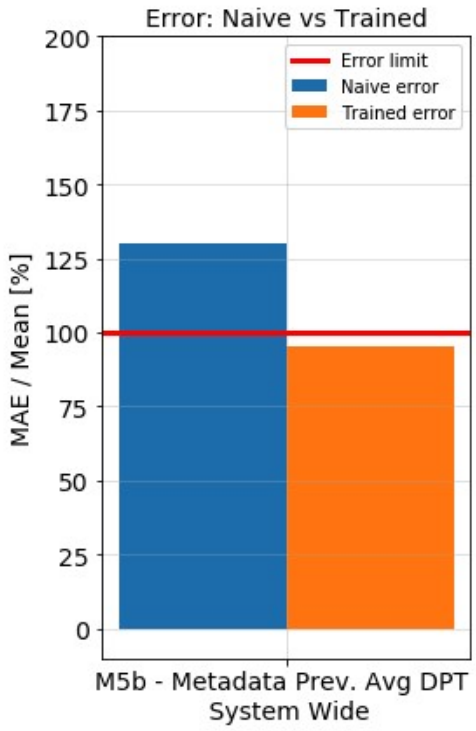

Figure 5.20: The error for model 5b, using one minute intervals, compared to the error of the naive prediction algorithm. The red bar depicts the maximum error considered acceptable by RF.

5.6.4 Analysis

Adding instance-level previous DPT slightly reduces the error compared to only considering document metadata, but this is not available for a real model. On the contrary, average DPT across entire system increases the error, indicating that average DPT is too coarse-grained to make accurate predictions. Overall, the results are in-line with the findings from model four: the DPT is too volatile to aid any prediction model and any good results may be down due to luck.

5.7 Job-level predictions

Below are the results presented for the predictions made on artificially created jobs, which could also be considered very big documents, as they are the sum of batches of sampled documents from the data set of 136,000 documents. As can be seen in Table 5.29, values in absolute milliseconds are very large since up to roughly 100,000 documents were combined into jobs.

The validation set here is very small, and this is because actual jobs that had been run through the system were used as validation. As a job can contain thousands of documents, and due to the aforementioned limitations when fetching data from the production environment, the validation set had to be limited in size of the set and size of each job. Each job added to the validation set had a lower limit of 100 documents and an upper limit of 10,000 documents.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Training	100,000	569,691	1,730,156,359	832,716,258	831,338,277	484,093,772	1,588,108,395
Validation	53	3,765,619	240,370,376	26,862,209	15,668,748	41,302,942	68,427,973

Table 5.29: Statistical data on the DPT in the data set. Values are in milliseconds.

The prediction errors are displayed in Table 5.30

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	30,363,901	**0.4012**	27,408,256	0.4423
Naive	37,965,706	**0.8467**	28,610,938	0.5790

Table 5.30: Model prediction errors. Values are in milliseconds.

5.7.1 Analysis

Being able to accurately predict the time required for jobs is closer to RF's original use case and vision. The model achieves an error of 40%, which is relatively good. The result is interesting considering model one, which predicts the time per document, is 70% off at best. The errors seem to cancel out over a large set of documents, rather than accumulate.

However, due to the large quantities of data required for proper verification, the results may not be that representative of use in production. Additional work is required to determine if the results are simply down to luck or not.

5.8 Results - Summary

To conclude this chapter, the prediction error achieved using all of the different models will be presented in a table for quick reference.

Number	Name
1	Document metadata
2	Document metadata, system performance
3	Document metadata and analyzer total workload (parallel documents)
4a	Time series, same instance
4b	Time series, same instance and weighted DPT
4c	Time series, average DPT across all analyzers during next interval
5a	Document metadata, past document durations on same instance
5b	Document metadata, past average document durations across entire system

Table 5.31: Model numbers and their names

Model no.	**1**	**2**	**3**	**4a**	**4b**	**4c**	**5a**	**5b**
Total sample size	136,000	5,700	2,800	43,419	18,552	4595	37,000	11,506
Validation sample size	27,000	1,150	560	8,686	3,691	766	7,400	2302
Mean [ms]	6,264	507,804	25,399	6,103	6,745	4,903	2,803	3231
MAE [ms]	4,690	372,571	9,244	3,381	2,803	3,381	1,844	1047
Median [ms]	964	427,419	15,929	913	852	4,294	651	786
MedAE [ms]	661	349,963	6,493	354	689	354	355	248
MedAE / Median [%]	67	41	46	38	50	76	57	81
MAE / Mean [%]	**75**	**36**	**70**	**53**	**42**	**58**	**73**	**95**
Compared to naive [%]	-64	-45	-22	-68	-69	9	-73	-35

Table 5.32: Prediction error achieved for each of the models investigated in this project.

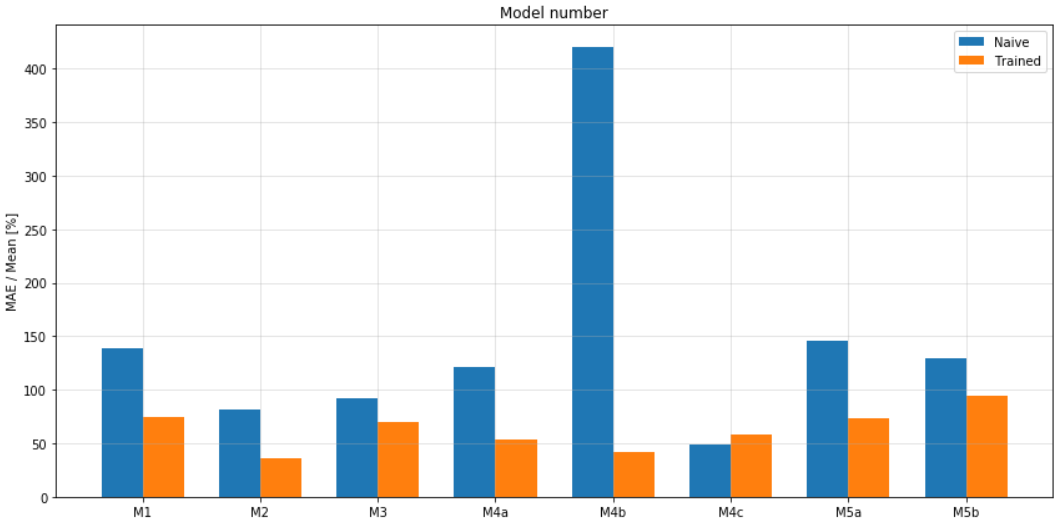

Figure 5.21: Chart depicting prediction error by the naive algorithm and the trained model

6
Discussion

In this chapter, some general reflections we have regarding this project, as well as some of the key insights garnered while developing the prediction models will be presented. Section 6.1 contains an analysis of the performance achieved by each model, and in section 6.2 a discussion regarding whether or not the model error can be said to be indicative of which factors are responsible for an increase or decrease in processing time.

6.1 Model performance

The mean average error achieved when considering different input feature sets vary significantly - the worst model has an error of over three times that of the most accurate model. Also, in the majority of cases the naive prediction algorithm performs far worse than the neural network. This indicates that it can be worth it to set up a solution based on machine learning, even though it is more complex and requires more computation.

The best results are achieved by M2 (Model 2), which considers a combination of document metadata and EC2-instance/JVM performance during analysis. Not only is the error half that of the first model, but the feature selection process revealed that considering only system state metrics achieves a lower error than when considering only document metadata, i.e. the input to the system.

Some results are perhaps more surprising than others - for instance, M3 which considers all input currently being processed on an instance is only 5% more accurate compared to M1. This, in combination with the results achieved when considering EC2-instance performance in M2 indicate that the virtual cores on the instance are highly independent, and the problems with fluctuating processing time described in the results from the first model can not be attributed to unfair resource sharing.

It is important to note that not all of the models could be used for prediction in a production environment, at least not in RF's system. M2 and M3 are based on data from the specific EC2-instance a document was processed on, and with the current system configuration it is not known which instance a document will be allocated to beforehand. In addition, both models consider what happened *during* analysis of the document, which obviously is unknown until after the document has been processed. M4a, M4b and M5a all make use of DPT on a per-instance level.

Finally, all of the predictions that make use of previous processing times in some way seem to be quite volatile. While some of the achieved results are quite good - M4a and M4b are the second and third best performing models - the error varies significantly between series. Also, as mentioned previously these models are based on instance-level data which is not available until after a document has been processed and these can therefore not be used in the real system. M4c which considers average DPT and tries to predict the average DPT for the next one- or ten-minute interval across the entire system achieves an error of 42% . While not bad, the naive algorithm which always guesses the mean DPT in the data set performs better. This indicates that the neural network has a hard time identifying patterns and trends in the data, which is not surprising considering the fluctuations observed in some of the data sets when plotting the DPT.

There is also a problem related to the way the time-series models are tested - as the data points have to be consecutive, the model is validated on the last 20% of the set. If the DPT in the validation set is more uniform, a good result is achieved and vice-versa.

The results from M5a and M5b, where each data point consisted of a combination of document metadata and previous DPT, reinforce the sentiment that past processing times are not necessarily indicative of future ones. Since the data used here does not have to consecutive, the aforementioned problem of skewed validation data is avoided. In this sense, the results may be more representative than the Both of these models perform worse than M1.

In summary, we do not think a prediction model based on time-series data is going to provide accurate results in this system.

6.2 Prediction Error versus causes of varying processing times

One of the aims of this project was the increased insight gained into which aspects of a complex system should - or should not - be considered when trying to optimize or otherwise analyse the input processing time. While a lower prediction error when considering a set of input features may indicate that those features play a large role in determining the required processing time, it is important to remember the old axiom: *"correlation is not causation"*. As mentioned several times throughout this report, we have been unable to retrieve enough data to be able to properly verify many of the findings. Other than the architecture of RF's system being highly complex in terms of the number of servers, queues, and general, each process is highly advanced and consists of tens of thousands lines of code. As a result of this, there is an astronomically large number of paths one piece of input can take through the system, and an almost equally large number of things that can go either wrong or behave in an anomalous way.

With this in mind, our methodology should not be interpreted as a relatively simple way to identify performance problems, and the results should not be considered in a vacuum - just because considering garbage collection significantly reduced the prediction error in our case, it does not mean that garbage collection can be said to be a general cause of increased task execution time in general systems.

7

Conclusion

"Debugging distributed systems is hard." [33]

In this section, we summarize our findings and thoughts. We also suggest future work based on the results and insights we have garnered while working on this project.

One of the goals for this thesis was to determine if it was possible to answer the question *"how long will this operation run for?"* in a complex distributed system. Based on the results, our cautious answer is "Yes, but..." followed by an explanation of the problems that are associated with that particular approach. There is always a trade-off between low prediction error and model complexity with respect to how hard it would be to implement it in a real setting.

However, the final models could still be valuable, depending on how accurate the predictions need to be. In particular, the promising results from aggregating input to larger jobs indicate that they could be used to get an initial overview of how much computation is going to be performed by the system in the near future.

In addition, other than giving an indication of which approaches should be considered further, the results also show that some features are not at all important when it comes to predicting task execution time. As mentioned in the discussion, however, this does not necessarily mean that those aspects can be ignored when developing similar models for other systems.

Many of the challenges we have faced have mostly been practical ones in the sense that they arise from the use of a very chaotic and non-laboratory nature of a real system, the problem of forecasting task execution time or workload in a distributed system is hard by its very nature. That said, it is our belief that the problem would have been made far easier had more information been available. In particular, more detailed data-lineage - logs describing exactly what happens to a piece of input as it passes through the system - would have saved time by making it far easier to study what happens to the documents.

7.1 Future work

The results and findings in this project provide a number of interesting options for future work. One of these is to go more in-depth when investigating system performance and its effect on processing time. While the results were promising, future research could try to confirm whether or not certain system metrics can be said to have an effect on processing time in general systems, using a more laboratory and controlled environment.

RF's original vision for our project was a prediction model that could forecast whether or not a set of input would necessitate a scale-up of the system to handle the increased load. Due to the limited time available, we decided to focus on time required instead, as it was considered easier to evaluate and more fine-grained data was available. Our results show that the processing time can be predicted to a reasonably good degree, and it could be possible to use this information to develop a model that, based on the estimated total processing time required for a set of input and current message queue size, tries to predict how many additional EC2-instances will be required.

Another interesting project is to try to develop a more advanced scheduling algorithm [34]. Currently, there is no real allocation algorithm used for the document analysers. Each analyser thread processes one document at a time, and when complete it fetches the first document on the queue. Considering the results from the investigation on instance performance at the time of analysis, it might be possible to get lower processing times by implementing an algorithm that tries to ensure fair workload across the instances, and by considering factors like when an instance is performing garbage collection.

Bibliography

[1] Iverson, M. A., Ozguner, F., & Potter, L. C. (1999). Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99) (pp. 99-111). IEEE.

[2] Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. Mobile networks and applications, 19(2), 171-209.

[3] Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information sciences, 275, 314-347.

[4] Bizer, C., Boncz, P., Brodie, M. L., & Erling, O. (2012). The meaningful use of big data: four perspectives–four challenges. ACM Sigmod Record, 40(4), 56-60.

[5] Rimal, B. P., Choi, E., & Lumb, I. (2009, August). A taxonomy and survey of cloud computing systems. In 2009 Fifth International Joint Conference on INC, IMS and IDC (pp. 44-51). Ieee.

[6] Google Architecture, http://highscalability.com/google-architecture Visited: May 2019

[7] Dean, J. (2009, February). Challenges in building large-scale information retrieval systems: invited talk. In Proceedings of the Second ACM International Conference on Web Search and Data Mining (pp. 1-1). ACM.

[8] van Steen, M., Pierre, G., & Voulgaris, S. (2012). Challenges in very large distributed systems. Journal of Internet Services and Applications, 3(1), 59-66.

[9] Wang, J., Crawl, D., Purawat, S., Nguyen, M., & Altintas, I. (2015, October). Big data provenance: Challenges, state of the art and opportunities. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 2509-2516). IEEE.

[10] Herbst, N. R., Kounev, S., & Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13) (pp. 23-27).

[11] Recorded Future: Threat Intelligence Powered By Machine Learning, https://www.recordedfuture.com/, Visited: February 2019

[12] Keras: The Python Deep Learning library https://keras.io/, Visited: June 2019

[13] TensorFlow: An end-to-end open source machine learning platform https://www.tensorflow.org/, Visited: June 2019

[14] PyTorch: An open source deep learning platform that provides a seamless path from research prototyping to production deployment. https://pytorch.org/, Visited: June 2019

[15] S. Strandberg, S. Westlund "Automatic Scaling of machine resources in complex computing systems", M.Sc Thesis, Chalmers University of Tech-

nology, June 2018. Accessed on: Jan 25, 2019. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/256191/256191.pdf

[16] Roy, N., Dubey, A., & Gokhale, A. (2011, July). Efficient autoscaling in the cloud using predictive models for workload forecasting. In 2011 IEEE 4th International Conference on Cloud Computing (pp. 500-507). IEEE.

[17] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., & Stoica, I. (2016). Ernest: efficient performance prediction for large-scale advanced analytics. In 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16) (pp. 363-378).

[18] Moreno, I. S., Garraghan, P., Townend, P., & Xu, J. (2014). Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. IEEE Transactions on Cloud Computing, 2(2), 208-221.

[19] Pham, T. P., Durillo, J. J., & Fahringer, T. (2017). Predicting workflow task execution time in the cloud using a two-stage machine learning approach. IEEE Transactions on Cloud Computing.

[20] Askalon Programming Environment, http://dps.uibk.ac.at/projects/execution/ Visited : May 2019

[21] Lee, B. D. (2003, December). Run-time prediction of parallel applications on shared environments. In 2003 Proceedings IEEE International Conference on Cluster Computing (pp. 487-491). IEEE.

[22] Pietri, I., Juve, G., Deelman, E., & Sakellariou, R. (2014, November). A performance model to estimate execution time of scientific workflows on the cloud. In 2014 9th Workshop on Workflows in Support of Large-Scale Science (pp. 11-19). IEEE.

[23] RabbitMQ, https://www.rabbitmq.com/ Visited: February 2019

[24] MongoDB Atlas, https://www.mongodb.com/ Visited: February 2019

[25] ElasticSearch, https://www.elastic.co/products/elasticsearch Visited: February 2019

[26] Amazon Web Services, https://aws.amazon.com/ Visited: May 2019

[27] Lindholm, T., Yellin, F., Bracha, G. & Buckley, A. (2015). Oracle Documentation. Retrieved from https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

[28] Maas, M., Harris, T., Asanović, K., & Kubiatowicz, J. (2015). Trash day: Coordinating garbage collection in distributed systems. In 15th Workshop on Hot Topics in Operating Systems (HotOS XV).

[29] Graphite App, https://graphiteapp.org/ Visited: May 2019

[30] Grafana Labs, https://grafana.com/ Visited: May 2019

[31] Amazon Cloudwatch, https://aws.amazon.com/cloudwatch/ Visited: May 2019

[32] Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. Geoscientific model development, 7(3), 1247-1250.

[33] Whittaker, M., Teodoropol, C., Alvaro, P., & Hellerstein, J. M. (2018, January). Debugging Distributed Systems with Why-Across-Time Provenance. In Proceedings of the ACM Symposium on Cloud Computing.

[34] Al Nuaimi, K., Mohamed, N., Al Nuaimi, M., & Al-Jaroodi, J. (2012, December). A survey of load balancing in cloud computing: Challenges and algo-

rithms. In 2012 Second Symposium on Network Cloud Computing and Applications (pp. 137-142). IEEE.

A

System overview

A.1 Poller

While some of RF's data comes from pre-processed files that are manually fed into the system, a large portion of the documents come from the automatic web scraping performed by processes called pollers or harvesters. The poller processes continuously check a variety of online sources for new data. Upon discovery, information on where this data can be found is sent to the resolver for duplicate checking.

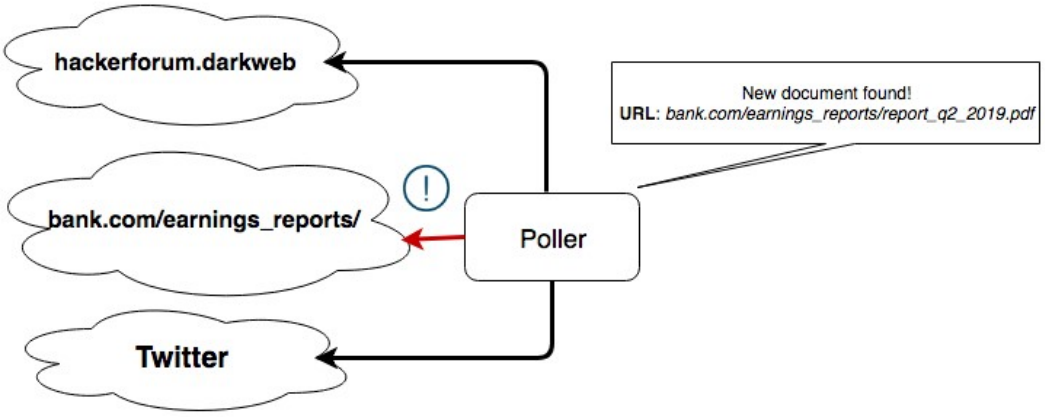

Figure A.1: Example of a poller finding new information from an online source

A.2 Document Resolver

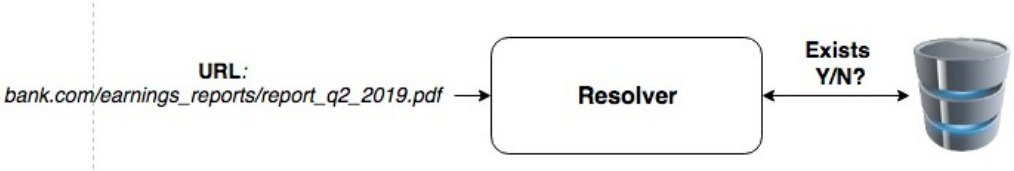

Figure A.2: The resolver performing duplicate checking using a database lookup

The primary function of the resolver is duplicate checking - that is, making sure that the same document is not being downloaded and subsequently analyzed more than once. In some cases this can be done by simply checking the URL of the source,

but this does not always work as the same file can exist in multiple places on the web. To deal with this, the resolver performs some more sophisticated duplicate checking using internal databases. If it discovers that a document already exists, it can bypass the downloader entirely and instruct the text extractor to start parsing the document.

A.3 Document Downloader

If the resolver has determined that a file does not already exist in the system it sends a request to the document downloader to fetch it from the web. As the look and format of documents can vary significantly depending on the source - a tweet, a forum post, a pdf, or a code repository - some sophisticated functionality and error handling is needed to download the document correctly. Once downloaded, the document is stored in a database for future retrieval.

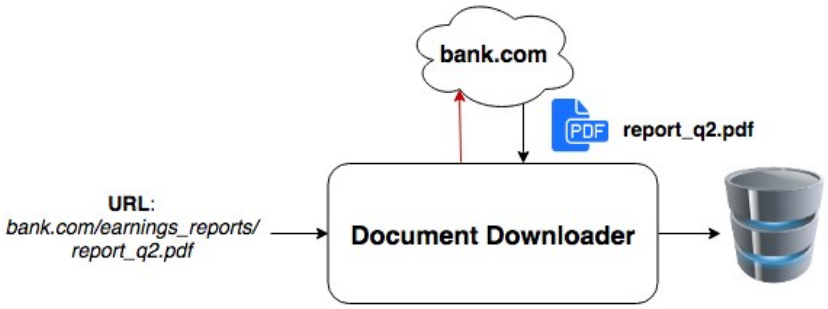

Figure A.3: The downloader fetching a document for storage

A.4 Text Extractor

As the name implies, the text extractor processes the downloaded file and extracts the text contained in it. Depending on the type of file the complexity of this operation varies greatly - reading a text file is far simpler than extracting text from an online magazine with images and adverts.

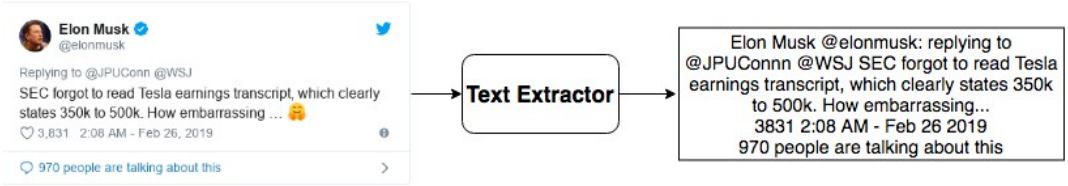

Figure A.4: An example of (simple) text extraction from a tweet

A.5 RabbitMQ

RabbitMQ is a message broker used to pass messages from one process in the system to another [23]. The basic premise is that the first process (known as a *producer* in

this context) appends messages to a *queue* which is listened to by the second process (known as a *consumer*). When new data is pushed to the queue as a *message*, the consumer is notified. If the consumer is unable to process messages at the rate the producer pushes them the queue will grow in size. Queue size can sometimes be an indicator of delays and bottlenecks in the system.

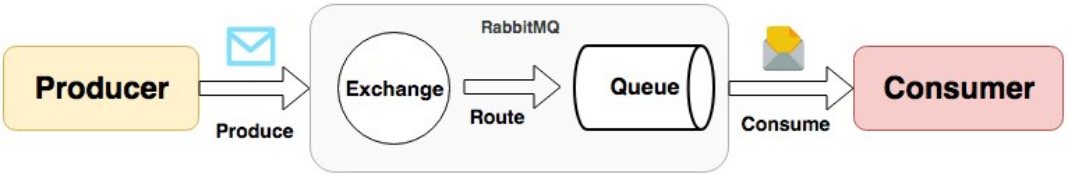

Figure A.5: Overview of RabbitMQ

RabbitMQ also supports more advanced functionality like load balancing and message routing. A component known as a *exchange* directs messages to the appropriate queue based on a set of rules. This feature is used by RF to let some documents skip certain stages in the pipeline.

A.5.1 The consumer

A RabbitMQ consumer is not an internal construct of RabbitMQ itself - it is a real process, executing some program, configured to listen to a particular queue. For instance, if the document analyzer *queue* has 100 consumers, that means there currently are 100 processor cores running the document analyzer *process*. The number of consumers of a specific queue is therefore a quick way to gauge how much workload the corresponding part in the analysis chain is currently experiencing. Similarly, the total number of RabbitMQ consumers across all queues can give a rough estimate of how much stress the system is under.

A.6 MongoDB

A NoSQL database [24] that uses JSON-like format to execute queries. A MongoDB database consists of collections, and the collections consist of documents. A document (not to be confused with what is denoted at the input for RF's system) consists of arbitrarily many key-value pairs. MongoDB is what RF uses as their database, where they store a majority of their collected data.

A.7 ElasticSearch

A search and analytics engine [25]. It is based on a REST-API and it provides functionality like **full text searching** where a segment of text is divided into smaller parts for efficient lookup.

B

Overview of all features considered

B.1 Candidate features related to document meta-data

Feature name	**Explanation**
Text Length	The number of characters in the analyzed text
Light Entities	An entity with a unique name
Heavy Entities	An entity that requires name resolution before database lookup
References	Very roughly, a reference is an interesting event found in the analyzed text
Max Offset	The largest offset, counting from the start of the document, where a reference was found
Event type, other	If all attempts to classify an event into one of the pre-defined types fail, the event is assigned type "other"
Language	The language the document was written in
Weekday	Which weekday the analysis was performed on
Hour of day	Which hour of the day the analysis was performed on
Author	The author of a document - for instance, the twitter account if the document was a tweet. Not always present.
Medusa	A process that performs additional natural language processing. Not always used.

B.2 Features related to system state

B.2.1 Amazon EC2 - server health

Feature	Explanation
CPU-usage	How much load the CPU is under. Detailed further in Theory
Network traffic - in	Number of bytes received on all network interfaces
Network traffic - out	Number of bytes sent on all network interfaces
Disk Operations - read	Number of I/O read-operations performed
Disk Operations - write	Number of I/O write-operations performed

B.2.2 Java Virtual Machine

Feature	Explanation
Marksweep - Time	Time spent running the computationally intensive Marksweep garbage collection algorithm
Marksweep - Count	Amount of memory freed up by marksweep
Scavenge - Time	Time spent running the lighter scavenge garbage collection algorithm
Scavenge - Count	Amount of memory freed up by scavenge

Table B.1: Garbage collection

Feature	Explanation
Code Cache Memory	Memory used to store compiled Java code
Compressed Class Space	Memory used for pointers to Java classes
Meta Space	A type of memory used by the Java Virtual Machine since Java 8
Non-heap Memory	Memory used for per-class structures, such as field and method data
PsEden	Memory used to store newly created objects
PsOldGen	Memory used to store older objects
PsSurvivor	Objects that have survived a round of garbage collection
Heap Memory	Memory used by dynamically allocated variables

Table B.2: Memory

Feature	Explanation
Thread Count	Number of virtual threads started by the Java Virtual Machine since initiation of the process

Table B.3: Other

B.2.3 Other

Feature	Explanation
Database operations	Number of reads- and writes to the database

C

Data retrieval - approach and limitations

C.1 Data for M1 - Document metadata

Checkpoint data - containing the document ID and corresponding processing time - is stored for 30 days. Document metadata is perpetually available once a document has been analysed, resulting in tens of billions of data points being available for study. However, due to limitations imposed by RF, no more than around 100.000 retrievals per week is possible. This is by far the largest data set used for any model.

C.2 Data for M2 - Document metadata and system performance metrics

Metrics about EC2-instance performance is stored in CloudWatch for one week. Internal system metrics (such as JVM-performance) are available in Graphite for one month. For a data point to be considered valid, *all* performance metrics had to be available. Due to technical issues leading to frequent collection errors, a large portion of the data had to be discarded because of this requirement - of 100.000 documents, around 5000 would have matching performance metrics.

C.3 Data for M3 - Metadata of documents analysed in parallel

Like in M1, only document metadata was required for the data set. However, each document could potentially have hundreds of other documents analysed in parallel on the same EC2-instance, depending on the processing time. Since a data set containing 1000 data points would then necessitate the fetching of hundreds of thousands of documents, at most twenty parallel documents were considered. Even with this restriction, the data set used for this model had to be far smaller than the one used for M1.

C.4 Data for M4 - Time-series data

As the only feature considered here was the document processing time, all of the required information could be found in the Checkpoint database. The same set as in M1 could not be used as those documents were sampled randomly from an interval of thirty days, and the data points have to be temporally ordered when performing time series analysis.
Fetching a consecutive series of documents from individual instances was simple to do technically, but the length of the series was limited by the amount of time the instance had been online. The system-wide investigation could, in theory, have used all the data available for the last thirty days for training and testing. However, calculating the average of all document processing times across all instances required a considerable amount of computing. Thus, the amount of considered one- and ten-minute intervals had to be limited.

C.5 Data for M5 - Document metadata and previous document processing time

The same difficulties encountered in M1 and M4 applied here. This was the hardest set to construct from a technical point of view. For one data point, the following steps had to be performed:

- Fetch an entry from the Checkpoint database, containing document ID, time of analysis, EC2-instance ID, and the resulting DPT. This document will be referred to as the *target*
- Retrieve the document metadata, using the document ID
- Using the EC2-instance ID and time of analysis, retrieve a number of checkpoint entries containing documents analysed just before the target.
- Retrieve the document metadata for these previously analysed documents
- Using the time of analysis, compute the average DPT across all EC2-instances during the previous one- and ten-minute interval.

D

Neural Network Structure

D.1 Document metadata

Layer	Type	Neurons	Kernel Initializer	Activation function
Input	-	5	-	-
Hidden 1	Dense	32	Glorot/Xavier uniform	ReLU
Hidden 2	Dense	64	Glorot/Xavier uniform	ReLU
Hidden 3	Dense	64	Glorot/Xavier uniform	ReLU
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.1: The Neural Network structure for model one

Training Set Size	108,000
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	20
Batch Size	16

Table D.2: The Neural Network parameters for model one

D.2 Document metadata and system state

Layer	Type	Neurons	Kernel Initializer	Activation function
Input	-	120	-	-
Hidden 1	Dense	256	Glorot/Xavier uniform	ReLU
Hidden 2	Dense	512	Glorot/Xavier uniform	ReLU
Hidden 3	Dense	512	Glorot/Xavier uniform	ReLU
Hidden 4	Dense	256	Glorot/Xavier uniform	ReLU
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.3: The Neural Network structure for model two

Training Set Size	4,500
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	60
Batch Size	16

Table D.4: The Neural Network parameters for model two

D.3 Document metadata and total analyzer workload

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	15	-	-
Hidden 1	Dense	64	Glorot/Xavier uniform	ReLU
Hidden 2	Dense	128	Glorot/Xavier uniform	ReLU
Hidden 3	Dense	128	Glorot/Xavier uniform	ReLU
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.5: The Neural Network structure for model three

Training Set Size	2,200
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	80
Batch Size	16

Table D.6: The Neural Network parameters for model three

D.4 Time series prediction - Average DPTs

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	1	-	-
Hidden 1	LSTM	64	Glorot/Xavier uniform	tanh
Hidden 2	LSTM	32	Glorot/Xavier uniform	tanh
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.7: The Neural Network structure for model 4a

Training Set Size	6,500
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	15
Batch Size	32

Table D.8: The Neural Network parameters for model 4a

D.5 Time series prediction - Weighted Average DPTs

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	1	-	-
Hidden 1	LSTM	64	Glorot/Xavier uniform	tanh
Hidden 2	LSTM	32	Glorot/Xavier uniform	tanh
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.9: The Neural Network structure for model 4b

Training Set Size	4,800
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	15
Batch Size	32

Table D.10: The Neural Network parameters for model 4b

D.6 Time series prediction - Average DPTs All Instances

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	1	-	-
Hidden 1	LSTM	64	Glorot/Xavier uniform	tanh
Hidden 2	LSTM	32	Glorot/Xavier uniform	tanh
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.11: The Neural Network structure for model 4c

Training Set Size	3,000
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	15
Batch Size	32

Table D.12: The Neural Network parameters for model 4c

D.7 Document metadata and previous document processing times - Instance Level

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	17	-	-
Hidden 1	Dense	128	Glorot/Xavier uniform	-
Batch Normalization 1	Batch Normalization	-	-	-
Activation 1	Activation	-	-	ReLU
Hidden 2	Dense	256	Glorot/Xavier uniform	-
Batch Normalization 2	Batch Normalization	-	-	-
Activation 2	Activation	-	-	ReLU
Hidden 3	Dense	256	Glorot/Xavier uniform	-
Batch Normalization 3	Batch Normalization	-	-	-
Activation 3	Activation	-	-	ReLU
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.13: The Neural Network structure for model 5a

Training Set Size	30,000
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	50
Batch Size	16

Table D.14: The Neural Network parameters for model 5a

D.8 Document metadata and previous document processing times - System Wide

Layer	**Type**	**Neurons**	**Kernel Initializer**	**Activation function**
Input	-	7	-	-
Hidden 1	Dense	64	Glorot/Xavier uniform	ReLU
Hidden 2	Dense	128	Glorot/Xavier uniform	ReLU
Hidden 3	Dense	128	Glorot/Xavier uniform	ReLU
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.15: The Neural Network structure for model 5b

Training Set Size	5,500
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.001
Epochs	40
Batch Size	16

Table D.16: The Neural Network parameters for model 5b

D.9 Job metadata (aggregated documents)

Layer	Type	Neurons	Kernel Initializer	Activation function
Input	-	6	-	-
Hidden 1	Dense	32	Glorot/Xavier uniform	-
Batch Normalization 1	Batch Normalization	-	-	-
Activation 1	Activation	-	-	ReLU
Dropout 1	Dropout	0.8 [ratio]	-	-
Hidden 2	Dense	64	Glorot/Xavier uniform	-
Batch Normalization 2	Batch Normalization	-	-	-
Activation 2	Activation	-	-	ReLU
Dropout 2	Dropout	0.8 [ratio]	-	-
Hidden 3	Dense	64	Glorot/Xavier uniform	-
Batch Normalization 3	Batch Normalization	-	-	-
Activation 3	Activation	-	-	ReLU
Dropout 3	Dropout	0.8 [ratio]	-	-
Output	Dense	1	Glorot/Xavier uniform	Sigmoid

Table D.17: The Neural Network structure for model 5a

Training Set Size	100,000
Loss Function	Mean Absolute Error
Optimizer	Adam
Learning Rate	0.01
Epochs	40
Batch Size	32

Table D.18: The Neural Network parameters for model 5a

E

Detailed Time-Series Prediction Results

E.1 Series Two

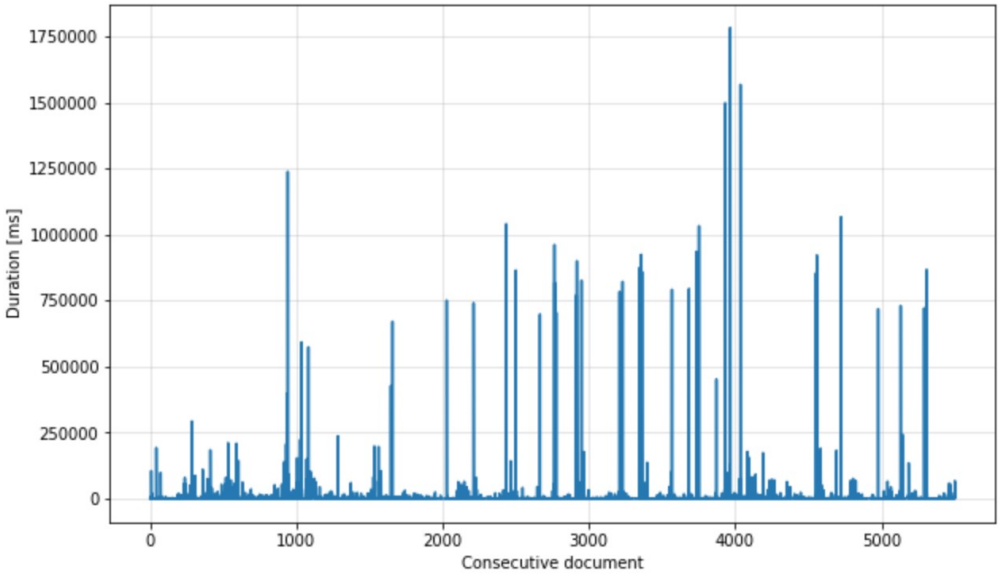

Figure E.1: Consecutive document processing times (series two)

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	5,504	4	1,785,399	10,767	915	77,533	24,145
Validation	1,101	5	1,067,726	8,812	812	68,701	17,354

Naive MAE	Naive MAE / Mean	Naive MedAE	Naive MedAE / Median
14,413	1.63	501	0.61

Model MAE	Model MAE / Mean	Model MedAE	Model MedAE / Median
5,047	0.57	319	0.29

E.2 Series Three

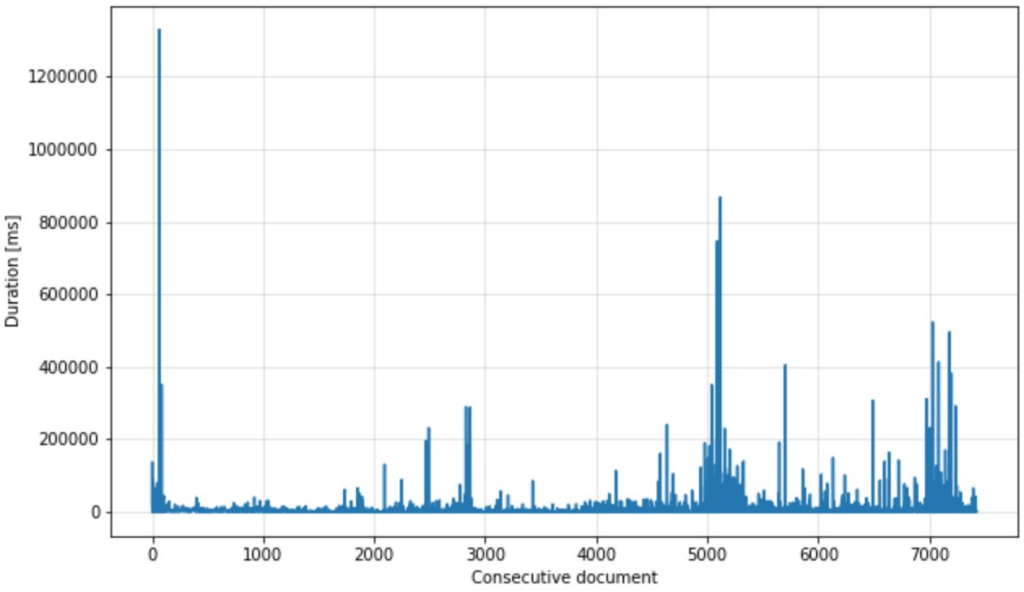

Figure E.2: Consecutive document processing times (series three)

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	7,427	3	1,329,210	6,194	990	29,544	22,301
Validation	1,486	4	523,316	8,757	987	32,133	35,096

Table E.1: Series Three

Naive MAE	Naive MAE / Mean	Naive MedAE	Naive MedAE / Median
11,906	1.35	782	0.79

Model MAE	Model MAE / Mean	Model MedAE	Model MedAE / Median
3,290	0.37	308	0.59

E.3 Series Four

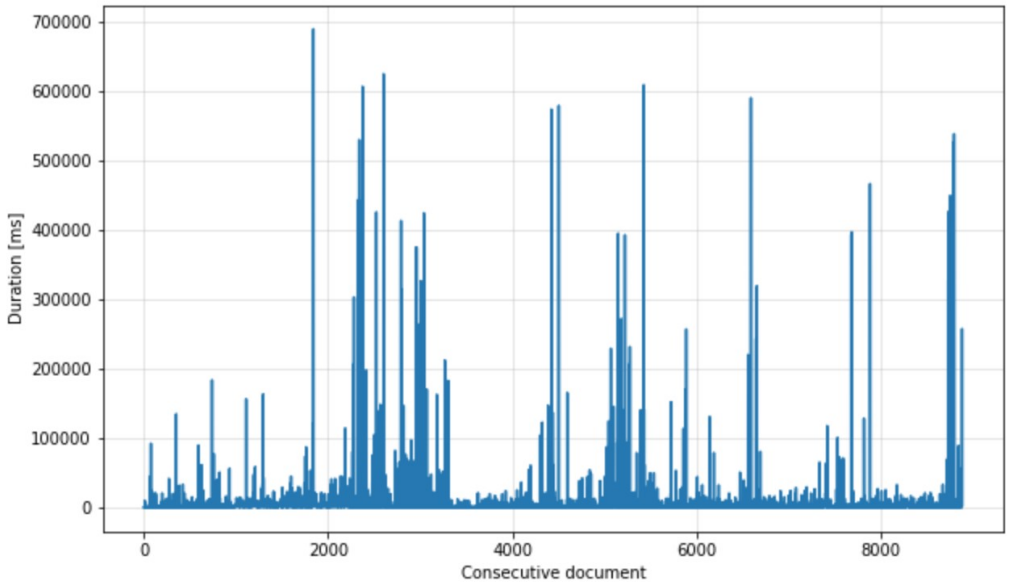

Figure E.3: Consecutive document processing times (series four)

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	8,879	3	689,101	6,330	1,002	31,780	19,721
Validation	1,776	3	538,040	6,021	1,331	35,431	12,592

Naive MAE	Naive MAE / Mean	Naive MedAE	Naive MedAE / Median
8,026	1.33	1,026	0.77

Model MAE	Model MAE / Mean	Model MedAE	Model MedAE / Median
4,302	0.71	624	0.46

E.4 Series Five

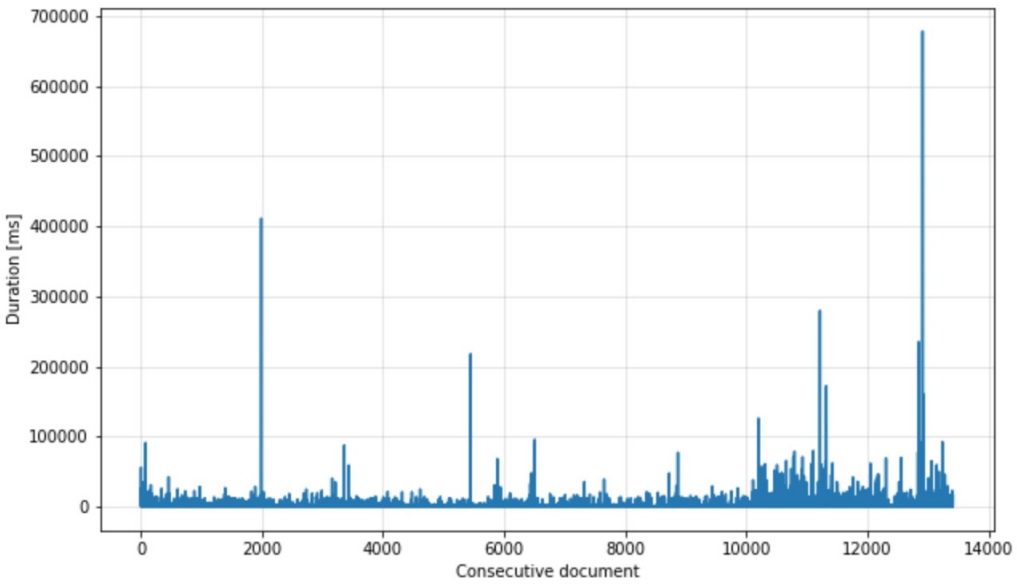

Figure E.4: Consecutive document processing times (series five)

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	13,412	2	678,196	2,162	600	10,044	11,382
Validation	2,683	3	678,196	4,258	681	18,187	19,709

Naive MAE	Naive MAE / Mean	Naive MedAE	Naive MedAE / Median
6,075	1.42	415	0.6

Model MAE	Model MAE / Mean	Model MedAE	Model MedAE / Median
3,920	0.92	412	0.61

E.5 Weighted Series Two

The same data as series two was used.

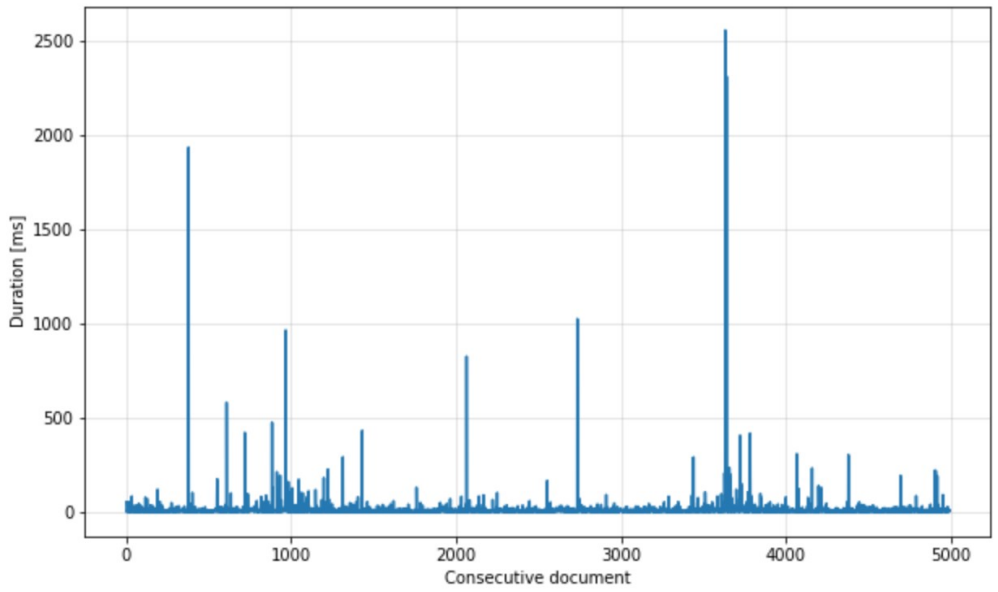

Figure E.5: Plot of the second time series, with each document processing time divided by its text length

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,022	0.11	505	0.62
Naive	9,800	1.11	4,216	5.19

Table E.2: Weighted Series Two Results

E.6 Weighted Series Three

The same data set as series three was used

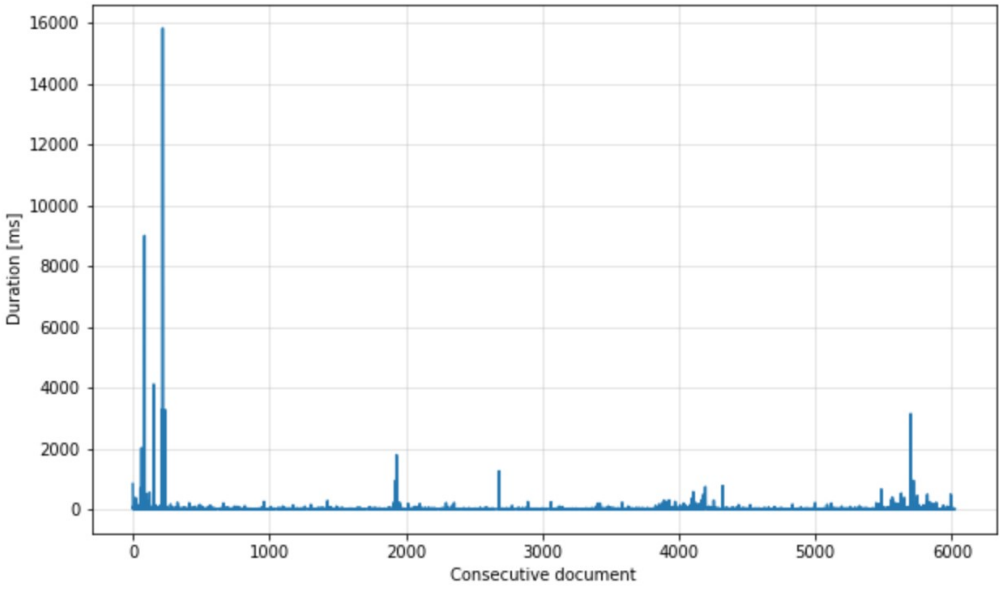

Figure E.6: Plot of the third time series, with each document processing time divided by its text length

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	6,171	0.71	1,274	1.29
Naive	39,891	4.55	7,360	7.46

Table E.3: Weighted Series Three Results

F

Examining processing times when resources were manually over-provisioned

F.1 Methodology of the experiment

An auto scaling system is often told to scale up, i.e. start new machines, when currently running machines cannot handle the load according to some predefined rule like having too many messages in a queue for too long. This scale up is meant to make the system perform better in terms of for example handling more requests, incoming messages or process data faster. In some cases, the trigger of a scale up is caused after some time of running at under capacity, and the start of new machines can take a couple of seconds to a few minutes. Thus, there can be a delay between the time of added computing power and the time of actual need for it. This delay can cause the system to run at a sub-optimal rate, potentially affecting the data processing performance because of queue build ups or very high CPU-load.

In order to test for a performance increase in document analysis processing times, an experiment was run where the analysis chain was run at over capacity. During half a day, the amount of consumers, threads which performs the document analysis on individual documents, was set to a constant amount. Instead of having the system scale up the amount of consumers according to the normal rules, the amount of consumers was set to the max limit. By doing so, each machine would, on average, run at lower capacity than in normal when the amount of machines running is minimized. The CPU load of each machine should then be lower, meaning that if the CPU load plays a large role in the document analysis performance, the DPT would potentially be more consistent.

If the processing time of the document analyzer is shown the be more consistent, the hypothesis was that using data from this interval of running at over capacity would produce more accurate models for prediction. For example, training a model on document metadata where the processing time depends more on the document metadata than on CPU load, because now CPU load plays a smaller role, would be much better since the document metadata is the only input. In practice, it might not be feasible for a system to run at max capacity all the time, but this experiment could help point in the direction of which to further investigate what causes the

large fluctuations in processing time.

F.1.1 Models evaluated

Model 1 - the **Document metadata** based model was evaluated with this new set of documents. The reason being that this is considered the simplest model and easiest to implement in a production environment. Furthermore, part of the reason for the experiment with over capacity was to see whether it would be possible to just use the document metadata, which this model does, and get more accurate predictions.

Model 4a - the **Instance-level time-series data** based model. This is the model that uses functions of previous processing times as input features together with the document metadata in order to perform predictions. If the previous processing times show to play a role in future processing times, and if processing time are now more consistent, this model has potential, why it was selected in this experiment.

Model 4c - the **System-wide time-series data** based model. Similar to model 4a, but more realistic for actual implementation since it considers the whole system and since there currently is no way of knowing which machine will acquire which document.

Model "job" - the model for **Job-level** investigation. Since the setup of this model is the closest to the initial goal of being able to accurately predict on complete jobs, it would be unwise not to include it. Furthermore, the potentially increased consistency in processing times could benefit this model's accuracy as it is considered "one big document".

F.2 Results

The average DPT during the six hour interval, in which the system was run at over capacity, was 3355 ms. The average DPT during the entire week prior to the experiment was 5260 ms. The same interval of six hours, but during the days prior and after to the experiment had an average DPT of 5928, 4344, and 2744 ms.

F.2.1 Document metadata predictions

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Total	134,000	29	1,433,558	2,825	680	15,756	11,829
Validation	26,000	33	1,433,558	2,819	673	17,846	11,495

Table F.1: Statistical data on the DPT in the data set used for Model one while the system was run at over capacity. Values are in milliseconds.

The prediction error is available in table F.2

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,137	0.7032	168	0.3820
Naive	2,139	1.3395	273	0.6708

Table F.2: Model prediction errors. Values are in milliseconds.

F.2.2 Time series prediction

Three instances active during the interval where resources were over-provisioned were selected at random, and a consecutive series of documents was fetched from each of them. A recurrent neural network was then trained and tested on each series. Statistical information about each data series and the prediction results are available in Table F.3 - F.8

F.2.2.1 Series one

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	14,916	2	528,136	2,258	674	10,629	9,091
Validation	2,984	2	528,136	1,983	759	10,404	8,254

Table F.3: Statistical data on the DPT in the first data set used for time series prediction while the system was run at over capacity

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,601	0.80	418	0.55
Naive	2,120	1.06	435	0.57

Table F.4: Prediction error when using instance-level previous DPT to predict future DPT, while the system was run at over capacity. First data set.

F.2.2.2 Series two

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	4,453	4	290,511	2,527	733	9,482	11,541
Validation	891	5	157,321	1,951	732	6,952	7,410

Table F.5: Statistical data on the DPT in the second data set used for time series prediction while the system was run at over capacity

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	857	0.43	249	0.34
Naive	2,098	1.07	458	0.62

Table F.6: Prediction error when using instance-level previous DPT to predict future DPT, while the system was run at over capacity. Second data set.

F.2.2.3 Series three

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	12,352	2	264,341	2,152	697	7,914	8,302
Validation	2,471	46	93,706	1,667	708	4,127	6,771

Table F.7: Statistical data on the DPT in the third data set used for time series prediction while the system was run at over capacity

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	509	0.30	201	0.28
Naive	1,693	1.01	436	0.61

Table F.8: Prediction error when using instance-level previous DPT to predict future DPT, while the system was run at over capacity. Third data set.

F.2.3 Predicting average DPT during intervals

To investigate how the average DPT fluctuated during the interval where resources were over-provisioned, a series of one- and ten-minute intervals and their corresponding average DPT was retrieved and used as input data to the RNN.

F.2.3.1 One-minute intervals

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	359	1,409	12,964	3,163	2,764	1,556	6,253
Validation	72	1,801	5,768	2,723	2,690	636	3,400

Table F.9: Statistial data on series of one-minute intervals when system was run at over capacity

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	1,951	0.71	1,926	0.71
Naive	409	0.15	351	0.13

Table F.10: Prediction results from over capacity across all instances (average DPT), one-minute intervals

F.2.3.2 Ten-minute intervals

Data Set	Sample Size	Min	Max	Mean	Median	STD	95 perc
Total	35	2,014	8,465	3,222	2,831	1,354	5,598
Validation	7	2,153	4,635	3,035	2,834	787	4,342

Table F.11: Statistial data on series of ten-minute intervals when system was run at over capacity

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained	3,144	1.03	3,164	1.11
Naive	627	0.20	369	0.13

Table F.12: Prediction results from overcapacity across all instances (average DPT), ten-minute intervals

F.2.4 Job metadata predictions

Since the hypothesis was that when running at over capacity, the DPT would mainly be more consistent, validation data from the same interval must also be used. So in order to validate using actual jobs, which was done in the normal case, new data was fetched from jobs which had been run during this time.

Due to the small window of six hours, not many jobs had been run that also filled the criteria of having at least 100 and at most 10,000 related documents. Thus, the validation size was very low: 8 jobs. This validation set is noted below as *Validation 1*, while the original data set of 53 jobs was also used as a comparison, and is noted below as *Validation 2*.

Data set	Sample size	Min	Max	Mean	Median	STD	95 perc
Train	100,000	244	317,005,220	129,126,439	113,731,294	92,018,105	285,348,824
Validation 1	8	1,606,977	378,072,475	88,163,933	9,316,055	132,973,023	328,852,111
Validation 2	53	3,765,619	240,370,376	26,862,209	15,668,748	41,302,942	68,427,973

Table F.13: Functions of the processing duration of documents in the data set. Values are in milliseconds.

From the models output predictions, the following values were calculated:

Model	MAE	MAE / Mean	MedAE	MedAE / Median
Trained - validation 1	6,717,717	0.0590	4,439,659	0.3674
Trained - validation 2	11,882,846	0.3514	7,505,308	0.3142
Naive	111,403,845	1.2685	8,796,667	0.9326

Table F.14: Model prediction errors. Values are in milliseconds.

F.3 Summary and analysis of the experiment

Model no.	**1**	**4a**	**4c**
MAE / Mean [%]	70	80	71
Change from standard	+7	+17	+13

Table F.15: System overcapacity - summary of results

Number	**Name**
1	Document metadata
4a	Time series, same instance
4c	Time Series, predicting average processing time across system

Table F.16: Model numbers and their names

As hypothesized, the first model that considers only document metadata performs better, albeit not by much. The results from using previous DPT's to predict future ones are worse, indicating that increasing the amount of available computational power does not necessarily reduce the fluctuations that can be observed.