Modelling large-scale scientific data transfers

Joaquin Bogado
Supervisor: Mario Lassnig
Supervisor: Javier Díaz
Scientific Advisor: Fernando Monticelli

May 13, 2021

A mis Amigues @ CERN.
To my Friends @ CERN.
À mes Amis @ CERN.
An meine Freunde @ CERN.
Ai miei Amici @ CERN.
Στους φίλους μου @ CERN.
Моим друзьям @ CERN.
Aos meus Amigos @ CERN.

A Gabi, que me acompaña
siempre de cerca o de lejos.

A Emil, que llegues a donde
quieras llegar.

Contents

Chapter 1

Introduction

The Road goes ever on and on
Out from the door where it began.

Bilbo Baggins

1.1 Motivation

The ability to predict the transfer times of files that move through the network could lead to significant improvements in job scheduling and equally significant improvements in storage resource management in the Worldwide LHC Computing Grid (WLCG).

This work focuses on the creation and study of a publicly available dataset [1], from which it is possible to reconstruct the internal state of part of the WLCG's distributed data management system. The work seeks to investigate if there are viable models that allow predicting the transfer time of files and groups of files called rules with sufficient precision to improve existing scheduling systems.

Rucio is the scientific data management system of the ATLAS Experiment at CERN, service that tracks data placement and movement across the WLCG and the source of the data of the mentioned dataset. The size of the files, number of accesses, creation, submission, starting, and ending timestamps of the requests are stored in the Rucio database, both to comply with the data retention policy and with monitoring purposes. The Rucio ATLAS instance processed transfer requests at a rate of 25 Hz during July 2019, totalling more than 788 million transfer requests. This number does not include the deletion requests, also processed by Rucio.

There have been efforts in order to model data transfers in the WLCG since the Rucio commissioning at the end of 2014. The work cited in [2] focuses on Transfers Time To Complete (TTC) predictions. The work cited in [3] focuses on the prediction of the network throughput. The work cited in [4] focuses on the prediction of the length of the queues of the system, with emphasis on the importance of network throughput. However, prior studies have failed to delves into Rucio's replication rules modeling and Rules TTC prediction.

The ability to predict the ending time of a transfer after its creation allows to make scheduling decisions early in the lifetime of the transfer or group of transfers, called rules. We call this *Transfers TTC* and *Rules TTC*. Better scheduling techniques is expected to lead to network and storage optimization.

Rucio stores 4 timestamps related to transfer requests states. Creation time, submission to transfer tool time, network starting time, and ending time. These data allow us to reconstruct the lifetime of each transfer processed by the system in a detailed way, at a time resolution of seconds. Rucio also stores the rule id of the transfer, associated to groups of transfers that belongs to the same dataset. This should allow us to group the transfers associated to a rule, and to reconstruct the rule life time.

1.2 Research questions

The main research question of this work is if is it possible to predict the Rule TTC of a given rule using the data available in Rucio at its creation time? A model able to predict the Rule TTC when the rules are created, within 10% of relative error and more than 90% of the times is considered by the experts as good enough to be used in scheduling tasks.

The main hypothesis is that given the set of variables `account`, `state`, `activity`, `SIZE`, `id`, `rule_id`, `external_host`, `src_rse`, `dst_rse`, `previous _attempt_id`, `retry_count`,`created`, `submitted`, `started` and,`ended`, stored in the Rucio database, it is possible to reconstruct the past state of the distributed data management system with resolution of seconds. These data are enough to predict the Rules TTC at rule creation time within 10% relative error more than 90% of the time, and the individual Transfers TTC at request creation time, also within similar boundaries.

1.3 Research outline

This work studies the Rucio transfers requests data of the REQUESTS_HISTORY table in the Rucio database. The studied data comprehends transfers created during the 2 months of June and July of 2019. The data from the last 6 months of Rucio operation is available upon request from the ATLAS Experiment. The data older than 6 months does not comply with the retention policy and is deleted. To allow further analysis and replication of this work, data was extracted and made publicly accessible [1].

Chapter 2, starting on page 8, describes the studied system in detail. The Rucio system and its interaction with File Transfers System version 3 (FTS) are highlighted. Naming conventions and acronyms used in this work are presented. A Glossary with common terms and abbreviations can be found in page 93. The most important aspects of the FTS system are also covered.

In Chapter 3, starting on page 17, the data analyzed in this work is presented. The dataset extraction, cleaning, and transformation methods are explained. The calculation of derived attributes like network time, FTS queue time, and Rucio time, as well as transfer lifetime reconstruction are discussed. Metrics to determine the accuracy of the models are presented. Special emphasis is put on the FoPG metric.

In Chapter 4, starting on page 30, a novel family of models α and $\alpha\prime$ are presented. These models for Rule TTC prediction are based on intra rule transfer time extrapolation. Model accuracy is discussed. The suitability of the model is put into question.

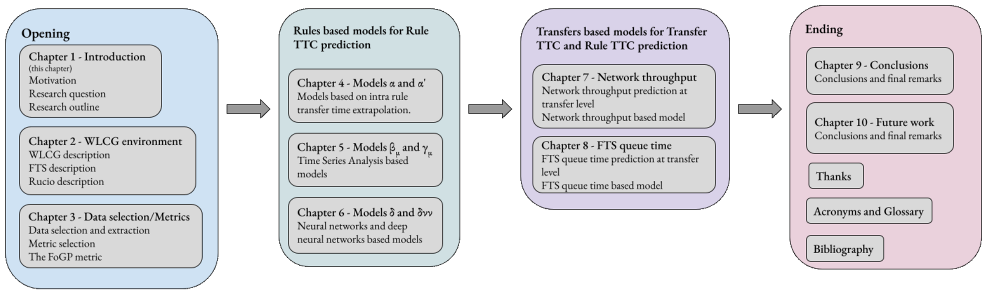
Time series analysis techniques are used on Chapter 5, starting on page 40 to study the time dependency of the Rule TTC. The novel β_μ model family is featured. Model β_μ requires the μ function of the Rule TTC time series to be known. These data is not available in Rucio Database at rule creation time, and thus β_μ is considered a theoretical model. The novel model family γ_μ, also presented in this chapter, utilizes time series analysis techniques to forecast the value of the μ function from data available in the Rucio database at rule creation time. The performance of the models are evaluated and the bounds in accuracy imposed by the theoretical model is discussed. The trivial κ model, the model that predicts a constant Rule TTC is presented, in order to have a valid baseline model other models can be compared to.

In Chapter 6, starting on page 51, the novel δ and $\delta\nu\nu$ models based on neural networks are presented. This chapter describes the methods and tools used to train the models, as well as the data transformations needed to apply the techniques. Model δ is a model adapted from literature to forecast the Rule TTC based on time series in a similar fashion as β_μ. Model $\delta\nu\nu$ is a novel multi-input deep neural network model, that next to the time series used for the δ model, includes information of the current state of the system reconstructed from the data studied in this work. Performance of δ, $\delta\nu\nu$, and κ models are compared, being the last one, a trivial model that predict a constant time for every Rule TTC.

In Chapter 7, starting on page 62, presents a novel model for Rule TTC prediction based on network properties inferred from the Rucio data. The model is based on an equation to predict the transfers rate based on the size of those transfers. The FoGP metric is used to evaluate the model in comparison with previous models. A proof that the perfect network time prediction model for transfers can play an important role in models for Rule TTC predictions is presented.

In Chapter 8, starting on page 74, a novel model for FTS Queue Time prediction based on simulation and calculation of internal states of FTS system is presented. Availability of the data to feed the model and possibility to simulate in real time is discussed. A similar proof to the one presented in Chapter 7 is presented, to show the relevance of FTS queue time of the transfers in the prediction of Rules TTC.

In Chapter 9, starting on page 82, the main conclusions of this work are presented, as well as possible future research lines and extensions of this work are listed in Chapter 10, starting on page 90.


Opening
Chapter 1 - Introduction
(this chapter)
Motivation
Research question
Research outline
Chapter 2 - WLCG environment
WLCG description
FTS description
Rucio description
Chapter 3 - Data selection/Metrics
Data selection and extraction
Metric selection
The FoGP metric
Rules based models for Rule TTC prediction
Chapter 4 - Models α and α′
Models based on intra rule transfer time extrapolation.
Chapter 5 - Models β_μ and γ_μ
Time Series Analysis based models
Chapter 6 - Models δ and δνν
Neural networks and deep neural networks based models
Transfers based models for Transfer TTC and Rule TTC prediction
Chapter 7 - Network throughput
Network throughput prediction at transfer level
Network throughput based model
Chapter 8 - FTS queue time
FTS queue time prediction at transfer level
FTS queue time based model
Ending
Chapter 9 - Conclusions
Conclusions and final remarks
Chapter 10 - Future work
Conclusions and final remarks
Thanks
Acronyms and Glossary
Bibliography

Thesis outline

Chapter 2

The distributed data management environment

¡Oh memoria, enemiga mortal
de mi descanso!

Miguel de Cervantes Saavedra

2.1 The World LHC Computing Grid

The transfers studied in this work take place over the World LHC Computing Grid (WLCG) infrastructure. The following section describes the what is WLCG and remarks the aspects that are more relevant for this study.

The Worldwide LHC Computing Grid [5] is a global computing infrastructure whose mission is to provide computing resources to store, distribute and analyze the data generated by the Large Hadron Collider (LHC), making these data equally available to all partners, regardless of their physical location.

WLCG is the world's largest computing grid. It is supported by many associated national and international grids across the world, such as the European Grid Initiative (Europe-based) and the Open Science Grid (US-based), as well as many other regional grids.

WLCG is coordinated by CERN. It is managed and operated by a worldwide collaboration between the experiments (ALICE, ATLAS, CMS and LHCb) and the participating computer centers. It is reviewed by a board of delegates from partner country funding agencies, and scientifically reviewed by the LHC Experiments Committee. WLCG computing enabled physicists to announce the discovery of the Higgs Boson on 4 July 2012.

The four main component layers of the WLCG are networking, hardware, middleware and physics analysis software.

WLCG is organized in four layers, or "tiers"; 0, 1, 2 and 3. Each tier provides a specific set of services. Except for Tier 0 which is unique and CERN based, each tier is composed on many sites or data centers. A site usually represents a set of resources pledge by an institution.

The Tier 0 is the CERN Data Center, which is located in Geneva, Switzerland. All data from the LHC passes through the central CERN hub, but CERN only provides around 20% of the total compute capacity. Tier 0 is responsible for the safe-keeping of the raw data, first pass reconstruction, distribution of raw data and reconstruction output to the Tier 1s, and reprocessing of data during LHC down-times.

The Tier 1s are thirteen large computer centers with sufficient storage capacity and with round-the-clock support for the grid. They are responsible for the safe-keeping of a proportional share of raw and reconstructed data, large-scale reprocessing and safe-keeping of corresponding output, distribution of data to Tier 2s and safe-keeping of a share of simulated data produced at these Tier 2s.

The Tier 2s are typically universities and other scientific institutes, which can store sufficient data and provide adequate computing power for specific analysis tasks. They handle analysis requirements and proportional share of simulated event production and reconstruction. There are currently around 160 Tier 2 sites covering most of the globe.

Individual scientists will access these facilities through local, also sometimes referred to as Tier 3, computing resources, which can consist of local clusters in a university department or even just an individual PC. There is no formal engagement between WLCG and Tier 3 resources.

WLCG can initiate the distribution of data to the hundreds of collaborating institutes worldwide thanks to the excellent connectivity and dedicated networking infrastructure set up at CERN and subsequently worldwide. CERN has its own Internet Exchange Point (IXP). It was set up in 1989 to be able connect directly to major national and international networks. This helps to reduce costs, time and the number of different networks, the number of *hops* the data needs to pass through into order to reach its destination.

CERN's Tier-0 can take advantage of CERN's own internet exchange point, the CIXP to pass data straight onto the dedicated networks for global exchange. Connectivity between WLCG Sites usually is through a dedicated link.

CERN is connected to each of the Tier 1s around the world on a dedicated, private, high-bandwidth network called the LHC Optical Private Network (LHCOPN). This consists of optical-fiber links working between 10 to 100 gigabits per second, spanning oceans and continents.

Exchanging data between the WLCG centres is managed by the File Transfer Service or FTS, initially developed together with the Enabling Grids for E-science projects from 2002 onward. It has been tailored to support the special needs of grid computing, including authentication and confidentiality features, reliability and fault tolerance, third party and partial file transfer.

Each grid center manages a large collection of computers and storage systems. Installing and regularly upgrading the necessary software manually is labor intensive, so large-scale management systems, some such as Quattor, developed at CERN, automate these services. They ensure that the correct software is installed from the operating system all the way to the experiment-specific physics libraries, and make this information available to the overall grid scheduling system, which decides which centers are available to run a particular job. Each of the Tier 1 centers also maintains disk and tape servers. These centers use specialized storage tools such as the dCache system developed at the Deutsches Elektronen Synchrotron (DESY) laboratory in Germany, the ENSTORE system at Fermilab in the US or the CERN advanced storage system (CASTOR) and EOS developed at CERN to allow access to data for simulation and analysis independent of the medium, tape or disk, that the information is stored on.

Middleware is the software infrastructure which allows access to an enormous amount of distributed computing resources and archives, and is able to support powerful, complicated and time-consuming data analysis. This software is called ”middleware” because it sits between the operating systems of the computers and the physics applications software that can solves a scientist’s particular problem.

The most important middleware stack used in the WLCG are from the European Middleware Initiative, which combines several middleware providers (ARC, gLite, UNICORE and dCache); the Globus Toolkit developed by the Globus Alliance; and the Virtual Data Toolkit.

To analyse the enormous amount of data that the LHC produces, physicists need software tools that go beyond what is commercially available. The immense and changing demands of the high energy physics environment require dedicated software to analyze vast amounts of data as efficiently as possible.

The main physics analysis software is ROOT, a set of object-oriented core libraries used by all the LHC experiments. It is a versatile open-source tool, developed at CERN and Fermilab (USA), and used for big data processing, statistical analysis, visualization and storage.

2.2 The File Transfer Service

The File Transfer Service version 3[6] is the latest version of the service responsible for globally moving the majority of the LHC data across the WLCG infrastructure.

It has been designed in a modular and extensible way to allow good scalability. FTS is responsible for moving the majority of LHC data across WLCG infrastructure, which it provides with reliable, multi-protocol (GridFTP, SRM, HTTP, xroot), adaptively optimized data transfers. Core functionality of FTS3 is extended with various Web-oriented tools like versatile monitoring and WebFTS user interface with support of Federated Identity.

According to its public site[7], during 2019, 23 FTS instances were responsible to move 950 PB of data across WLCG data centers for 30 different experiments, also called Virtual Organizations (VOs).

WebFTS is a web interface that provides a file transfer and management solution in order to allow users to invoke reliable, managed data transfers on distributed infrastructures. FTS provides a Python REST API through it integrates with frameworks and a CLIs for copying files from one site to another among a monitoring for several profiles: General monitoring, Grafana, for end users, Discovery Data, Kibana, for researches and Service Specific, ftsmon/Kibana, for service managers.

An important part of FTS is its optimizer that makes it possible to run transfers between any two random endpoints with good reliability and performance with zero configuration by default. The FTS optimizer has an impact on how much time take for an individual transfer to start and eventually, to finish, and thus it is discussed later in this work.

FTS also support a plugin based library for file manipulation supporting multiple protocols, Webdav/https, GridFTP, xroot, and SRM. This allows users and other tools, specially Rucio, to move files between endpoints independently of the protocol the endpoints accept.

Many instances of FTS are installed in several data centers across the WLCG, and as this instances usually give services to different experiments or Virtual Organizations (VOs). The study the transfers between grid endpoints for a single VO is full of caveats that will be discussed in this work.

It is important to notice that while FTS manages file transfers between grid sites, FTS does not keep track of where the files are, nor even if the files exist. It is the responsibility of the users and the experiments to catalog, curate, organize, and delete files they don't need anymore. That is the main reason of Rucio's existence, to act as the file catalog, allowing access and location for every single file across the grid.

2.3 Rucio

Rucio is the ATLAS Experiment solution to keep track of the experiment files and datasets. Its development started in early 2012 to cope with the more demanding requirements of LHC Run 2 in late 2014, and to address the lack of scalability of it is predecessor, Don Quixote 2 (DQ2). Rucio was designed from scratch to provide not only file catalog for ATLAS experiment, but a comprehensive tool for distributed data management, allowing the experiment to set and enforce a wide variety of policies. Through a complex system of rules and subscriptions, the ATLAS data management team is able to set how many copies of what files or group of files should be available in each WLCG site and for how long should be there. Rucio also generates automatic deletion orders directly to the endpoints for those files that are not needed anymore, which i reduces the person power needed for system operations, and to avoid the ATLAS portion of WLCG storage to become completely full.

Rucio does not do file transfers. Instead, it delegates the actual transfer of files between WLCG endpoints to the several instances of the File Transfers System FTS[6] transfer tool. The instance selected to do the transfers of the files depends on the destination of the file transfer request. These instances operate at the WLCG level and serve transfers from several VOs and not only ATLAS specific transfers. The Rucio database does not contain information about the transfer tool other than which FTS instance that is used for each transfer request. Information about FTS queues state, scheduling and retries, number of nodes, and configuration are hidden from Rucio. Transfers in an FTS server from other VOs are also hidden from Rucio.

The BNL FTS instance is particularly interesting for this work because it is the only instance that is ATLAS specific, as other FTS instances also manage transfers for other experiments, introducing an extra source of complexity to the study.

Each instance of FTS has a centralized database with several nodes, between three to forty, that access it. Each node processes a part of the transfers submitted by Rucio. The transfers that can not be processed immediately are queued, introducing a delay that affects the total transfer time. Rucio has some limits that avoid FTS instances to get saturated by Rucio's requests.

This study is centered on the analysis of the data Rucio collects about the transfers requests, specifically about total transfer times since the creation of the request in Rucio till FTS communicates back to Rucio that the transfer is finished. The next subsections will detail the Rucio interaction with FTS and the internal structure of Rucio, emphasizing the relevant parts covered in this study.

It is important to notice that Rucio is a very dynamic tool. The Rucio development team needs to cope with new requirements from the physics community on a weekly basis and has undergone major changes since its commissioning in late 2014. At that time, Rucio was an ATLAS specific tool, created, designed, developed, and maintained by the ATLAS experiment community. Since late 2018 that is not the case anymore as Rucio was presented in the 1st Rucio Community Workshop. Since then, other experiments started to notice the power of Rucio as a distributed data management tool, and today it is used and supported by more than 20 big experiments worldwide[8] including AMS, LIGO, CMS, CTA, IceCube, DUNE, LSST, and Xenon.

Still, the Rucio instance for the ATLAS Experiment at CERN is by far the largest. This instance had indexed more than 500 Petabytes of data by middle 2020. This include centrally produced data from the experiment, such as detector data and MonteCarlo simulations, but also data from groups and users analysis.

2.3.1 Rucio Data IDentifiers

Data in Rucio is organized using Data IDentifiers. DIDs have three levels of granularity. The smallest unit of operation in Rucio is the **file**, which corresponds with the actual file persisted on the storage systems. The next level of granularity is the **dataset**. In Rucio, datasets are logical units that reference a set of files with purpose of apply bulk operations over a group of them, i.e, transfers, deletions, or replications. The biggest level of granularity is the **container**. Rucio containers allows to make large scale groupings of files and datasets, like annual detector outputs or physics simulation with similar properties. Datasets and Containers are referred to as collections. Files are allowed to be in multiple datasets. Files that belongs to a dataset do not need to be all in the same endpoint or data center, and can be distributed all across the grid. Collections can be in state **open** or **close**. While it is possible to add new DIDs to an open collection, it is not to a closed one. This is important, as shown later, all the transfer requests for files in a closed dataset are created at once.

2.3.2 Rucio Storage Elements

A Rucio Storage Element (RSE) is the minimal unit of globally addressable storage in Rucio. Most of the WLCG sites or data centers that pledge storage for ATLAS VO will expose this storage to Rucio through the setup of an RSE. This configurations are stored usually in the ATLAS Grid Information

System (AGIS) and Rucio will update its database using this information, but RSEs can be created in Rucio directly using the `rucio-admin` tool. Rucio stores data about RSEs that are needed to access them, like the network address and the port, protocols supported, and the local file system path, but also support a list of attributes with arbitrary key-value pairs. This helps to create interesting heuristics like *all the tape storage endpoints in Europe* or *all the tier 2 sites in Germany or France*. All the RSE configurations are stored in Rucio internally, so no software services are needed in the storage endpoints in order to work. File DIDs are associated to RSEs when a physical copy of the file exists in an RSE. The association is called *replica*. There could be several replicas for a file, stored in different RSEs across the grid. Files could also have no replicas, which means the file is in the catalog but will not be accessible.

2.3.3 Replication rules and subscriptions

The replica management is based on *replication rules* associated directly or indirectly to DIDs. A replication rule is an abstraction that defines the minimum number of replicas for a file to be available at any given time on a RSE or a group of RSEs in which those replicas should be. Replication rules serve two main purposes: they allow users to request data transfers between RSEs and protect DIDs from deletion. As long as there is at least one rule for a DID, the replica cannot be deleted. Replication rules are owned by *Rucio accounts*. Multiple accounts can own replication rules on the same DIDs, and thus share the same physical copy of the data.

Replication rules are created using a formal language described in [9]. Four parameters are necessary to create a rule. The DID over which the rule will have effect, an RSE expression that expand to a list of RSEs where replicas can be placed, the number of copies to replicate and the lifetime for the rule. If the DID is a dataset or a container, then all the files in that dataset or container are affected by the rule. If the RSE expression returns a set of RSEs bigger than the number of copies is up to Rucio to choose the destination of the replicas. Rucio tries to minimize the number of transfers needed to satisfy the rule, prioritizing those RSEs where data is already or partially available. The mechanism that protect the replicas from deletion is called *replica lock*. A replica lock is always associated with a rule and set once the placement decision is made. This will avoid future unnecessary re-evaluation of the rule and constant data re-shuffling between RSEs. Users does not have control over replica locks, except indirectly through the creation of rules. Several replication rules could affect a file, either because a file is in several datasets or because several users create rules for the same file, but as soon as the last replication rule is removed or it expires, also do its replica lock. Then the replica is eligible for deletion. The actual deletion is done asynchronously and not necessarily immediately after the replica is marked eligible for deletion.

Once a client requests a replication rule for a DID, Rucio evaluates the RSEs for existing data, creates transfer requests if data is not available in the specified RSEs, and creates the replica locks for the DID, among other tasks, i.e., check for user quota or space available in the RSEs. Until the deletion or expiration of the rule, the replica locks will prevent the deletion of the DID and the physical files associated with it.

2.3.4 Replica management and transfers

As mentioned before, Rucio relies on FTS3 in order to make transfers between sites. Rucio is able to use other transfer tools. The abstraction layer between Rucio and the transfer tool needs to be implemented for each transfer tool Rucio is supporting. This allow Rucio daemons to submit, query, and cancel transfers generically.

Files can be placed physically on the storage in two different ways. Either a replica is uploaded via the command line interface or the web interface, or the replica is created by a transfer to satisfy a *replication rule*. Rucio users only means to request to transfer files between storage elements is through replication rules. That is the case when a client needs to run a job in a data center, therefore the files involved in the analysis needs to be there. The Rucio internal workflow to transfer request handling is detailed in [10]. When a user creates a rule, transfer requests are created simultaneously with the destination RSEs already defined. The transfer requests are continuously read by one of Rucio daemons, which is responsible for rank the sources for each request, selects the matching protocols for source and destination, and submit the transfers, usually in bulk, to the configured transfer tool. Two separate daemons continuously check for successful or failed transfers. One of the daemons checks for FTS messages though polling continuously the transfer tool in a closed loop, but this polling could have a big impact on the transfer tool load. Most of the transfers are checked by the daemon listening at the ActiveMQ message queue[11]. The last step is to update the state of the replication rules. If all the transfers were finished successfully, the state of the rule is changed from *REPLICATING* to *OK*. Otherwise, i.e., if one of the transfers fail for some reason, it will be changed to *STUCK* and another daemon will decide if the failed transfer should be resubmitted or if a new transfer request with a different destination RSE should created.

For closed datasets, all the transfers are created at the same time when the rule is created. For open datasets, some transfers can be created as new files are added to the dataset. Although Rucio does not submit the transfers of a rule to the transfer tool all at the same time, even if they are all created at the same time. This is done in order to avoid overload the transfer tool.

Rucio persists data for every transfer request in the REQUESTS table in the Rucio database. When a transfer is created in order to satisfy a replication rule, the `created_at` timestamp is logged in the Rucio database. Likewise, the `submitted_at` timestamp is logged when the transfer is submitted to the transfer tool. Rucio will know that the transfer ends through one of its daemons. At this time, Rucio will also know about the `started_at` and `ended_at` timestamps of the transfer. The first is the time when the transfer begins and the second, when the transfer ends. Data about the file name, its size in bytes, source and destination RSEs, associated rule id, account, activity , priority, and instance of the transfer tool the transfer was submitted to, is also persisted in the Rucio database.

Using these data it is possible to determine the Rule TTC of a replication rule once the rule is in state `OK`. This time will be the difference between the maximum `ended_at` and the minimum `created_at` of all the transfers of the rule.

Chapter 3

Data selection and model metrics

In God we trust, all others must bring data.

W. Edwards Deming

3.1 Rucio data extraction and selection

Through the study of the data persisted by Rucio about the transfers requests, this work tries to probe if it is possible to make a reasonable forecast of the Time To Complete of rules and transfers. The data selection was planned taking into account the following factors that could impact the study.

3.1.1 Transfers and Deletions

Rucio handles transfers between sites but also handles the deletion requests in order to comply with the data retention policy. Both, transfers and deletions requests are stored in the REQUESTS and REQUESTS_HISTORY tables: the REQUESTS table store the current requests, the ones that will be updated soon, i.e.: with new timestamps or states, while the REQUESTS_HISTORY table is an archive of requests that will not be updated anymore, that is, the requests with final state. Deletion requests do not affect RSE transfer performance, and then, can be ignored. Only transfers requests were took into account for this work.

3.1.2 FTS Server

There are several caveats about FTS that need to be addressed in order to create a dataset from the Rucio database that allows the study of the transfer times. The Rucio database does not contain information about the transfer tool other than the instance that is used for each transfer request. Information about FTS queue states, scheduling and retries, number of nodes, and configuration are hidden from Rucio. As mentioned in the previous section, there are several instances of the transfer tool among the grid. These instances work at WLCG level and serve transfers from several VOs and not only ATLAS specific transfers. Yet transfers in an FTS server from other VOs are also hidden from Rucio. Given a slice of time, there is no way to know the total amount of transfers an FTS server is processing other than ATLAS transfers, looking at the ATLAS Rucio database instance. However, one hypothesis is that the load in FTS could have an impact in the difference of the submission and starting time of a transfer. The more load at FTS, the more time will elapse between the submission and starting of a transfer.

The BNL FTS instance is ATLAS VO specific. Selecting transfers that only go through this FTS server will prevent effects on data related to transfers Rucio cannot see, but that we assume have an impact on the TTC of transfers we have in our datasets. The downside is that some transfers processed by Rucio going to other FTS servers in the same time span will be filtered out and it is impact on the transfer time of the studied transfers will not be taken into account. We assume this impact in negligible, but it is unproven yet.

For some studies, data was filtered in order to get only those request transfers with `external_host` equal to `https://fts.usatlas.bnl.gov:8446`, see Table 3.1. These transfers are served by the BNL FTS, an instance that is ATLAS specific. This allow to focus on groups of transfers that are not competing for FTS resources with other transfers, invisible to Rucio, from other VOs.

3.1.3 TAPE activities

The WLCG infrastructure uses tape endpoints present at some sites, usually Tier 0 or Tier 1, in order to archive datasets. As the access time of tape endpoints is large when compared to disk, there is a set of policies that strictly regulate which files are going to tape and who are the users who can read or write to tape endpoints. Rucio distinguishes between tape or disk RSEs internally, but for users, the transfer requests are transparent.

However, at infrastructure level, transfers that involve at least one tape

RSE behave completely different from the ones that involves disk RSEs only. When a transfer going from TAPE to DISK RSEs is submitted by Rucio to the FTS, once the submission is attended by FTS the file in the source which is in tape support is copied first to a local disk buffer in the same RSE, in a process called *staging*. Once the staging is complete, the transfer of the file to the destination is started. FTS will timestamp this event as `started_at` indicating the time the transfer of the file starts to consume bandwidth from the network. Once the file transfer is finished or fail, the event is also timestamped as `ended_at`. This field is called `transferred_at` in the Rucio database, but have been renamed for convenience. See Table 3.1 on page 22. Only after the `ended_at` attribute is set, the FTS server generates the log messages to communicate all these data to Rucio. From the Rucio database point of view, if the request transfer is from tape to disk there is no way to identify the time a transfer is queued in FTS from the staging time as both are contained in the difference between `started_at` and `submitted_at` timestamps.

For this reason, an important aspect is the number of tape to disk transfers requests in the data sample. A time window in which the tape transfer activity was low was identified and selected, and only data from this time period was included in the study. Such period contains the transfers created between June 6th and July 31st of 2019, with specially low tape activity during July.

Since the beginning of 2020, the Rucio development team had introduced a change in the Rucio database that allows to distinguish the staging time from the FTS queue time for tape transfers. Future studies can benefit from it and make more detailed analyses.

3.1.4 Failed transfers

Transfer that failed at some point during the network transfer are also a source of noise. TTC predictions should be intended only for transfers that do not fail. However an attempt to filter out failed transfers from the dataset proved to be the wrong decision. That is because the failed transfers usually fail during the network stage, and thus spend time in FTS queue. FTS still need to process the transfer that only in a later stage will be marked as FAILED in the Rucio database. Thus, failed transfers had been included in the studied dataset, but filtered out later after its effects in FTS queue time are removed.

3.1.5 Data extraction and treatment

Data extraction from the Rucio database was done through a Hadoop script that dumps the raw content of the REQUESTS_HISTORY table in Rucio. This table stores all the information about the transfer requests that finished. While the information in the REQUESTS table can be updated by Rucio daemons, once the transfer is done or has failed then its information is copied to the REQUESTS_HISTORY table for database optimization reasons. As the REQUESTS table is updated constantly, it needs to be small as each modification locks the table. The REQUESTS_HISTORY table is bigger but partitioned in several indexes, i.e., by the request created_at attribute. The database retention policy for REQUESTS_HISTORY table is six months.

The transfers archived in the Rucio database that were created between June 6th and July 31st of 2019 affect around 128.6 million rows.

The data was filtered using the criteria described in the previous section, meaning all the requested transfers with DONE and FAILED state that were created between June 6th and July 31st of 2019 were included in this study.

Only source and destination RSE identification (id) information are stored in the REQUESTS_HISTORY table, with respect to where the data is coming from or where is going to. The RSES table has the association between the RSE id and the RSE name. The mapping between the RSE id and the RSE name were done using the same Hadoop script mentioned before.

Output was a CSV file of approximately 128.6 million rows with fields described in Table 3.1, but also `src_rse` and `dst_rse` with the mapped names of the source and destination RSEs respectively.

If only BNL FTS transfers are accounted, the number of rows to treat were reduced to 31.8 million. From this, around 18.84% or approximately 6 million rows represent failed transfers. There are 25.5 million rows that represents files that were successfully transferred. Also there are 238518 rows with state S or SUBMITTED, 82820 rows with state L or LOST, 2722 rows with state G or SUBMITTING and 238 rows with state A or SUBMITTION_FAILED. These transfers add up to 324295 rows or 1% of the total, and are in an unfinished state for Rucio. They do not have any value for the columns started_at and transferred_at. As is not possible to reconstruct the full lifetime of this requests, they have been ignored in the studies involving BNL FTS transfers only.

All the pre-processing beyond this point were done using Python scripts, with extensive use of *numpy*[12] and *pandas*[13] libraries.

The CSV file was read into a pandas DataFrame. The fields `created_at`, `submitted_at`, `started_at`, and `ended_at`, were mapped from strings to pandas Datetime objects, from `created_at`, `submitted_at`, `started_at`, and `transferred_at` respectively. Rows with duplicated transfer id, `id` field were dropped, less than 0.01%. Rows that do not satisfy the condition `created_at` ≤ `submitted_at` ≤ `started_at` ≤ `ended_at` were dropped, also less than 1%.

`RTIME`, `QTIME`, and `NTIME` columns were calculated as the number of seconds between `created_at` and `submitted_at`, `submitted_at` and `started_at` and `started_at` and `ended_at` respectively. The `size` column was renamed to `SIZE` due to `pandas.DataFrame.size` is an attribute of the pandas DataFrame object and SIZE is not. The column `RATE` was calculated dividing the `SIZE` among `NTIME`. This gives us an approximated rate of the transfer during its network time. It is an approximation because it is not possible using these data to know the instantaneous rate of a transfer. The rate of the transfers is discussed with detail in Chapter 7, starting on page 62.

WLCG and Rucio make a difference between sites and RSEs. While a site can host several RSEs, the network links of the WLCG are between sites. This network links are shared among several experiments and VOs. Rucio however do not know about sites. Site information needs to be reconstructed from RSE name and data in ATLAS Grid Information System (AGIS). The information for which RSE pertains to which site is mapped in AGIS and can be downloaded in JSON format. JSON files are trivially mapped to Python dictionary objects and equally trivially mapped to pandas DataFrame columns. A `link` column was created with the concatenation of the source site, the string `'__'`, and the destination site, i.e., a transfer from source RSE `CERN-PROD_TZERO` to destination RSE `BNL-OSG2_DATADISK` will have a value `CERN-PROD__BNL-ATLAS` for the link column, indicating that the source site is `CERN-PROD` and the destination site is `BNL-ATLAS`.

The columns `account`, `externa_host`, `src_rse`, `dst_rse` and `link` has been anonymized using the last 15 characters of the result of apply the SHA512 algorithm from Python's *hashlib* to the inputs. The resulting dataset has been made publicly available[1].

It is possible to calculate how many bytes were transferred between the sites. From June 6th to July 31st of 2019, Rucio accounts for 78.36 PiB of data, only including the successful transfers. Of that, 28.72 PiB were transferred through BNL FTS instance.

3.2 Metric election

In order to evaluate how good different models agree with the data it is required to define a metric. Standard metrics for regressions tested include root mean squared error (RMSE) and mean squared error (MSE)[14], mean absolute error (MAE) and median absolute error (MedAE), mean squared

Attribute Name	**Description**	**Type**
`id`	A unique identification number of the transfer	string
`rule_id`	The identification number of the rule that generated the transfer	string
`account`	Name of the account in which name the transfer is done	string
`activity`	Name of the activity of the transfer.	string
`size`	Size of the transferred file in bytes	number
`dst_rse_id`	The RSE id of the destination of the transfer	string
`src_rse_id`	The RSE id of the source of the transfer	string
`external_host`	The network name (protocol://hostname:port) of the FTS server attending the transfer of the file	string
`state`	The final state of the request, either `D` for a DONE transfer or `F` for a FAILED transfer	string
`previous_attempt_id`	The request id for the previous request to make the transfer. Only populated for transfers that have failed before	string
`created_at`	Timestamp of the creation time of the transfer in Rucio	datetime
`submitted_at`	Timestamp indicating the moment the transfer is submitted from Rucio to FTS	datetime
`started_at`	Timestamp of the beginning of the transfer between source and destination	datetime
`transferred_at`	Timestamp indicating the moment the transfer between source and destination ends	datetime

Table 3.1: Attributes and descriptions of the REQUEST_HISTORY table. This raw data was preprocessed and transformed in order to include more meaningful information for this study. The transformed dataset and the description of its fields is available at [1].

logarithmic error (MSLE)[14] and root mean squared logarithmic error (RMSLE), explained variance score, R^2 score, mean Tweedie deviance, mean absolute percentage error (MAPE)[14], relative error (RE) and related metrics, and finally the Fraction of Good Predictions (FoGP)[15].

In order to analyze the metrics we use the following notation. We define y as the vector of target or observed values of length n, and about $\hat{y}$, the vector of predictions of the same length, in which $\hat{y}_i$ is a prediction for the element y_i of y, with the sub-index i going from 0 to $n-1$. In all the equations $\overline{y}$ will stand for the arithmetic mean of y.

When the metrics are applied to evaluate two or more models, the target vector y is assumed to be the same. We denote the predictions of model α and model β, $\hat{y}_\alpha$ and $\hat{y}_\beta$ respectively and $\hat{y}_{\alpha i}$ to the prediction made for the model α of the target element y_i.

3.2.1 MSE and RMSE

The mean squared error measures the mean of the squared difference between the vectors y and $\hat{y}$, according the Equation 3.1.

$$MSE(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \tag{3.1}$$

As the differences are squared this metric penalizes more the big differences and as it is a mean value it is sensitive to outliers. The RMSE version is the squared root of the MSE and its units are comparable with the units of y and $\hat{y}$, so if y and $\hat{y}$ are in seconds, the RMSE can be interpreted in seconds too. When several models are to be compared or when the values of y have great variance, MSE and RMSE are not particularly useful. Two models α and β will be considered with comparable performance if $RMSE(y,\hat{y}_\alpha)$ and $RMSE(y,\hat{y}_\beta)$ are in the same order of magnitude, but always the model with less RMSE will be preferred.

3.2.2 MEA and MedAE

The mean absolute error and the median absolute error are the mean and median of the absolute value of the difference between y and $\hat{y}$ respectively. The MAE is calculated using Equation 3.2, whereas MedAE using Equation 3.3.

$$MAE(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i| \tag{3.2}$$

$$MedAE(y, \hat{y}) = median(|y_i - \hat{y_i}|), i = 0, 1, 2, ..., n - 1 \tag{3.3}$$

MAE and MedAE are simpler to interpret than MSE and RMSE, but MedAE is preferred for its robustness to outliers in y and $\hat{y}$. However, the four metrics are sensitive to the scale of y. This means that when the same model is evaluated, using two different set of observations y and y', the metric will be different for the same model and will depend on the distribution of y and y'. If y presents outliers and y' does not, the metric for the same model will be worse regardless the performance of the model. This is not a problem when two models are compared against the same y, but it does not give an idea of the goodness of the models in general.

3.2.3 MSLE and RMSLE

The mean squared logarithmic error is a metric robust against outliers and not sensitive to the scale of y. It is defined by Equation 3.4 as the mean of the squared differences between natural logarithms of $1 + y$ and the natural logarithm of $1 + \hat{y}$. The root mean squared logaritmic error is the squared root of MSLE.

$$MSLE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (ln(1 + (y_i) - ln(1 + (\hat{y}_i))^2 \tag{3.4}$$

Hidden in the definition lies the problem that this metric tends to penalize more the negative errors than the positive ones, and thus will favor a model that overestimates the predictions over one that underestimate them. This could be an advantage in some scenarios. But the metric is difficult to interpret and the results does not give a good idea of the goodness of a model.

3.2.4 Explained Variance and R^2 Score

The explained variance score and the R^2 score are two metrics related to each other. The differences are subtle but important. The explained variance score is calculated using the Equation 3.5, which expands to Equation 3.5 or in a more elegant way as Equation 3.7, in which the overbar stands for the arithmetic mean and the $1/n$ has been simplified. It can be interpreted as the proportion of the variance of y that is explained by the model though the predictions $\hat{y}$.

$$EV\_score(y, \hat{y}) = 1 - \frac{Var(y - \hat{y})}{Var(y)} \tag{3.5}$$

$$EV\_score(y, \hat{y}) = 1 - \frac{\frac{1}{n}\sum_{i=0}^{n-1}((y_i - \hat{y}_i) - \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i))^2}{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \frac{1}{n}\sum_{i=0}^{n-1} y_i)^2} \tag{3.6}$$

$$EV\_score(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}((y_i - \hat{y}_i) - (\overline{y_i - \hat{y}_i}))^2}{\sum_{i=0}^{n-1}(y_i - \overline{y_i})^2} \tag{3.7}$$

The R^2 score, also known as coefficient of determination, is defined as Equation 3.8 or more elegantly as Equation 3.9.

$$R^2\_score(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \frac{1}{n}\sum_{i=0}^{n-1} y_i)^2} \tag{3.8}$$

$$R^2\_score(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \overline{y})^2} \tag{3.9}$$

In Equations 3.7 and 3.9 it is possible to spot the difference at a glance. Both results are equal if $\overline{y_i - \hat{y}_i}$ is zero, meaning the R^2 score does not account for biased models as the explained variance do. This also make the R^2 slightly more sensitive to the scale of y.

Interpretation of both metrics is not clear at a glance but are implied directly from the equations. In both cases, if the prediction of the model is perfect, then $y - \hat{y} = 0$ and then both scores are equal to 1. This is the best score a model can achieve. By definition, a model that makes a prediction using the mean $\overline{y}$ has an score of 0, so any model with a score bigger than zero will be better than the naive model. But then the predictions can be infinitely worse returning negative values in both scores.

3.2.5 Mean Tweedie Deviance

The mean Tweedie deviance is a goodness-of-the-fit metric based on the Tweedie distribution. The Scykit-Learn `mean_tweedie_deviance()` computes the mean Tweedie deviance error between y and $\hat{y}$ with a power parameter p, using Equation 3.10. With $p = 0$, $p = 1$, and $p = 2$ the metric behaves as the mean squared error, mean poisson deviance, and mean gamma deviance respectively.

	$y_\alpha = 1$	$y_\beta = 100$
$\hat{y}$	1.5	150
$MTD(y, \hat{y}, 0)$	2.5	2500
$MTD(y, \hat{y}, 1)$	0.189	1.890
$MTD(y, \hat{y}, 2)$	0.144	0.144

Table 3.2: Summary of the results using the Mean Tweedy Deviance function.

$$MTD(y, \hat{y}, p) = \begin{cases} \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 & \text{for p = 0;} \\ \frac{1}{n}\sum_{i=0}^{n-1} 2(y_i ln(y_i/\hat{y}_i) + \hat{y}_i - y_i) & \text{for p = 1;} \\ \frac{1}{n}\sum_{i=0}^{n-1} 2(ln(\hat{y}_i/y_i) + y_i/\hat{y}_i - 1) & \text{for p = 2;} \\ \frac{1}{n}\sum_{i=0}^{n-1} 2(\frac{max(y_i,0)^{2-p}}{(1-p)(2-p)} - \frac{y_i\hat{y}_i^{1-p}}{1-p} + \frac{\hat{y}_i^{2-p}}{2-p}) & \text{else.} \end{cases} \tag{3.10}$$

Notice that for a trivial case of two models, α and β with equal relative error of 0.5, if $\alpha([1]) = [1.5]$ and $\beta([100]) = [150]$, then for $p = 0$, $MTD(y_\alpha, \hat{y}_\alpha, 0) = .25$ and $MTD(y_\beta, \hat{y}_\beta, 0) = 2500$; for $p = 1$, $MTD(y_\alpha, \hat{y}_\alpha, 1) = 0.189$ and $MTD(y_\beta, \hat{y}_\beta, 1) = 1.890$; and for $p = 2$, $MTD(y_\alpha, \hat{y}_\alpha, 2) = 0.144$ and $MTD(y_\beta, \hat{y}_\beta, 2) = 0.144$. All these values are summarized in Table 3.2

From these examples we conclude that the MTD with power of 2 is only sensitive to relative errors, meanwhile MTD with power 0 and 1 are sensitive to scale.

3.2.6 MAPE and RE

Mean Absolute Percentage Error has the advantage of being easy to interpret. MAPE is based on the Relative Error, that is the absolute value of the difference between target and the prediction relative to the target, as shown in Equation 3.11, and thus, is the error in the prediction relative to the observed value.

$$RE(y_i, \hat{y}_i) = \frac{|y_i - \hat{y}_i|}{y_i} \tag{3.11}$$

If this values are multiplied by 100, this gives the range of the error percentage or how far away is $\hat{y}_i$ from y_i in percentage. The MAPE is the mean of the RE expressed as a percentage, as in the Equation 3.12. But, as it is a mean value, it is sensitive to outliers in the relative errors. It is possible to overcome this by taking the median instead of the mean in

3.12. Mean Absolute Percentage Error and Median Absolute Percentage Error both diverge when y values are very close to zero.

$$MAPE(y, \hat{y}) = 100\frac{1}{n}\sum_{i=0}^{n-1}\frac{|y_i - \hat{y}_i|}{y_i} \tag{3.12}$$

Other MAPE derived metrics try to avoid its caveats. That is, it is known that MAPE penalizes more the positive forecasts values than the negative ones[16]. sMAPE or Symetric Mean Absolute Percentage Error and sMedAPE try to address this issue but still can suffer of the divergence problem due to the sum $y_{+}\hat{y}_i$ being small.

In [14], the MASE or Mean Absolute Scaled Error is introduced to circumvent the mentioned problems. Yet all these derived metrics opaque the simple interpretation of MAPE. Moreover, the models to evaluate in this work make predictions over positive integer targets, as both the Rule TTC and Transfer TTC are measured in seconds.

3.2.7 FoGP

The metric selected to compare models in this work is described in [15] (p.16) as percentage of predictions with less than X percent RE. We call this metric Fraction of Good Predictions ($FoGP$), expressed as a number between 0 and 1, in which X is the threshold of relative error below of which a prediction is considered good.

Formally, with the trivial function g defined as in Equation 3.13, $FoGP$ is defined as in Equation 3.14.

$$g(y_i, \hat{y}_i, \tau) = \begin{cases} 1 & \text{if } RE(y_i, \hat{y}_i) \leq \tau; \\ 0 & \text{else.} \end{cases} \tag{3.13}$$

$$FoGP(y, \hat{y}, \tau) = \frac{1}{n}\sum_{i=0}^{n-1} g(y_i, \hat{y}_i, \tau) \tag{3.14}$$

As an example, assume that a certain model made a prediction for y. We calculate the $FoGP$ with threshold 0.05 and we obtain the 0.5 as results. Formally, that can be expressed as $FoGP(y, \hat{y}, 0.05) = 0.5$. This means that 50% of the predictions in $\hat{y}$ are less than 5% away from it is real values.

3.2.8 Metrics comparison experiment

To demonstrate the suitability of the FoPG, we run a simple experiment comparing several of the metrics. We define four artificial target arrays y.

The target $y1$ of 100K integers is in the range (1, 1000), with uniform distribution. This range is typical of the Rule and Transfers TTC but both present outliers. Target $y2 = y1 * 10$, that is $y2$ is exactly the same as $y1$, but all its values were multiplied by a constant to check the sensitivity of the metric to the scale in y. Target $y3 = y1$, but a random 0.01% of $y3$ or 100 values were multiplied by 1000, to simulate 0.01% of outliers. And target $y4 = y2$ but a random 0.01% of $y4$ or 100 values were multiplied by 1000. Two models, α and β where defined in such a way that the predictions of α are always overestimated by 10%, and the ones of β underestimated by 10%. That is, $\alpha(y_i) = y_i + y_i * 0.1$ and $\beta(y_i) = y_i - y_i * 0.1$. These two models are by definition identically good, and its predictions were compared with all the metrics mentioned before. Results are summarized in Table 3.3.

From the table it is possible to see that for MAE, MSE, RMSE, both models are as good as each other, i.e., $MAE(y1, \alpha(y1)) = MAE(y1, \beta(y1))$. But these metrics do not give any information about how good the models are in general, as the metric values depend on the scale of the target, i.e: $MAE(y1, \alpha(y1)) \neq MAE(y2, \alpha(y2))$. That is the case for MedAE, but this metric is also robust against outliers in the target, i.e., $MedAE(y1, \alpha(y1)) = MedAE(y3, \alpha(y3))$. MSLE seems to be sensitive to the scale of y but robust to outliers. But it is clearly prone to benefit models that overestimate the predictions, i.e., $MSLE(y1, \alpha(y1)) < MSLE(y1, \beta(y1))$. Also it is very difficult to get an idea of the absolute goodness of a model. EVS and R^2 are both robust to scale in y, but R^2 seems to give higher values in the samples with outliers. More over, contrary to the MSLE, it is not difficult to get an idea of how good the model is in absolute terms. Both metrics top score is 1, and the results for this models are very close. Yet both predictions are off by 10%, which can be too high. $MTD(power = 1)$ is both sensitive to scale and outliers. These problems are solved increasing the power. But both metrics consistently report lower, better values for the model α that overestimate the target. It is still difficult to know the goodness of the model in absolute terms. The MAPE metric for both models is 0.1 as, by definition, both models predict exactly the target with a relative error of 10%, regardless the scale and fraction of outliers in y. Yet MAPE will not be robust to outliers in $\hat{y}$, that is, a couple of predictions with high RE will inflate the MAPE metric artificially. For the FoGP metric, both models are the same, MAPE sensitivity to outliers in $\hat{y}$ is not present as only the "good" predictions are took into account, and gives a good idea of the absolute goodness of the model. For that reason this is the main metric used in this study.

	MAE	*MedAE*	*MSE*	*RMSE*	*MSLE*	*EVS*	R^2	*MTD(p=1)*	*MTD(p=2)*	*MAPE*	*FoGP* (τ=0.1)
$\alpha(y1)$	49.963	50.0	3325.276	57.665	0.00898	0.990	0.960	4.68640	0.00880	0.10	1.0
$\alpha(y2)$	499.635	500.0	332527.572	576.652	0.00907	0.990	0.960	46.86396	0.00880	0.10	1.0
$\alpha(y3)$	103.363	50.0	3628962.150	1904.984	0.00898	0.990	0.990	9.69508	0.00880	0.10	1.0
$\alpha(y4)$	1007.806	500.0	350386237.518	18718.607	0.00907	0.990	0.990	94.52860	0.00880	0.10	1.0
$\beta(y1)$	49.963	50.0	3325.276	57.665	0.01096	0.990	0.960	5.35660	0.01150	0.10	1.0
$\beta(y2)$	499.635	500.0	332527.572	576.652	0.01108	0.990	0.960	53.56602	0.01150	0.10	1.0
$\beta(y3)$	103.363	50.0	3628962.150	1904.984	0.01096	0.990	0.990	11.08158	0.01150	0.10	1.0
$\beta(y4)$	1007.806	500.0	350386237.518	18718.607	0.01108	0.990	0.990	108.04722	0.01150	0.10	1.0

Table 3.3: Comparison of the different metrics used in regression. Model α overestimate the target by 10% while β underestimate it by 10%, that is $\alpha(y) = y + y * 0.1$ and $\beta(y) = y - y * 0.1$. Target $y1$ is an array of 100K integers in the range (1, 1000); $y2 = y1 * 10$ so is in the range(10,10000); $y3 = y1$ but a random 0.01% or 100 values where changed to $y3_i = y1_i * 1000$ to simulate 0.01% outliers; $y4 = y2$ but a random 0.01% or 100 values where changed to $y4_i = y2_i * 1000$ to simulate 0.01% outliers;

Chapter 4

Model of intra-rule Rule TTC extrapolation

There are two kinds of people in the world:
1.Those who can extrapolate from incomplete data.

Anonymous

4.1 Transfers per rule distribution

Using the data in the transfers dataset[1], it is possible to identify and group the requests triggered by a particular replication rule. In order to calculate statistics on the rules a new CSV file with summarized information about the rules was created. We call this dataset the rules dataset. This file contains the rule_id, the number of transfers calculated as the number of transfers done with the same rule_id field, the sum of bytes of the individual transfers, the min/max created, the min/max submitted, the min/max started, and the min/max ended timestamps. Figure 4.1 shows the frequency of the number of rows with the same *rule_id* field. Notice the peaks in 20, 25, 30, 40, 50, 100, and 200. Other notable peaks are 1, 2, 4, and 6.

The number of transfers per rule varies from rule to rule, but there are notable peaks in some numbers, most notably, rules with exactly 20 transfers. These rules are generated by an automated replication process by the experiment.

Analysis is done on data from FTS BNL instance for the reasons mentioned in 3.1.2. The set of transfers going through the BNL FTS instance

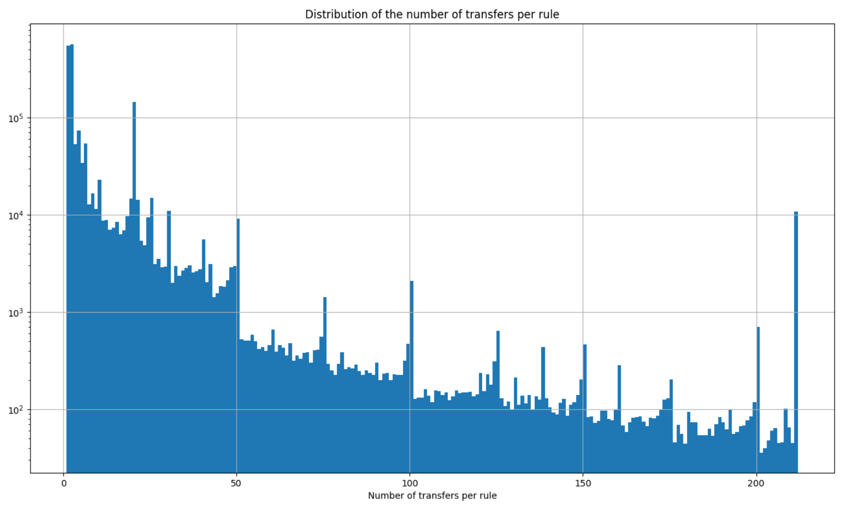


Figure 4.1: Distribution of the number of transfers per rule. The maximum number of transfers per rule in the dataset is 205951 but rules with more than 250 transfers are summarized in the last bin.

contains 1803027 different rules. The mean rule TTC is 3.1 hours but the mean varies within two orders of magnitude depending on the number of transfers per rule, as shown in Figure 4.2.

It can happen that the transfers of the rule are not created at the same time. This is normal for open Rucio datasets, where new files are added to the dataset as they are created. However, for 77.72% of the rules, the transfers are created all at once. It is possible to identify those from the rules summary file, selecting the rules with min_created equal to max_created timestamps.

The rules dataset does not provide information about the progress of the rule, but only historical data about rules that have already finished. This is important for subsequent models but not for model α described below. In that case, the transfers dataset was used to build a model that can predict the rule completion time based on the progress of transfers that already have finished.

4.2 The α and $\alpha\prime$ models

In this section, *alpha* model family is defined to predict the Rule TTC.The prediction for the Rule TTC is based on the created_at timestamp of the first created transfer, the ended_at timestamp of the first ended transfer and

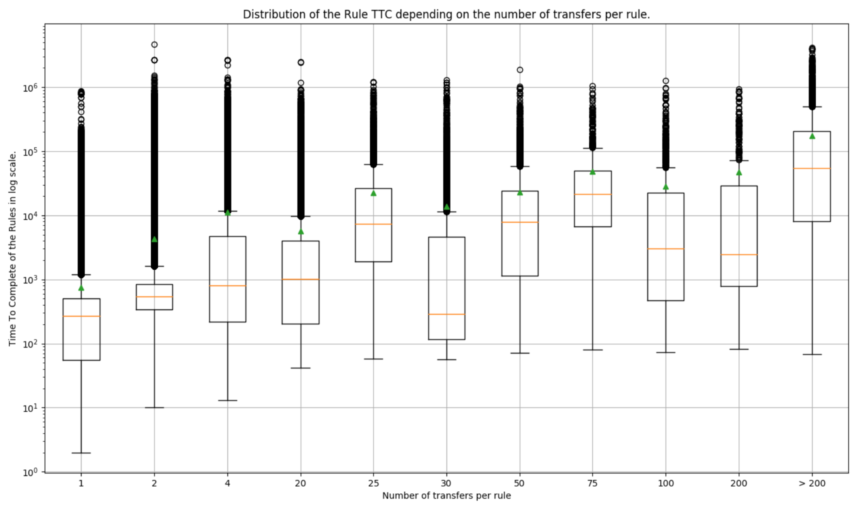


Figure 4.2: Box plots showing the differences in Rule TTC among the number of transfers per rule. The correlation between the number of files in the rule and the Rule TTC is weak.

the total number of transfers to be created and transferred to fulfill the rule. This number is known beforehand by Rucio if the dataset is closed, which happens approximately 70% of the time. A closed dataset is one at which no more files can be added, and thus, when a closed dataset is going to be moved, the total number of files to be transferred is know. If the number of transfers in the rule is known, it is possible to compute the true percentage of progress of the rule, using Equation 4.1, where $total\_xfers$ is the total number of transfers to be transferred to fulfill the rule and $ended\_xfers$, the number of transfers in the rule that already have been transferred. It is also possible to compute the elapsed time from the first creation until the ending time subtracting the minimum created_at to the ended_at timestamps of the transfers of the rule. Using regression analysis it is possible to approximate the Rule TTC using these data.

$$\%Completed = \frac{ended\_xfers * 100}{total\_xfers} \tag{4.1}$$

As an example, assume a rule needs 20 transfers done to be completed. Assume all the transfers are created at the same time, on 2019-07-27 00:00:00, and then ten seconds later, at 2019-07-27 00:00:10 the first transfer finishes. When plotted in the xy-plane, the creation of the first transfer is in the

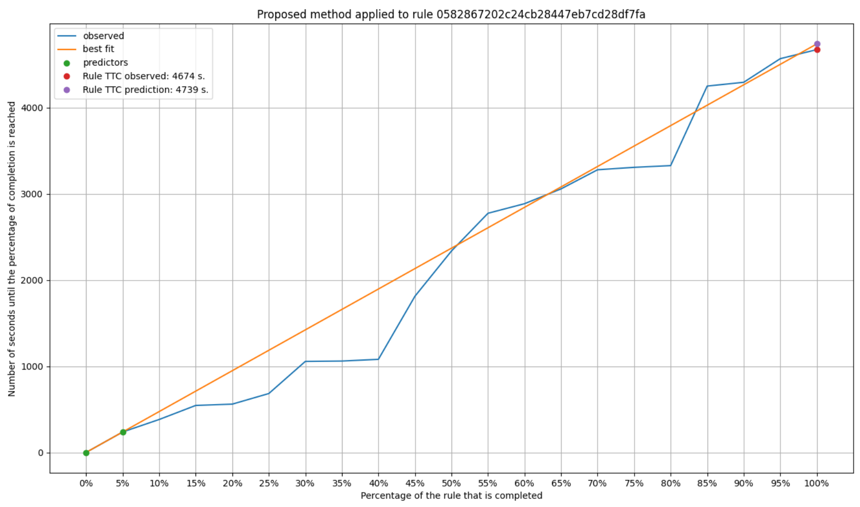


Figure 4.3: Regression analysis applied to a rule in order to predict its TTC. Prediction is made after the first transfer ends. In this rule, with 20 transfers, this means the prediction is made after 5% of the rule has been completed.

origin, at $x = 0\%$ completed and $y = 0$ seconds after the first creation. The first completed transfer is at $x = 5\%$ and $y = 10$ seconds. A one degree polynomial $y = 2x + 0$ is determined by these two points. Evaluating the polynomial in x = 100 will give us an approximation of the TTC of this rule, in this example, of 200 seconds. Figure 4.3 illustrates this approach using a rule with 20 transfers. The prediction for the Rule TTC is done after the first transfer is finished, that is, after the 5% rule completion. This case is a particularly good prediction although not the most common case. Most of the rules have a behaviour similar to that of Figure 4.4 on page 34 where the points t_0 or the rule creation time, the prediction time or the time after the first transfer in the rule ends, and the 100% rule completion time do not fall over a straight line. It is possible to see more examples of this behaviour for a selection of rules in Figure 4.5 on page 35.

Formally, the method consists of a regression analysis using ordinary least squares to fit a 1-degree polynomial where the independent variable is the completions percentage of the rule, and the dependent variable is the time at which the percentage of completion is reached.

More than two points can be used in the regression analysis. But the more points are used, the more transfers need to finish before the prediction can be made, therefore the more the rule needs to progress the less useful the

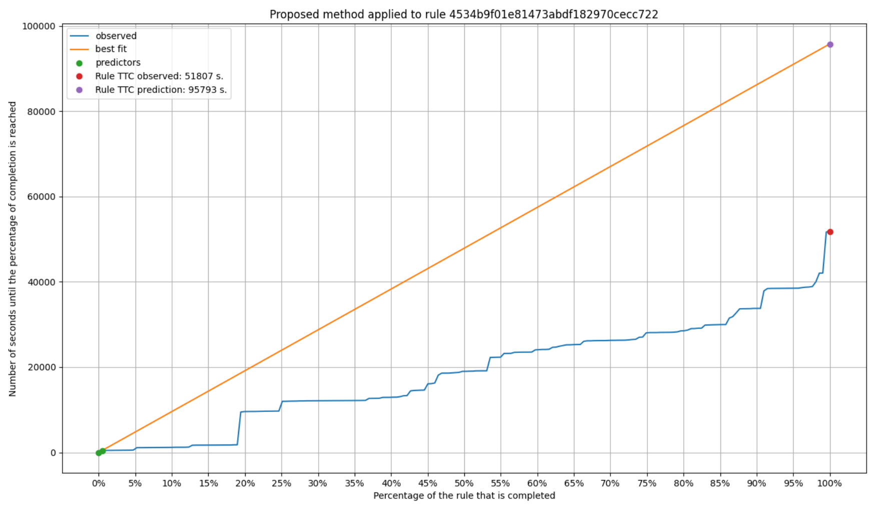


Figure 4.4: Regression analysis applied to a rule in order to predict its TTC. Prediction is made after the first transfer ends.

prediction will be. This can be an issue to take into account even for a low number of points, as for rules with two transfers the rule needs to progress till 50% before a prediction can be made. This method also cannot be applied to a rule with just one transfer. Also, given the non linearity of the behavior shown by the progress of the rules, adding more points to the fit does not guarantee improvements on the results.

The model is not computationally demanding and can be applied in real time in the Rucio dataflows. The method to solve the linear regression problem is based on the Numpy *polyfit()* function, which according to experiments done with `%timeit` ipython interactive interface puts an upper bound of 59 μs to fit a 1 degree polynomial with two random points, and 64 μs with one hundred points.

We define the family of models α_k as in Equation 4.2, where $ax + b$ is the polynomial that results from the fit of the predictor vector X_k, using the Ordinary Least Squares method. We call the predictor X_k to the vector of points $((\chi_{1j}, \chi_{2j}))$, where χ_{1j} is the component that represents the percentage of the rule that is completed, and χ_{2j} is the time elapsed in seconds until that χ_{1j} is reached. The sub-index k is the number of points used to make the fit, so the range is $k = 2, 3, 4, ..., n$ where n is the number of transfers in the rule. Thus, j sub-index range from $1, ..., k$.

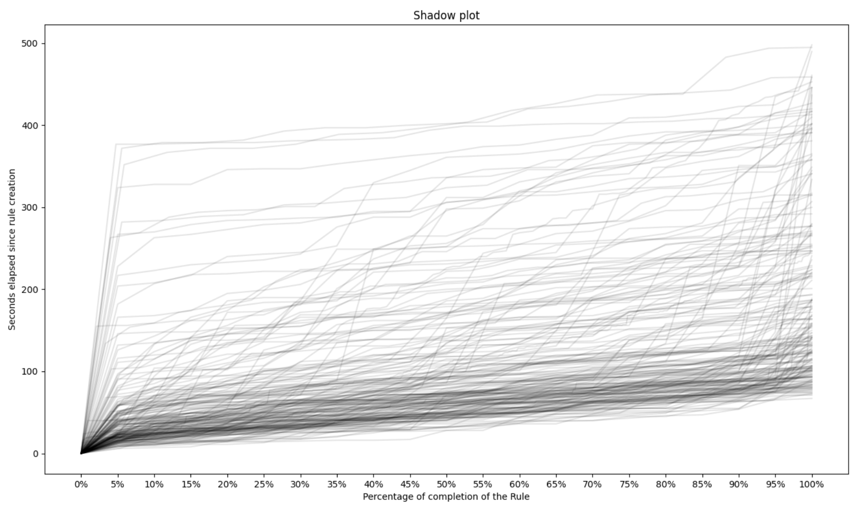


Figure 4.5: Time elapsed since rule creation as a function of percentage of completeness for rules with Rule TTC $\leq$ 500 seconds.

$$\alpha_k(X_k) = ax + b \tag{4.2}$$

One possible interpretation for k sub-index is that $k-1$ is the number of transfers of the rule that needs to be done or finished before a prediction can be made using the model. Naturally, α_n does not work, as the purpose of the these models is to make a prediction in advance. Hence, only the first members of the α_k family were evaluated.

4.3 Evaluation of results

The models $\alpha_2, \alpha_3, ..., \alpha_{10}$ were evaluated using the Fraction of Good Predictions (FoGP) metric and the following procedure.

A sample of 500 rules were picked at random, where all the transfers of each rule where created at the same time. Predictor vector X was created from the rule and transfer information. One prediction $\hat{y}_{\alpha_k i}$ was done using models α_k and X_k, with $k = 2, ..., 10$ and $i = 1, ..., 500$. This vector $\hat{y}_{\alpha_k}$ was compared with the vector y with the observed Rules TTC using the FoGP metric. Only rules with 15 transfers or more were chosen. From all the rules in the dataset, this rules represent the 20% and cover 85% of the transfers.

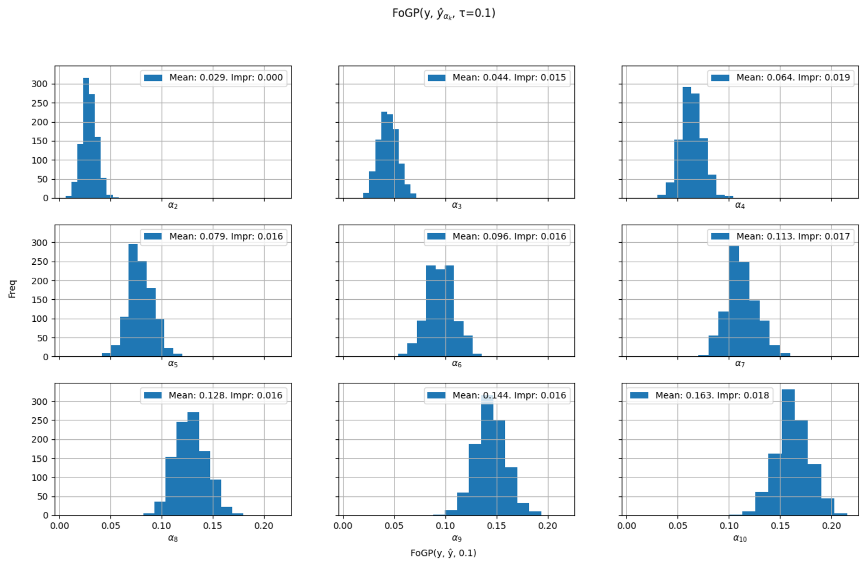


Figure 4.6: Distribution of the FoGP$(y, \hat{y}_{\alpha_k}, \tau = 0.1)$ for the first 9 members of the α_k family. The bigger the k, the more points are included in the regression, the more accurate the results, and the latest the prediction. For α_2, at least one transfer needs to finish before a prediction can be made, while for α_{10}, at least 9 transfers need to finish in order to make a prediction.

For each model, FoGP$(y, \hat{y}_{\alpha_k}, \tau)$, were computed, with thresholds $\tau = 0.1, 0.25, 0.5, 0.75, 0.9$, and 1.0.

The experiment has been repeated 1000 times, and the mean of the 1000 FoGP$(y, \hat{y}_{\alpha_k}, \tau)$ were used as a metric comparison between the models. Figure 4.6 on page 36 shows the distribution of the FoGP$(y, \hat{y}_{\alpha_k}, \tau)$ for $\tau = 0.1$ for the different models of the α_k family. Notice that the bigger the k, that is, the more points are used to make the fit and therefore the better performance of the model. Similar outcome was produced whatever the value of τ.

The number of transfers per rule follows the distribution in Figure 4.1 on page 31. When rules with less than 15 transfers are disregarded, then the mean transfers per rule is 31.1 and a median is 20. The model α_{10} will be able to make a prediction when 33% of the rule is completed on average, or when 50% of the rule is completed in the most typical case. In the worst case, the rule will have 15 transfers and the model will need 75% of the rule to be completed before a prediction can be made. This will happen independently of model accuracy. The model will need at least 10 transfers finished in the rule to make a prediction. The hypothesis is predictions made by model $\alpha_1 0$

	$\tau = 0.1$	$\tau = 0.25$	$\tau = 0.5$	$\tau = 0.75$	$\tau = 0.9$	$\tau = 1.0$
α_2	0.029	0.072	0.145	0.218	0.257	0.284
α_3	0.044	0.111	0.224	0.337	0.398	0.439
α_4	0.064	0.157	0.308	0.453	0.530	0.579
α_5	0.079	0.198	0.384	0.552	0.639	0.691
α_6	0.096	0.236	0.451	0.633	0.727	0.780
α_7	0.113	0.274	0.513	0.705	0.799	0.851
α_8	0.128	0.310	0.574	0.769	0.858	0.906
α_9	0.144	0.350	0.631	0.823	0.901	0.940
α_{10}	0.163	0.391	0.687	0.862	0.922	0.954

Table 4.1: Summary of FoGP$(y, \hat{y}_{\alpha_k}, \tau)$ for different thresholds of τ and different α_k models. Different thresholds τ are calculated for different α_k models. The bottom right corner shows that for model α_{10}, 95.4% of the predictions present less than 100% relative error, while the top left corner shows that for model α_2, 2.9% of the predictions lie within $\pm 10\%$ of its real value.

over rules with less transfers will be more accurate than those made over rules with more transfers, but this is unproven yet. In any case, model $\alpha_1 0$ will not be able to make predictions over rules with less that 10 transfers.

Table 4.1 on page 37 summarize the results of the experiment. Each column shows the threshold τ and each row, the model α_k of the family, where $k - 1$ is the number of transfers in the rule that needs to be done before a prediction can be made with the model.

A more detailed analysis of the progress of the rules over time determined that only a fraction of the Rules TTC are co-linear with the first points of the progress of the rule, including the origin point. That is visible in Figure 4.5 on page 35. This Figure shows the progress of rules with Rule TTC $\leq$ 500 seconds. Rule progress is not linear in general. This non linearity have a negative impact in the performance of the model.

From this observation, family of model $\alpha\prime_k$ was created and tested using the same procedure used on the α_k family. Every member of the $\alpha\prime_k$ family is equal to its relative α_{k+1}, but the origin point where the rule is created is removed from the regression. Then, $\alpha\prime_2$ is equal to α_3, and the ending time of the first two transfers are used to make the linear regression but the origin point. An important consequence is that the models $\alpha\prime_k$ and α_{k+1} can make a prediction in the same stage of the progress of the rule, that is, when the transfer k is finished.

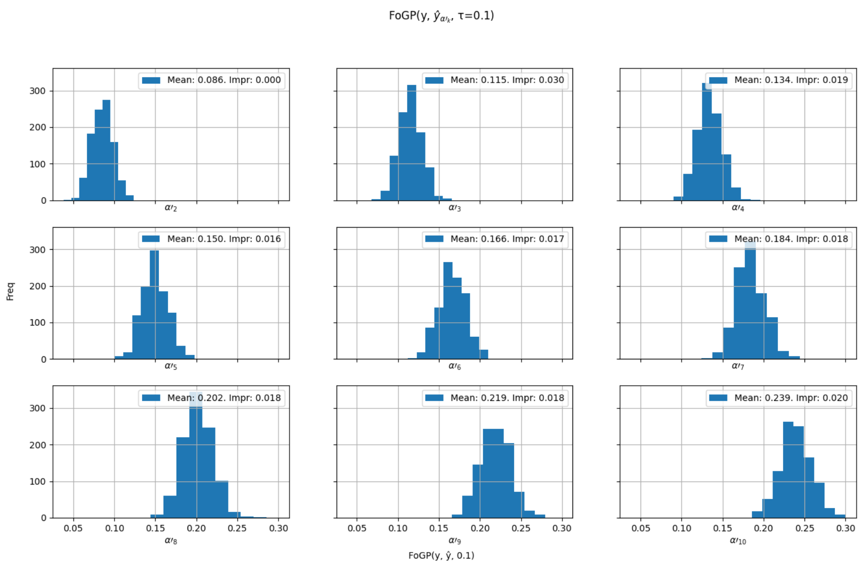


Figure 4.7: Distribution of the FoGP$(y, \hat{y}_{\alpha\prime_k}, \tau = 0.1)$ for the first 9 members of the $\alpha\prime_k$ family. Unlike Figure 4.6 on page 36, the k index represents the effective number of finished transfers of a rule before a prediction can be made. Hence, the results for model $\alpha\prime_2$ are comparable with those for model α_3, as both models can make a prediction only after the first two transfers of the rule are done.

	$\tau = 0.1$	$\tau = 0.25$	$\tau = 0.5$	$\tau = 0.75$	$\tau = 0.9$	$\tau = 1.0$
$\alpha\prime_2/\alpha_3$	0.086/0.044	0.213/0.111	0.425/0.224	0.655/0.337	0.801/0.398	0.886/0.439
$\alpha\prime_3/\alpha_4$	0.115/0.064	0.275/0.157	0.522/0.308	0.745/0.453	0.858/0.530	0.923/0.579
$\alpha\prime_4/\alpha_5$	0.134/0.079	0.317/0.198	0.580/0.384	0.792/0.552	0.892/0.639	0.948/0.691
$\alpha\prime_5/\alpha_6$	0.150/0.096	0.350/0.236	0.623/0.451	0.823/0.633	0.916/0.727	0.965/0.780
$\alpha\prime_6/\alpha_7$	0.166/0.113	0.382/0.274	0.662/0.513	0.850/0.705	0.932/0.799	0.977/0.851
$\alpha\prime_7/\alpha_8$	0.184/0.128	0.414/0.310	0.696/0.574	0.871/0.769	0.944/0.858	0.984/0.906
$\alpha\prime_8/\alpha_9$	0.202/0.144	0.447/0.350	0.730/0.631	0.888/0.823	0.953/0.901	0.988/0.940
$\alpha\prime_9/\alpha_{10}$	0.219/0.163	0.480/0.391	0.758/0.687	0.901/0.862	0.959/0.922	0.991/0.954

Table 4.2: FoGP$(y, \hat{y}_{\alpha\prime_k}, \tau)$ vs. FoGP$(y, \hat{y}_{\alpha_k}, \tau)$ comparison. Different thresholds τ are calculated for different $\alpha\prime_k$ and α_k models. Comparison criteria is based on number of finished transfers in the rule the model needs to make a prediction. $\alpha\prime_k$ and α_{k+1} are two models that need at least k transfers done. Top left corner shows that while for model α_3 only 4.4% of the predictions lays within 10% of it is real value, for model $\alpha\prime_2$, 8.8% of the predictions lays in the same range. This represents an improvement of 95.4% of model $\alpha\prime_2$ with respect to α_3.

Figure 4.7 on page 38 shows that the FoGP improves significantly if the origin is not included. The results for $\alpha\prime_k$ model is comparable with the same results for α_{k+1} model in Figure 4.6 on page 36, as both models are able to make a prediction of the Rule TTC only after at least k transfers of the rule are finished.

Table 4.2 on page 38 shows a summary that compares the performance of the first models of the $\alpha\prime_k$ and α_{k+1} family using the FoGP metric. In the top left corner we can see that while both models $\alpha\prime_2$ and α_3 can make a prediction after the second transfer of the rule is done, only 4.4% of the predictions of α_3 are within 10% of it is real value, while 8.6% of the predictions of $\alpha\prime_2$ are within the same margin of error. That represents an improvement of 95.4%. As expected, this margin of improvement gets smaller the more points are added to the regression. The $FoGP(y, \hat{y}_{\alpha\prime_9}, \tau = 0.1)$ vs. $FoGP(y, \hat{y}_{\alpha_{10}}, \tau = 0.1)$ for the same 10% threshold of relative error is 99.1% vs. 95.4% respectively, an improvement of $\alpha\prime_9$ over α_{10} of only 3.8%.

Chapter 5

Model of Rule TTC based on time series analysis

Chaos: When the present determines the future, but the approximate present does not approximately determine the future.

Edward Norton Lorenz

5.1 Problem framing

One important caveat with the model described in Chapter 4, starting on page 30 is that it will not be able to make any prediction for the Rule TTC before at least two transfers of the rule have finished. In order to estimate the Rule TTC at rule creation time, another approach is needed.

One approach is to use metrics such as the mean or the median of the Rule TTC of already finished rules as the predictor of Rule TTC of the newly created rules. Figure 5.1 on page 41 shows an example of how this method could work. Assume we want to predict the Rule TTC of the rule R_6 created at time t_0, with a window of 30 seconds, and the mean of the Rule TTCs of the those rules created between t_0 and $t_0 - 30$ seconds. Rules R_1 to R_5 were created over the last 30 seconds, and the mean TTC is 3.8 minutes, so that will be the prediction for the Rule TTC of R_6. This approach is implemented in model family β, detailed in section 5.2.

The main caveat of this technique is straightforward. None of the Rules R_1 to R_5 will be completed at t_0, so their TTCs will be unknown and not

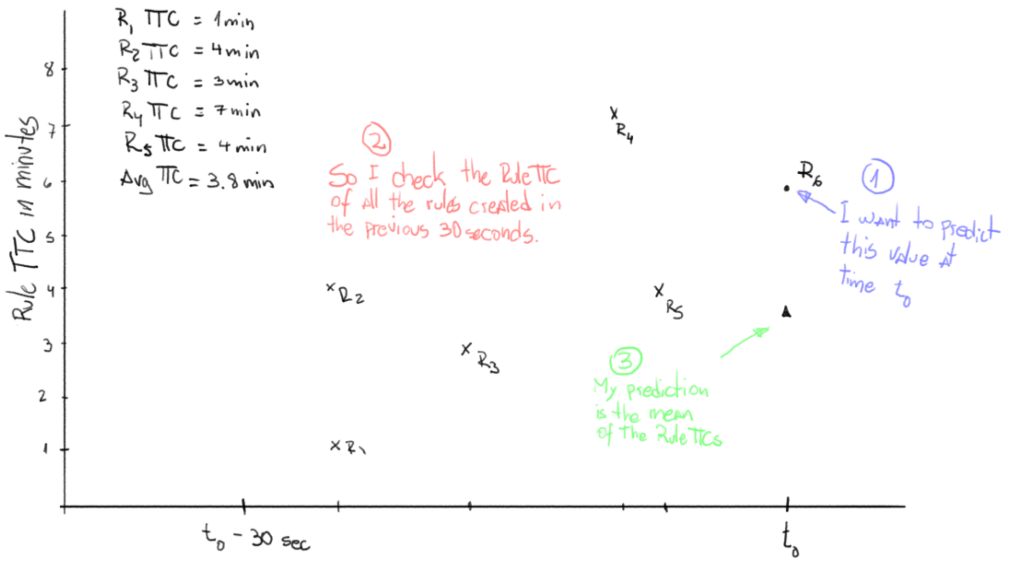


Figure 5.1: The Rule TTC of rule R_6 prediction, based on the mean of the Rule TTC of those rules created 30 seconds before rule R_6 creation.

available to make any prediction, and thus the β model cannot be implemented to be used in real time. The idea of this work is to use time series analysis techniques in order to predict the 3.8 minute mean in order to use it as predictor of the rules created at t_0. Models that use time series analysis are implemented in model family γ, in section 5.3.

Time series analysis is a proven technique used in diverse fields, such as econometrics, weather forecasting, earthquake prediction, or astronomy, that allows to forecast the values of a time series, that is, a sequence of observations per unit of time at regular intervals, using known values of the past.

In the following sections, the validity of the β model is tested taking advantage of that the real TTC values of all the rules are known, as this analysis is based on Rucio's historical data. Then, the γ models using TSA techniques is discussed and tested. A possible implementation of a real time γ model is presented. Finally, both β and γ model performance are compared.

5.2 The β_μ models

Formally, the $\beta_\mu(t_0, \rho)$ model family is defined as in Equation 5.1 on page 42. Here, y_{R_i} are the real Rule TTC of those rules created in the left-closed interval $[t_0 - \rho, t_0)$, that is, the rules in the dataset with `min_created` $< t_0$ and `min_created` $\geq (t_0 - \rho)$. The parameter ρ can be interpreted as the size of the rolling window, usually measured in seconds. The aggregation function μ will be a function that returns a number that represents a summary of the information contained in the $\{y_{R_i}\}$ set. The μ functions tested are $min()$, that returns the minimum element of the set, $max()$, that returns the maximum element of the set, $median()$, that calculates the arithmetic median, and $mean$, that returns the mean of the values of the set.

For clarity, we will often refer to $\beta_\mu(\rho)$ avoiding to mention the t_0 parameter when discussing the performance of the model. The t_0 parameter will be implicitly referenced at the creation time of the rule being predicted. However, to evaluate the performance of the model, several predictions are made with a random set of rules with very different t_0, and so the parameter is meaningless when predictions for several rules are compared.

$$\beta_\mu(t_0, \rho) = \mu(\{y_{R_1}, y_{R_2}, ...\}) \tag{5.1}$$

Using this notation, the example in Figure 5.1 on page 41 can be annotated as $\beta_{mean}(\rho = 30)$.

A correlation between the Rule TTC of rules created at t_0 and the Rule TTC of those rules created at some point in the past has been observed in preliminary studies of the data. To check if this correlation actually exists four members of the β_μ family were evaluated, scanning the ρ space. The tested models were β_μ with $\mu \in \{min(), median(), mean(), max()\}$.

In the experiment, we select a random sample of 300 rules and predict its Rule TTC with each of the models, using several values of the time window ρ. We call y to array of observed 300 Rule TTCs, and $\hat{y}_{\beta_\mu(\rho)}$, to the array of 300 predictions done using the model $\beta_\mu(\rho)$. For each of these arrays, the FoGP$(y, \hat{y}_{\beta_\mu(\rho)}, \tau = 0.1)$ metric is computed. This experiment was repeated 100 times.

Figure 5.2 on page 43 summarizes the result of the experiment. The ρ parameter were selected to explore space and test the predictive power of each model. The FoGP metric is calculated for the same 300 random rules for every window size ρ. The experiment is repeated 100 times. Orange lines are the median FoGP of the experiment and green triangles are the mean FoGP, for every window size. The observed min/median/mean/max Rule TTC of the sample was used to make a prediction. This value usually is not available in real time, i.e.: the rules created during the previous seconds usually take several minutes to complete. The result with better FoGP was obtained by the β_{median} model with ρ values between 20 and 30 seconds, through an FoGP$(y, \hat{y}_{\beta_\mu(\rho)}, \tau = 0.1) = 0.22$, meaning around 22% of the

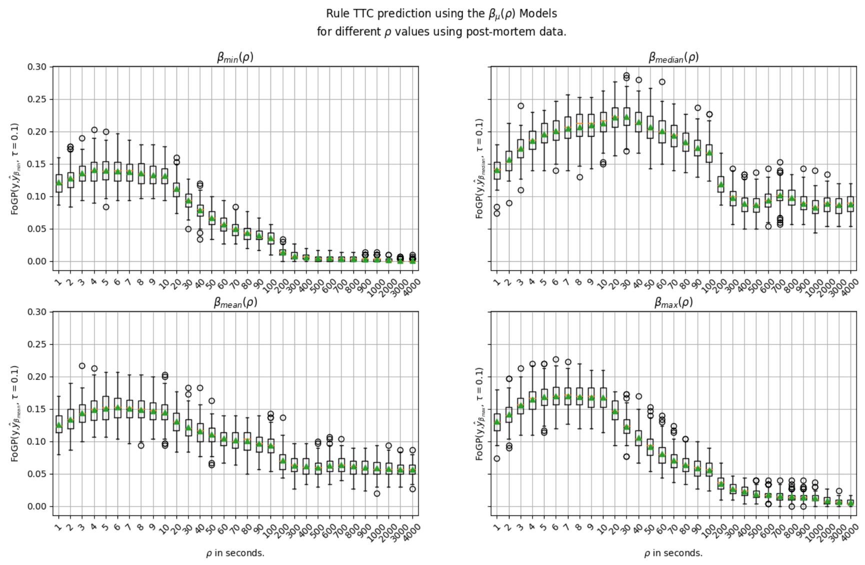


Figure 5.2: Distribution of the FoGP for the prediction of the Rule TTC of the models β_μ for μ one of the min(), median(), mean(), or max(), as a function of the window size ρ.

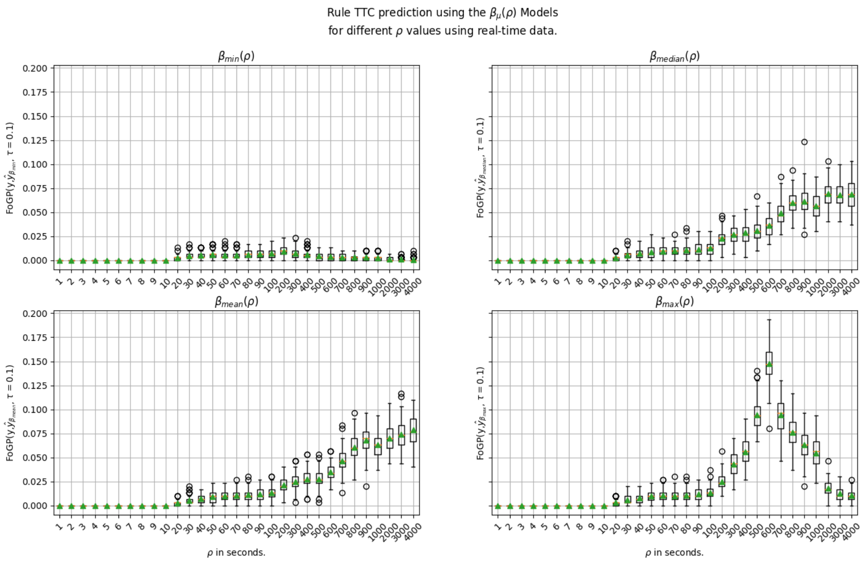


Figure 5.3: Distribution of the FoGP for the prediction of the Rule TTC of the models β_μ for μ one of the min(), median(), mean(), or max(), as a function of the window size ρ but using as input of the model the real time data available at t_0, that is excluding the TTC of the rules that finished after t_0.

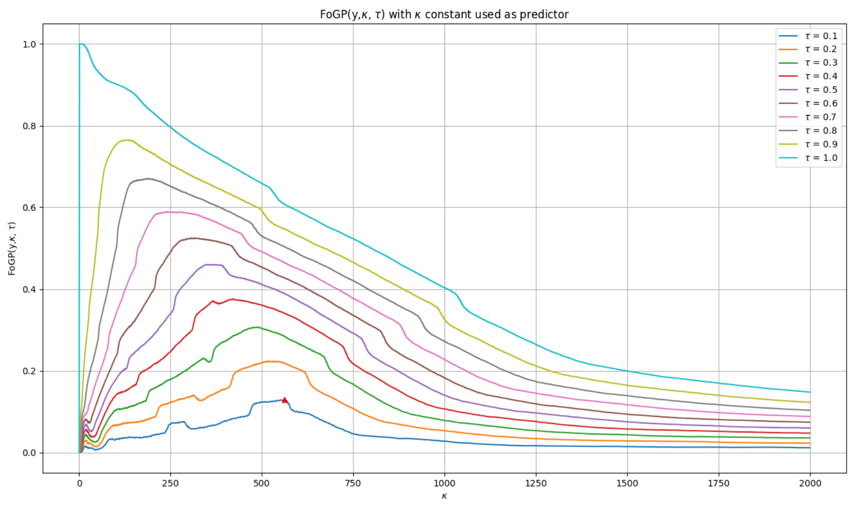


Figure 5.4: FoGP of of the Rule TTC prediction using a constant value κ. For each constant value κ, the metric FoGP(y, κ, τ) is calculated using different values of τ. The peak at FoGP$(y, \kappa = 562, \tau = 0.1) = 0.129$ (red triangle) means that 12.9% of the rules finish in 562 ± 56.2 seconds.

predictions will present less than 10% of relative error. Other βs present lower FoGP, and thus, lower predictive power. A new scanning of ρ in the interval [20, 30] shows no significant improvement in the FoGP. This also shows that the values of the μ aggregation function had some correlation with time and that these values could hold important information about the Rule TTC of near future rules.

If the β_μ model is implemented using only the data at real-time at t_0, that is, at the time to make a prediction for a rule created at t_0, then the only TTCs available will be of the ones of those rules with created and finished in the semi-open interval $[t_0 - \rho, t_0)$, that is, the rules with `min_created` $\geq (t_0 - \rho)$ and `max_ended` $< t_0$. Figure 5.3 on page 43 summarizes the results of the experiment of measure the FoGP over 300 Rule TTCs predictions, repeated 100 times. The β_μ function using the TTC of those rules that started between $t_0 - \rho$ and t_0 but also finished before t_0. This simulates the real time data available in the system to make the prediction, as those times beyond t_0 are in the future and Rule TTC data for the rules finishing after t_0 is usually not available. The FoGP metric is calculated for the same 300 random rules for every window size ρ. The experiment is repeated 100 times. Orange lines are the median FoGP of the experiment and green triangles are the mean FoGP, for every window size. The μ function were calculated using real time data. If ρ is small, usually the prediction is zero. That is because at the number of rules created and finished in the interval $[t_0 - \rho, t_0)$, is zero.

Surprisingly, the $\beta_{maximum}$ model peaks the FoGP when a window time of 600 seconds is used. This can be explained considering that it is possible to make a prediction for every single rule using a constant value for several rules and then calculate the FoGP. We call the model that predict a constant Rule TTC of κ seconds, κ model. We annotate the performance of that model at $\tau = 0.1$ as FoGP$(y, \kappa, 0.1)$. Figure 5.4 on page 44 shows the result of the FoGP for a prediction with a several members of the κ model family. When $\kappa = 562$ model is used to make a prediction, then the FoGP at $\tau = 0.1$ peaks at 0.129, meaning $\kappa = 562$ is the best constant model for $\tau = 0.1$. When the ρ is set to 600 seconds, the $\beta_{maximum}$ model usually select the maximum Rule TTC that will be compatible with the results obtained by the $\kappa = 562$ model.

5.3 The γ_μ models

As mentioned previously, model β_μ implemented using real time data, has low FoGP, as the μ aggregation function depends on real observed Rule TTC, and real time aggregation is not representative of the future Rule TTCs. However, as was demonstrated in the previous section, there is a time dependency in the aggregated values of the Rule TTCs. Model γ_μ presented here uses a forecast of the time series of the aggregation function μ in order to predict the Rule TTC of the new rules. Special attention was put in the $median()$ and $mean()$ aggregation functions, both because of its hypothetical predictive power and its good statistical properties.

Formally, the γ_μ model is defined as in Equation 5.2 on page 46 where $\hat{\mu}$ is the estimation of the function $\mu(y_{R_i})$ through the use of an auto-regressive model. As in the β_μ model, the set y_{R_i} is the Rule TTC of those rules created in the left-closed interval $[t_0 - \rho, t_0)$, and that have finished before t_0 in order to not include the information from the future. This equation is very similar to the β_μ equation, but in the γ model, the μ function is estimated using an auto-regressive model with λ lags of size ρ seconds. The ψ parameter represents the lookback, or how many lags are used to fit the model. The ω parameter represents the lookahead, or how many lags in the future the model will predict. As with β_μ models, t_0 represents the time at which the prediction is made and it is ignored when the performance of the γ_μ model is being discussed.

$$\gamma_\mu(t_0, \rho, \lambda, \psi, \omega) = \hat{\mu} \tag{5.2}$$

The algorithm proceeds as follows. First, all the rules that have been created between t_0 and $t_0 - \rho\psi$ seconds and that have finished before t_0 are selected. That is, all the transfers in the rules dataset which satisfy the conditions $t_0 - \rho\psi \leq$ `min_created` $< t_0$ and `max_ended` $< t_0$. The rules dataset contains the fields `min_created` and `max_created` time stamps that represents the time when the first transfer and last transfer of the rules were created. Thus `min_created` time stamp is equal to the time creation of the rule and the minimum `min_created` is the time of creation of the first rule in the dataset. Also, the `max_ended` is the time stamp of the last transfer to finish, and thus, the finishing time of the rule. The μ function is calculated over the bins of length ρ seconds, being the value of the first bin, the μ of the Rule TTC of the rules satisfying the condition $t_0 - \rho\psi \leq$ `min_created` $< t_0 - \rho\psi + \rho$, the value of the second bin the μ of the rules satisfying the condition $t_0 - \rho\psi + \rho \leq$ `min_created` $< t_0 - \rho\psi + 2\rho$, and so on. This generates a time series $\sigma_{\hat{\mu}}$ of frequency ρ with a total of ψ/ρ samples. This time series differs from the real time series σ_μ in that the μ Rules TTCs for the lags closer to t_0, due to the filter for select rules described before. Once the time series is obtained, a standard auto-regressive model $AR(p)$[17] is fitted using using the first $\psi/\rho - \omega$ samples. The parameter of the auto-regressive model is $p = \lambda$, meaning the model will need λ samples in order to make a prediction. Model training and prediction is implemented using the `AutoReg` function from the *Python statsmodels v0.11.1* package[18]. Once the model is fitted, a forecast is made using the last $\lambda - \omega$ samples in order to predict the following ω lags. The $\hat{\mu}$, that is, the prediction of the γ_μ will be the last value of the returned forecast.

Figure 5.5 on page 47 shows the case for $\gamma_{median}(\rho = 30, \lambda = 30, \psi = 90, \omega = 16)$ model. In this case, the t_0 is the date 2019-06-12 22:54:31. The transfers selected to create the time series were those satisfying the conditions $t_0 - 30 * 90$ seconds $\leq$ `min_created` $< t_0$ and `max_ended` $< t_0$. The median of the Rule TTC were calculated to get the time series using bins of 30 seconds. The $\sigma_{\hat{\mu}}$ time series is showed in orange while σ_μ time series values, that is the median of the Rule TTC including those of the rules that end after t_0 is plotted in blue. These time series are very similar except in the last minutes before t_0. The main reason for this discrepancy is that rules created some minutes before t_0 only finish after t_0 and are excluded because they do not satisfy the condition `max_ended`$< t_0$. This will happen in a hypothetical implementation of this method were everything after t_0 is unknown as it is in the future.

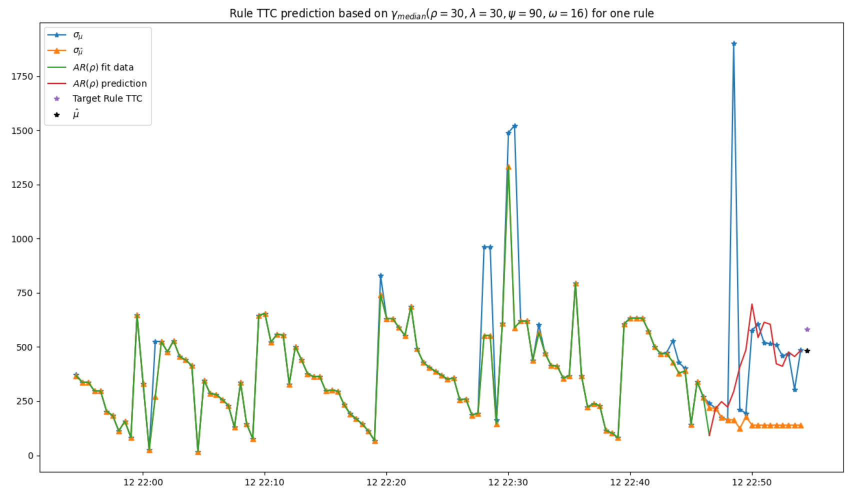


Figure 5.5: Time series process to forecast the TTC of one Rule. The rule to predict was created on 2019-06-12 22:54:31 or t_0. Marked with a purple asterisk at the lower right part of the plot lays its real Rule TTC. Below there is a black asterisk that correspond with the prediction done by the $\gamma_{median}(\rho = 30, \lambda = 30, \psi = 90, \omega = 16)$ model. The real median Rule TTC Time Series is plotted in blue. The orange line is the observed median Rule TTC, or the TTC of those rules that are finished before t_0. The green line in the plot is section of the data used to fit the γ_{median} model. The red line correspond to the prediction made for the model for 16 lags ahead of $t_0 - 8$ minutes.

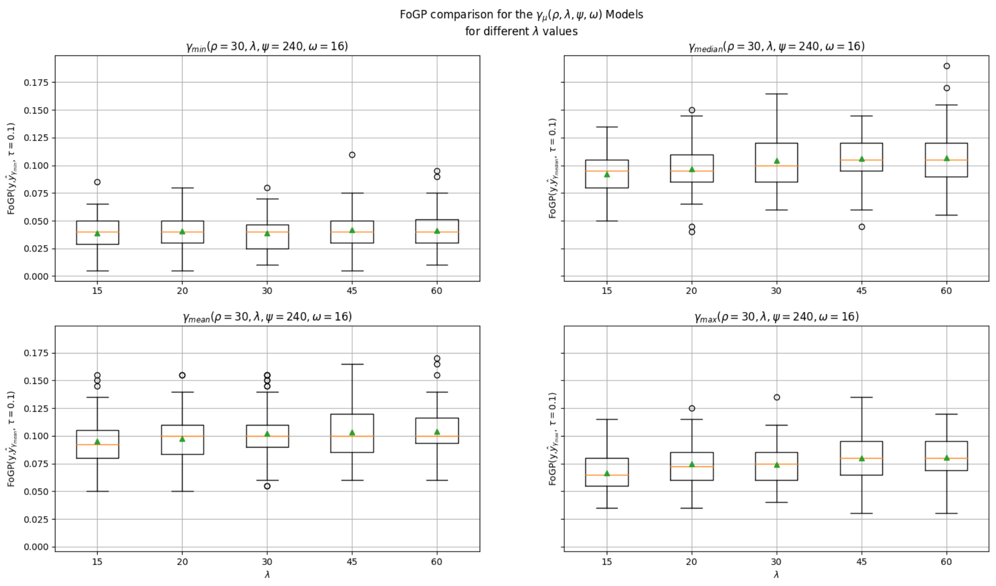


Figure 5.6: FoGP comparison of different γ_μ models when predicting the Rule TTC. The boxes show the distribution of the FoGP$(y, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}, \tau = 0.1)$ for ρ, ψ, and ω fixed and a range of values of λ.

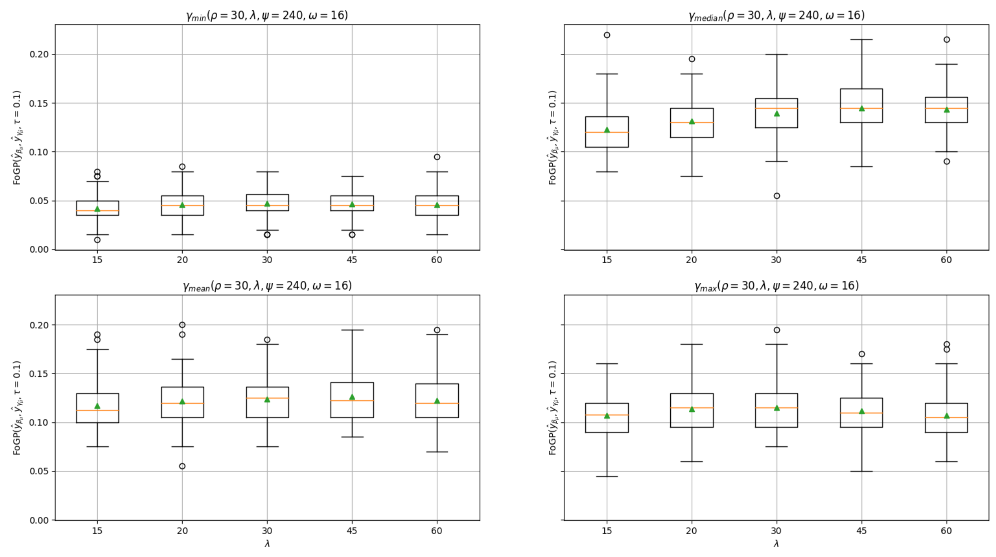


Figure 5.7: FoGP comparison of the different γ_μ models when predicting the value of the function μ, that is, the output of the β_μ model. The boxes show the distribution of the FoGP$(\hat{y}_{\beta_\mu(\rho)}, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}, \tau = 0.1)$ for ρ, ψ, and ω fixed and a range of values of λ.

The ω parameter for the experiments was set to 8 minutes. This threshold was estimated based on the median Rule TTC of the rules dataset. This means that most of the rules created until 8 minutes before t_0 will be finished at t_0. All the tested models fix this parameter to represent 8 minutes but as it depends on ρ, it is different for every model. The parameter is calculated using the formula $\omega = 8 * 60/\rho$. That is, for the model $\gamma_{\hat{\mu}}(\rho, \lambda, \psi, \omega)$ with $\rho = 30$ seconds, ω will be 16 lags and for the models with $\rho = 60$ seconds, ω will be 8 lags.

For each prediction, a slice of 45 minutes of data is taken. The first 74 lags of $\sigma_{\hat{\mu}}$ which represents 37 minutes are used to fit the model and that is showed in the green line. Using the last 30 lags or 15 minutes of this series, the model forecast 16 lags, shown with a red line. In this example, the value of $\hat{\mu}$ is very close from the real μ, the end of blue and red lines. This value will be used to approximate the Rule TTC marked as a purple asterisk in Figure 5.5 on page 47.

We call y the vector of the true Rule TTCs of a random set of rules, $\hat{y}_{\beta_\mu(\rho)}$ the vector of predictions of y using model $\beta_\mu(\rho)$, and $\hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}$ a vector of predictions using the $\gamma_{\hat{\mu}}(\rho, \lambda, \psi, \omega)$ model.

The (ρ, λ, ψ) parameter space was grid searched for possible peaks in FoGP$(y, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}, \tau = 0.1)$, but also in FoGP$(\hat{y}_{\beta_\mu(\rho)}, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}, \tau = 0.1)$. The first metric describes how accurate the γ_μ model is to predict the Rule TTCs and with respect to other models as α and β_μ. Second metric will describe how accurate the $AR(\rho)$ model is to predict the value of μ that model β_μ uses to make its prediction.

The evaluation experiment is conducted as follow. First, 200 random rules are selected from the rules dataset of 1.8 million rules. Then, 200 predictions are made using the model $\gamma_{\hat{\mu}}(\rho, \lambda, \psi, \omega)$ with μ being the minimum, median, mean and maximum functions and with a particular combination of the parameters ρ, λ and, ψ and the FoPG with respect to y and with respect to $\hat{y}_{\beta_\mu(\rho)}$ is calculated. Then another model is picked and tested against the same set of rules. When all the combination of parameters of ρ equal to 30, 60 and, 90 seconds, λ equal to 15, 20, 30, 45 and 60 lags and ψ of 120, 240 and, 480 lags are exhausted, another set of 200 rules are selected. The experiment is repeated 100 times. This means all the models are tested against the same set of rules.

Figure 5.6 on page 48 summarizes the results of the experiment for $FoGP(y, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)}, \tau = 0.1)$ for a particular choice of the parameters, that is, how good the models $\gamma_{\hat{\mu}}(\rho, \lambda, \psi, \omega)$ are in order to predict the Rule TTC of the rules, for μ being the minimum, median, mean and maximum functions. Figure 5.6 on page 48 can be interpreted as follows. In the lower left plot, for the model $\gamma_{\hat{mean}}(\rho = 30, \lambda = 45, \psi = 240, \omega = 16)$, on average, a bit over

10% of the TTC predictions made with the model will have less than 10% relative error. The $(\rho, \lambda, \psi, \omega)$ parameters where selected using a grid search in order to maximize the FoGP at $\tau = 0.1$ for the median function and can be sub-optimal for other μ functions.

Figure 5.7 on page 48 summarizes the results for FoGP$(\hat{y}_{\beta_\mu(\rho)}, \hat{y}_{\gamma_{\hat{\mu}}(\rho,\lambda,\psi,\omega)},$ $\tau = 0.1)$ for a particular choice of the parameters. Instead of compares the predictions of the model $\gamma_{\hat{\mu}}$ against the observed Rule TTC as in Figure 5.6 on page 48, Figure 5.7 on page 48 compares the $\gamma_{\hat{\mu}}$ predictions against the real $\beta_\mu(\rho)$, showing the accuracy of the model to predict the function μ of the Rule TTC of those rules that have not finished at t_0.

Chapter 6

Model of Rule TTC based on deep neural networks

Truth... is much too
complicated to allow for
anything but approximations.

John von Neumann

6.1 The δ_n Model

In his book "Deep Learning with Python"[19], Francois Chollet introduces a deep neural network (DNN) architecture able to predict the temperatures for the Jena Dataset[20] slightly better than a naive model. A similar approach was tried in order to predict the Rule TTC, based on the time series of a number of observables we suspect could determine the TTC of such rule at its creation time. This study should not be considered exhaustive, but a exploratory study about the use of deep neural networks to predict the Rule TTC.

From the rules dataset it is possible to create a set of time series from observables which can influence the Rule TTC. The minimum, median, mean, and maximum Rule TTC of previous transfers demonstrate at least some predictive power and have been used in previous models with limited success. Other variables that could influence the Rule TTC of future rules is the amount of transfers pending and also the amount of bytes pending. A way to calculate this is by extending the routines to calculate the time series for the minimum, median, mean, and maximum functions. The bins are filled with the sum of the bytes or the sum of the transfers of each rule for both time

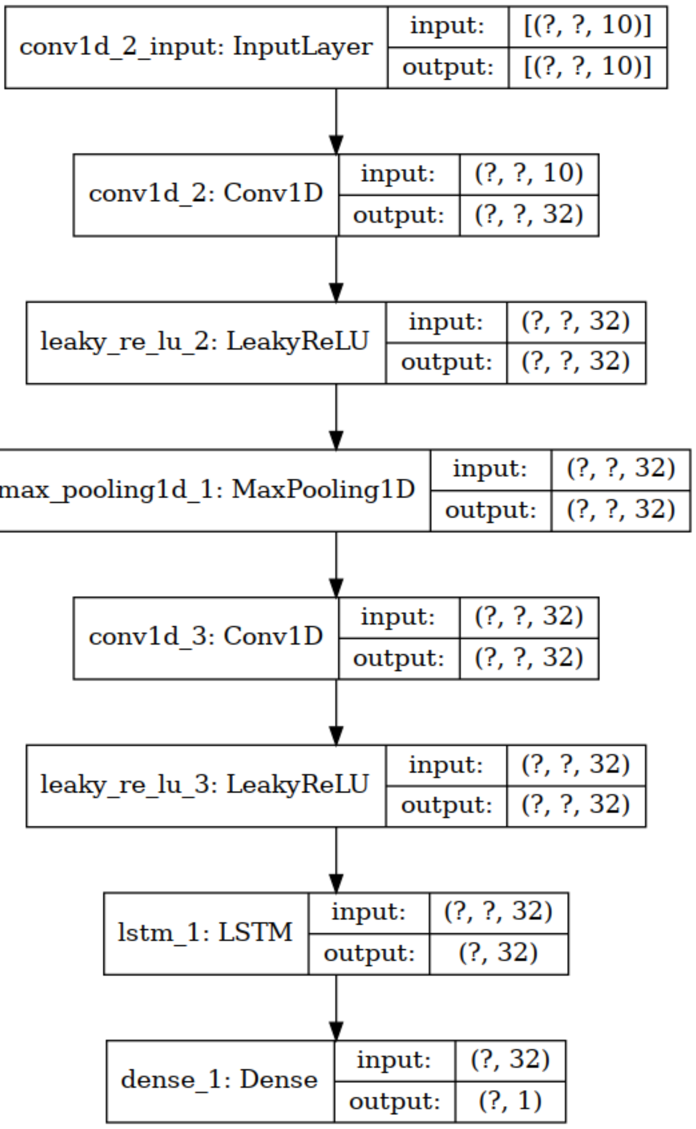


Figure 6.1: Chollet Jena Temperatures Model adapted to solve the Rule TTC problem. The LSTM layer replaces the GRU layer from the original model.

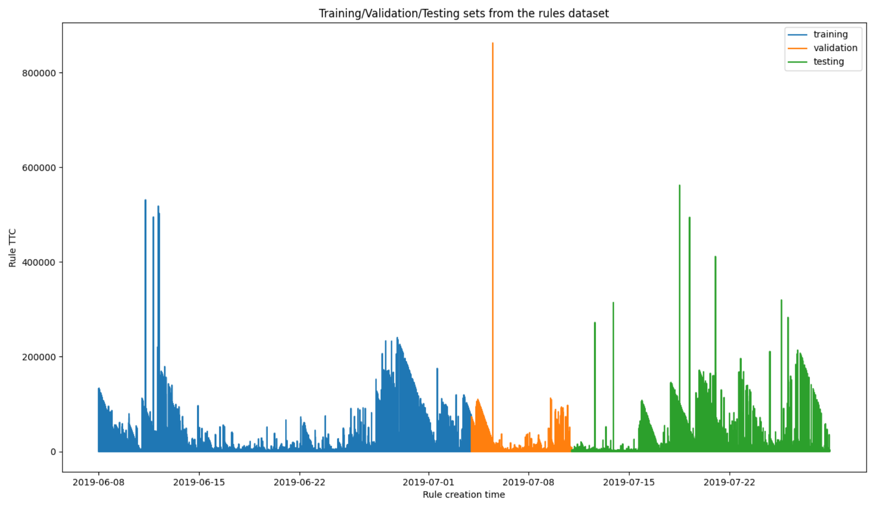


Figure 6.2: Rule dataset split for training, validation and testing

series, the observed and the real one. The difference between the two will be the time series of unfinished transfers and unfinished bytes. These values are known at rule creation time or can be approximated. As the majority of rules affects closed datasets, that is datasets over which new files can not be added, the number of files to be transferred and the size of each is mostly known at rule creation time.

We develop a model with a very similar structure of that proposed in Chapter 6 in [19]. We call it δ_n, where n is the number of convolutional filters and the number of LSTM neurons. Figure 6.1 on page 52 shows the architecture of the δ_{32} model. The main difference with regard to the original Chollet model is the substitution of the GRU layer for a LSTM layer, as was a proposed improvement suggested in the book. The input of the model consists of 10 channels, each of which represents the time series of some attribute calculated between t_0 or the rule creation time, and $t_0 - 120$ minutes, in bins of 30 seconds. The attributes used to build this time series were the minimum, median, mean, and maximum Rule TTC of each bin, plus the sum of transfers and bytes of both, finished, created, and pending rules. Each model was implemented using the Keras/TensorFlow Python API and trained for 120 epochs using the RMSProp optimizer to minimize the Mean Absolute Error loss function.

Training and validation data were selected based on the distribution of the the Rule TTCs at creation time. The training data consists of all the rules created between June 8th 2019 to July 3rd 2019. The validation set includes the rules created between July 4th 2019 and July 10th 2019. And finally, testing set includes the rules created between July 11th 2019 and July 29th 2019. Figure 6.2 on page 53 shows the split for training, validation, and testing sets.

6.2 The $\delta\nu\nu_n$ Model

The δ_n model family does not take into account information about the rule of which we want to predict the TTC, that is, the model does not include information about the target rule. In this section we present a model that includes the number of transfers the target rule consists of, the sum of bytes of all this transfers, and the links this transfers will affect, that is, the list of sources and destinations for all the transfers. Unlike the time series information fed to the previous model, the data about the target rule is point wise, such that it is not data about the past state of the system, but of the present or t_0 time.

The model has 3 inputs, the several time series representing the past of the system, the sum of bytes and the number of transfers of the rule, and the list of links affected by those transfers. The only output of the system will be the Rule TTC. This kind of models cannot be implemented using the Keras Sequential Model. Instead, the Keras Functional API was used to conceive a family of models capable of handling the different type of inputs. We call this family the $\delta\nu\nu_n$ model family, where the n parameter is the number of convolutional filters or the number of LSTM neurons of the model.

Figure 6.3 on page 55 shows the architecture of the $\delta\nu\nu_{32}$ model. Data flows from top to bottom. The left branch is the Chollet-Jena (δ_{32}) model in charge of digesting the time series data. The center branch input is the number of transfers and sum of bytes of the target rule. The right branch input is a list of integers, each of which represents a link that will be affected by one of the transfers. This list is truncated to 50, so only the first 50 links are going to be accounted. If less that 50 links are used, the sequence is padded with zeros. There is a special need to convert the (`source, destination`) pairs into a unique number in order to feed the `emb_input` layer. This process is done in a prepossessing stage using the Keras Tokenizer tool. The alphabet of links is 8762 words of the form `SRCSITE__DSTSITE`. The LSTM layer after the embedding process the links orderly. Even though link order should not matter, that is, the order in which the links appear in the embedding should not determine or affect the Rule TTC, the usage of this layer has proven important because the prediction rate over the testing set is up to 2% better

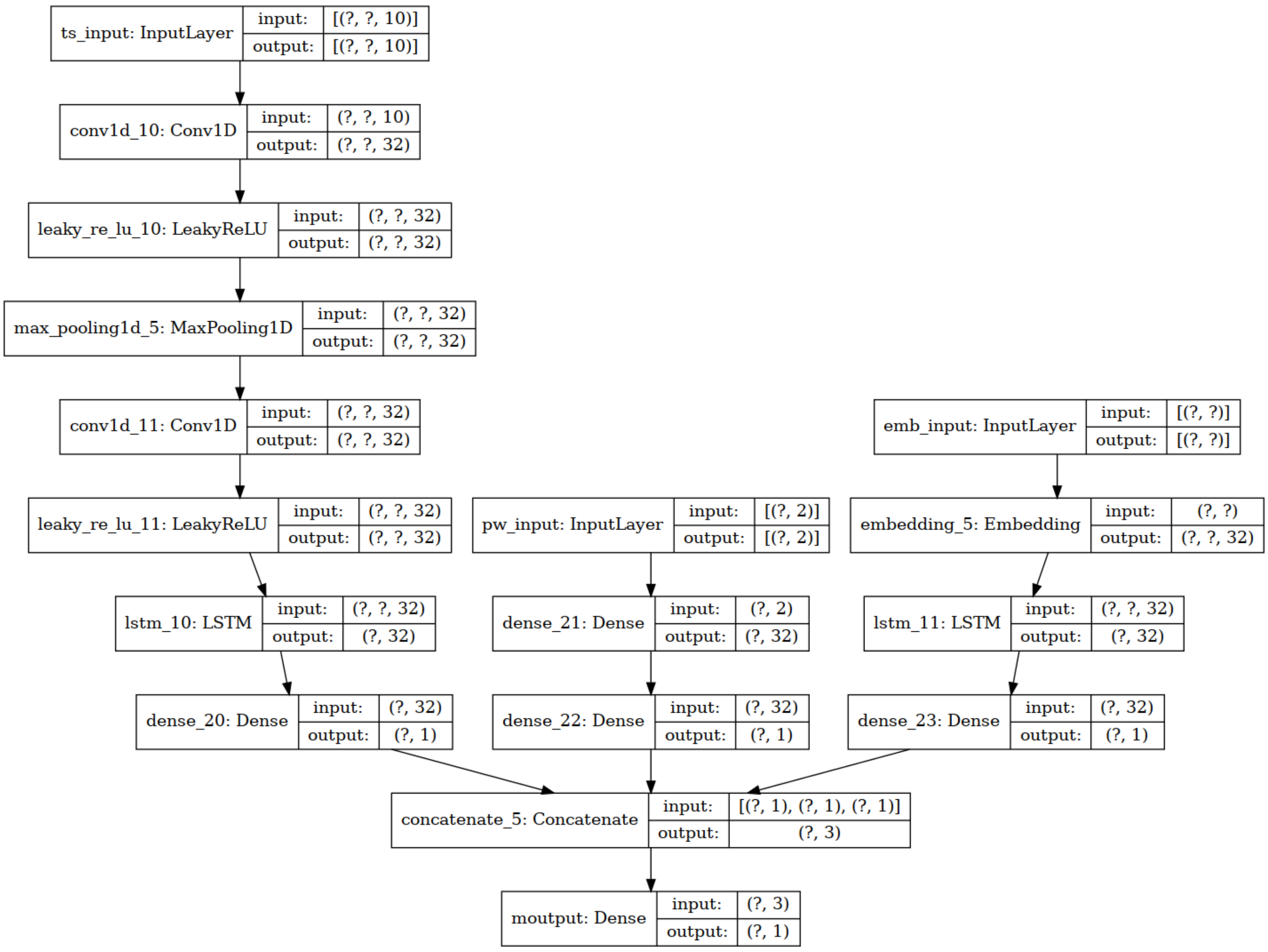

ts_input: InputLayer | input: [(?, ?, 10)] | output: [(?, ?, 10)]
conv1d_10: Conv1D | input: (?, ?, 10) | output: (?, ?, 32)
leaky_re_lu_10: LeakyReLU | input: (?, ?, 32) | output: (?, ?, 32)
max_pooling1d_5: MaxPooling1D | input: (?, ?, 32) | output: (?, ?, 32)
conv1d_11: Conv1D | input: (?, ?, 32) | output: (?, ?, 32)
leaky_re_lu_11: LeakyReLU | input: (?, ?, 32) | output: (?, ?, 32)
lstm_10: LSTM | input: (?, ?, 32) | output: (?, 32)
dense_20: Dense | input: (?, 32) | output: (?, 1)
pw_input: InputLayer | input: [(?, 2)] | output: [(?, 2)]
dense_21: Dense | input: (?, 2) | output: (?, 32)
dense_22: Dense | input: (?, 32) | output: (?, 1)
emb_input: InputLayer | input: [(?, ?)] | output: [(?, ?)]
embedding_5: Embedding | input: (?, ?) | output: (?, ?, 32)
lstm_11: LSTM | input: (?, ?, 32) | output: (?, 32)
dense_23: Dense | input: (?, 32) | output: (?, 1)
concatenate_5: Concatenate | input: [(?, 1), (?, 1), (?, 1)] | output: (?, 3)
moutput: Dense | input: (?, 3) | output: (?, 1)

Figure 6.3: FunnelNet architecture, used for the $\delta\nu\nu$ models.

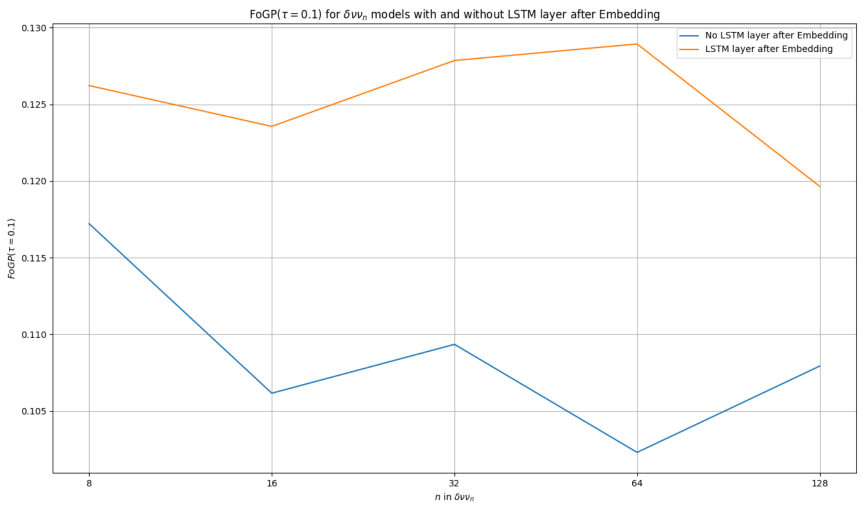


Figure 6.4: FoGP($\tau = 0.1$) for two variants of the $\delta\nu\nu_n$ models. Models without a LSTM layer after the embedding layer performs worse than the models with a LSTM layer after the embedding layer.

for the model family that use the LSTM layer, as shown in Figure 6.4 on page 56.

Normalization of all the numerical data was done using Equation 6.1 on page 56. This allows the model not to give more importance to some observables over others because of the scale. Typical normalization where values are subtracted the mean and divided by the variance is not enough in this case, due to the very long tail of the distribution of the values.

$$\ell = (ln(x) - ln(mean(x))/ln(std(x)) \tag{6.1}$$

Figure 6.5 on page 57 shows the histograms of the normalized vs. not normalized Rule TTC.

6.3 Comparison of the models performance

Both models were trained using the *EarlyStopping* callback that allows to monitor the progress of the validation loss. The callback stops training if there is no improvement after a fixed number of epochs and rolls back the weights to the ones of the last best model.

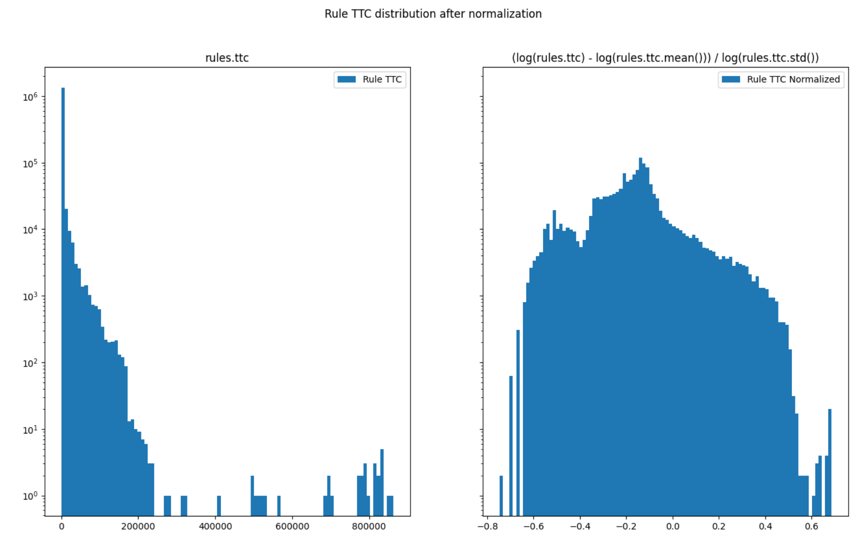


Figure 6.5: Distribution of the Rule TTC without normalization (left) with normalization (right) as defined in Equation 6.1 on page 56.

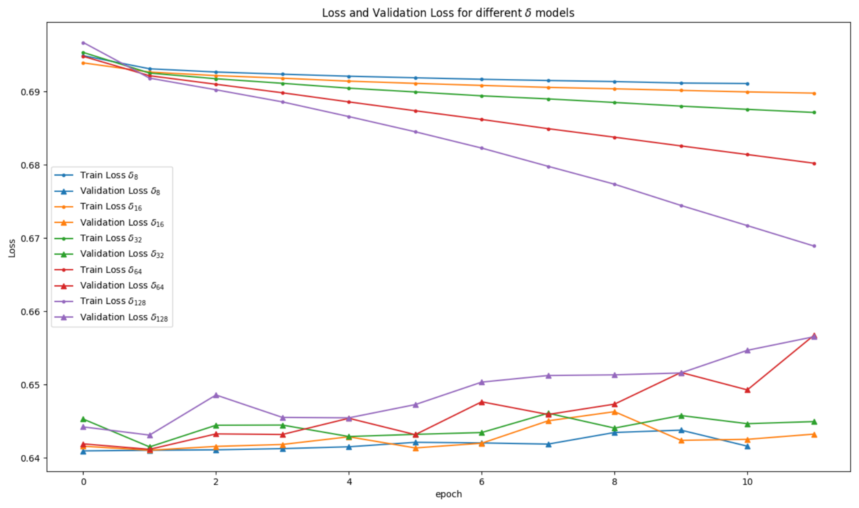


Figure 6.6: Training loss vs. validation loss for several δ_n models. Training was done by setting an EarlyStopping callback measuring the validation loss with patience of 10 epochs. This plot suggest the model is not able to learn from the training data.

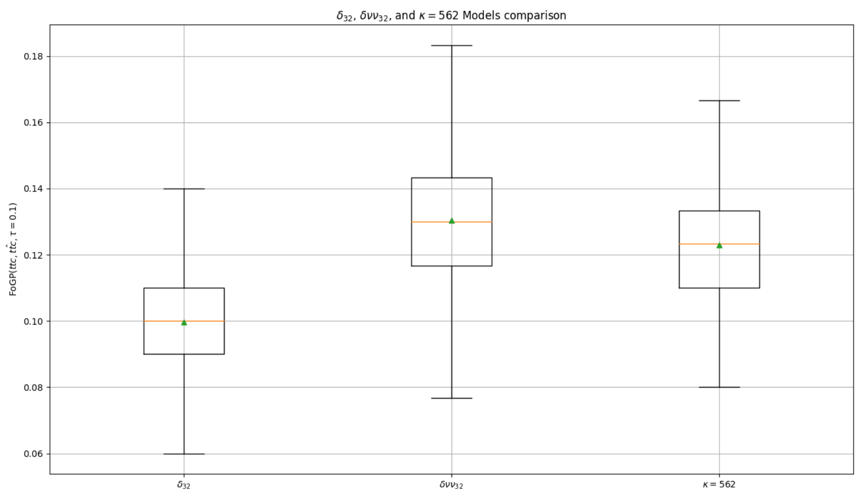


Figure 6.7: Distribution of 1000 FoGP$(y, \delta_{32}, \tau = 0.1)$ and FoGP$(y, \delta\nu\nu_{32}, \tau = 0.1)$. Each FoGP is calculated for the predictions made for 300 random Rule TTCs. The model $\kappa = 562$, the one that make a constant prediction for the Rule TTC of 562 seconds, was included for comparison. For this model 12.2% of the predictions have less than 10% relative error.

For δ_n models, the patience of the callback was set to 10 epochs. Figure 6.6 on page 57 shows the δ_n model family stops after 10 or 11 epochs, meaning the best model is obtained after only 1 or 2 training epochs. The δ_n models are not able to generalize, and if trained for more epochs, model predictions converge to values around 480.

For the $\delta\nu\nu_n$ models, EarlyStopping patience was set to 5. Figure 6.9 on page 59 shows that this model family learns from the training data until epoch 12 in the best case, that is for model $\delta\nu\nu_{32}$. After that, there is no improvement in validation loss. Naturally, the larger the model, that is the n, the faster the model overfits.

Figure 6.7 on page 58 shows the comparison of the FoGP$(y, \hat{y}, \tau = 0.1)$ of the δ_{32} and $\delta\nu\nu_{32}$ models. The model $\kappa = 562$ that predicts the constant optimal value of 562 seconds discussed in Section 5.2 on page 41, is included for comparison as a benchmark. A subsample of 300 rules were selected. For each rule, a prediction is made using all the models. Then, the FoGP$(y, \hat{y}_\delta, \tau = 0.1)$, FoGP$(y, \hat{y}_{\delta\nu\nu}, \tau = 0.1)$, and FoGP$(y, \hat{y}_\kappa, \tau = 0.1)$ was calculated. The procedure was repeated 1000 times. The box plot shows the

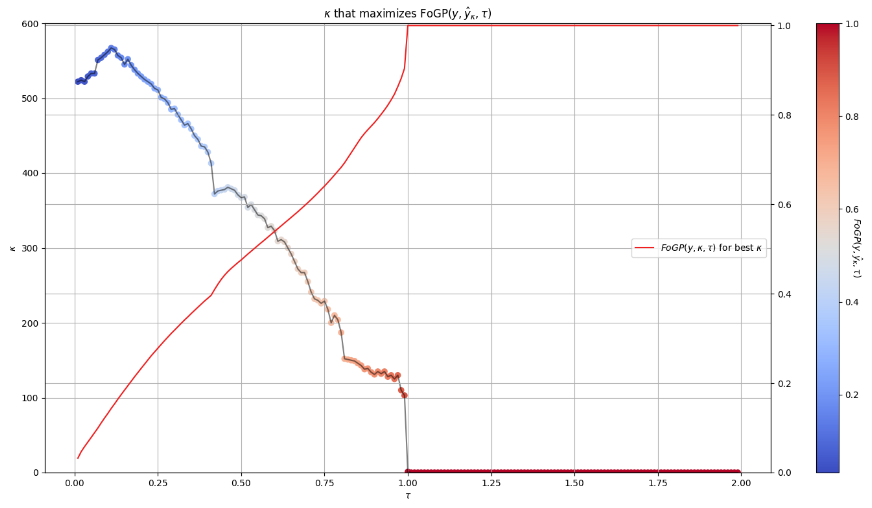


Figure 6.8: Constant prediction κ that maximizes the $\text{FoGP}(y, \hat{y}_\kappa, \tau)$ function. Black line represents the κ with maximum $\text{FoGP}(y, \hat{y}_\kappa, \tau)$ for a given τ. The colored points shows the value of the $\text{FoGP}(y, \hat{y}_\kappa, \tau)$ metric, the bluer the worse, the redder the better. The red line also represents the achieved FoGP with the y-axis on the right.

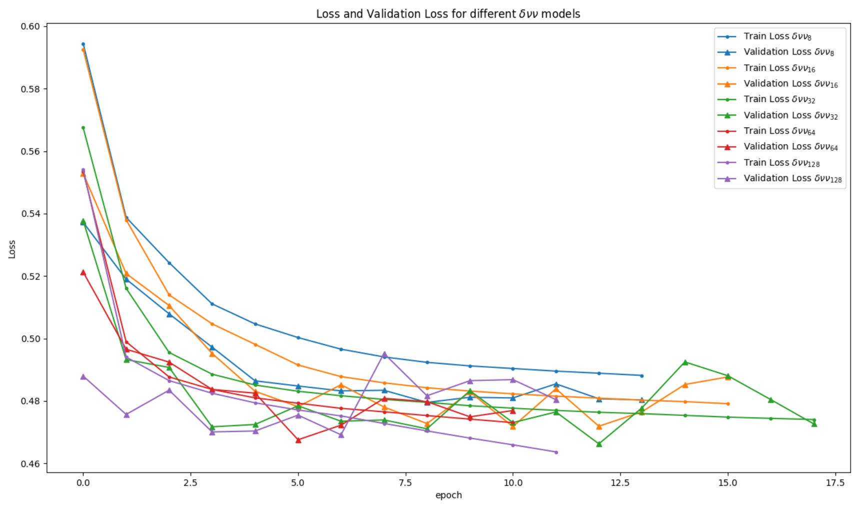


Figure 6.9: Training loss vs. validation loss for several $\delta\nu\nu_n$ models. Training was done setting an EarlyStopping callback measuring the validation loss with patience of 5 epochs.

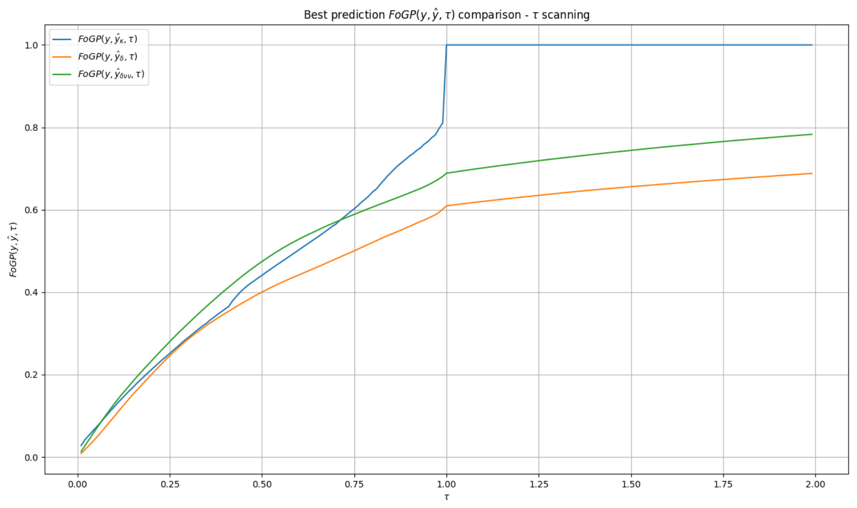


Figure 6.10: Fraction of Good Predictions for several models made over the testing set. For κ model, the best know constant for every τ was used. For τ bigger than 1, the κ model returns 0, meaning 100% of the predictions have less than 100% relative error.

distribution of the different FoGP obtained. Model $\kappa = 562$ shows that on average, 12.2% of the predictions are within a 10% relative error from the real value. The $\delta\nu\nu_{32}$ model outperforms the $\kappa = 562$ model by a modest 5.7% on average. Numbers show that, on average, 9.96% of the predictions made with model δ, also known as Chollet-Jena, have less than 10% relative error. Meanwhile, the 13.03% of the predictions made with model $\delta\nu\nu$, also known as FunnelNet, have less than 10% relative error.

It is educative to compare the models with several values of τ, especially in the range (0.01, 0.25), in order to see the fractions of predictions of each model with less than between 1% and 25% relative error. For the comparison with the κ model to be fair, the best constant for each τ must be selected in order to maximize the FoGP$(y, \hat{y}_\kappa, \tau)$. Using the same training data used to fit models δ and $\delta\nu\nu$, the κ/τ space was scanned calculating the FoGP$(y, \hat{y}_\kappa, \tau)$ in the ranges $\kappa = (0, 2000)$ in steps of 1 and $\tau = (0.01, 2.0)$ in steps of 0.01. This procedure gives place to a surface defined in $\mathbb{R}^3$ with a local FoGP$(y, \hat{y}_\kappa, \tau)$ maximum for each κ and τ. Figure 6.8 on page 59 shows this local maximum, that is, the constant that predicts the training set with the highest FoGP. Several things arise from this analysis. First, there is a peak at $\kappa = 567$ which does not correspond neither with the mean Rule TTC of the training set, that is 1962.1 seconds, nor with the median of 439 seconds. This means that both the prediction using the mean and the median are sub-optimal in terms of FoGP. Second, when $\kappa = 0$ the FoGP tops at 1.0, meaning that all the predictions have always a 100% relative error. The explanation for this effect is straightforward. If the prediction for the value of x is 0, then the relative error is calculated as $|x - 0|/x = 1$, meaning the error is 100%. As the FoGP measures how many predictions are less than τ, when $\tau > 1.0$, if the prediction is 0, all the predictions accounted as have less than 100% relative error. Third, the FoGP values in the τ range (0.01, 0.25) fall between 0.027 and 0.251, meaning between 2.7% and 25.1% of the predictions presents less than between 1% and 25% relative error.

When κ model is compared with δ and $\delta\nu\nu$ models, we found that the κ model outperforms δ and $\delta\nu\nu$ when τ is in the range (0.01, 0.04) and for τ bigger than 0.65. In the range (0.04, 0.65) the $\delta\nu\nu$ model is better. This results are shows in Figure 6.10 on page 60.

Both δ and $\delta\nu\nu$ models return continuous values and hence do not make any sense to measure the FoGP when $\tau = 0$ as the probability of the model to predict the exact value of the Rule TTC is almost zero. It makes sense to measure it for the κ model as the constant value is integer. This explains the better performance of the κ model for low values of τ. However, there is no noticeable change in the FoGP when flooring or ceiling the predictions of the δ and $\delta\nu\nu$ models.

Chapter 7

Network time to predict Transfer TTC and Rule TTC

There is nothing certain,
but the uncertain.

proverb

7.1 Network Time for a single transfer

This chapter addresses the question if it is possible to predict the Transfer TTC using the data available in the Rucio database. If so, the the following question would be whether or not its possible to predict the TTC of a given Rule knowing the TTC of individual transfers. Previous models centered the attention on the Rucio replication rules as a whole. However, rules are typically composed of several individual transfer requests.

The transfers dataset cited in [1] as described in Chapter 3, includes data at transfer level. Four timestamps are important to reconstruct the lifetime of a request, namely `created`, `submitted`, `started`, and `ended`. From these timestamps, three states in the transfer lifetime can be distinguished. From the moment where the transfer request is created until it is submitted to the File Transfer System (FTS) transfer tool, the request is under Rucio's responsibility. That means one of Rucio's daemons is responsible to submit transfer requests to FTS. These submissions can be delayed at Rucio's discretion, i.e., when a certain number of transfers have been already submitted to a particular FTS server, to avoid overloading it. Thus, the difference between `submitted` and `created` timestamps is known as Rucio Time or *RTIME*.

After the request is submitted to FTS, but before the actual transfer starts, the transfer request is under FTS' responsibility. FTS queues the requests, sorts them, and processes them per link. A request transfer could be delayed in FTS queues only at FTS Optimizer discretion. The FTS Optimizer is an algorithm that aims to increase network usage between links, by managing the number of concurrent transfers a link can handle based on transfer failures on that link. If a link presents a failure, the FTS Optimizer reduces the number of concurrent transfers and vice versa. The difference between the `started` timestamp and `submitted` timestamp is known as FTS Queue Time, or *QTIME*.

Once FTS triggers the transfer request, the transfer of the file begins. During this period the transfer will be using the network resources. The difference between `ended` and `started` timestamps is known as Network Time or *NTIME*. Intuition tells us that the bigger the file the longer the Network Time will be, and the wider the link the faster the transfer for transfers of the same size will be. However the number of concurrent transfers, that is the number of files being transferred at the same time using the same link, also influence the Network Time. Rucio knows the `created` and `submitted` timestamps almost instantly after the events have happen. The `started` and `ended` timestamps are known to Rucio after FTS report it, that is, some seconds after the completion of the transfer. This limits the information available to Rucio at a given time, as shown during discussion of models β_μ, $\beta\prime_\mu$, and γ_μ in Chapter 5, starting on page 40.

Along with the four timestamps already mentioned, the transfers dataset contains the sizes of the files requested to be transferred, and the links affected by the transfers.

The *rate* of a transfer is the amount of bytes successfully transferred in an interval of time[21]. This rate is determined by the network throughput of the endpoints, but also by the amount of simultaneous transfers happening at a given time. Rucio does not store information of the throughput of the links. Rucio neither store nor can infer the amount of simultaneous transfers happening at a given time. That is the most important consequence of FTS reporting the `started` and `ended` timestamps once a transfer ends. It is possible though, to calculate the amount of concurrent transfers and to approximate the rate of the link for historical transfers, based on the timestamps stored in Rucio. The average rate of a transfer can be approximated using Equation 7.1 on page 63, that is the size of the transfer divided by the Network Time.

$$xfer_{rate} = \frac{xfer_{size}}{xfer_{ended} - xfer_{started}} \tag{7.1}$$

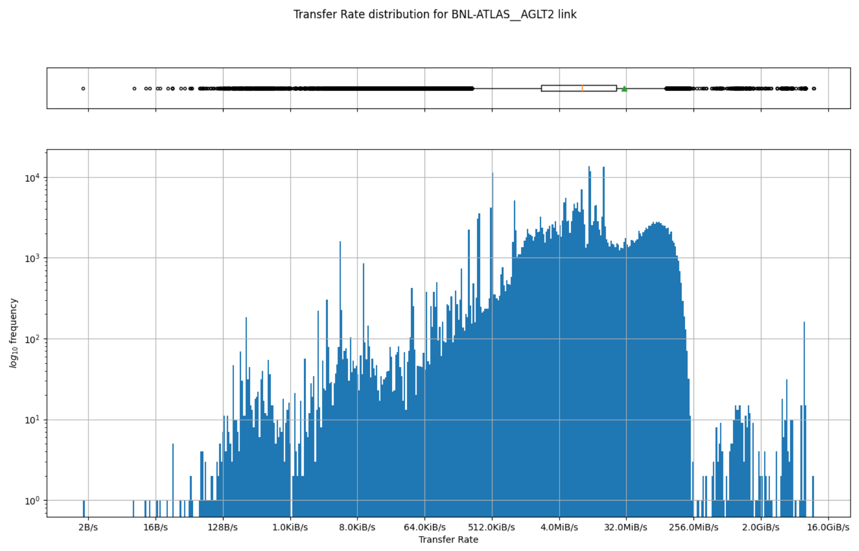


Figure 7.1: Distribution of the individual transfer rate as in Equation 7.1 on page 63 for the link `AGLT2__BNL-ATLAS`.

Figure 7.1 on page 64 shows the rate distribution of individual transfers of a representative link, calculated as in Equation 7.1 on page 63. The mean rate is 30.26 MiB/s. The rate distribution is not normal. It is important to notice that first, a particular transfer usually is sharing the link with other transfers. Moreover, this number is not constant during the Network Time of a transfer, as other transfer can be submitted or end during this period. Second, if a file is big enough, the rate will be I/O bound, as network bandwidth is usually higher than the read/write bandwidth of the storage. Third, if these files are read/written from different storage pools, each one averaging 80-100MiB/sec, then between 9 and 11 transfers should saturate a 10Gbps link. If that is the case mean rate should converge to a constant. As this does not happen according to Figure 7.1 on page 64, a possible explanations are that the link bandwidth is not being fully utilized, or that the link is being utilized by non-ATLAS transfers invisible to Rucio.

A relation between the size of the transfers and the rate was discovered during the research and is shown in Figure 7.3 on page 67. The figure shows the rate of a transfer as a function of its size, in logarithmic scale, for the transfer requests in the link `AGLT2__BNL-ATLAS`. Two features can be distinguished here. Distribution of transfer rate increases proportionally with file size up to 265MiB. Above that value starts to saturate. For transfers bigger than 256MiB the rate does not increase proportionally to the size of the transfer. Similar behaviour has been observed on most of the links.

If the rate is a function of the transfer size, then it is not linear for all the sizes, and for some file sizes the transfers are bound by the storage throughput. Also, the transfers suffer a delay product of the time it took to establish the connection between the source and destination site. This should be less than few seconds but could be not negligible, especially for short transfers.

If this is the case, it is possible to express the rate of a transfer as a function of its size as in Equation 7.2 on page 65. The mean rate of a transfer $rate_{xfer}$ can be approximated as the size of the transfer $size_{xfer}$ divided by the time the transfer needs to finish. This time can be expressed in seconds, as the size of the transfer divided by the *rate* that depends on the size plus an *overhead*. All these should be bound or be less than some limit at which the file can be read from or be written to the storage. Therefore, the *rate*, the *overhead*, and the *diskrw_limit* are unknowns. This equation has infinite solutions for a single transfer. However, for a set of transfer of a given link it is possible to approximate a solution using the ordinary least squares method. Moreover, some of the algorithms used to find the solution can diverge and find no solutions at all or find several solutions for the same set of data if initial conditions are randomized.

$$rate_{xfer} = \frac{size_{xfer}}{(\frac{size_{xfer}}{rate}) + overhead)} < diskrw\_limit \tag{7.2}$$

Notice that the least squares fit should be done using only transfers of the same link.

For all the transfers from BNL-ATLAS to AGLT2 or link `955b9..8222e4` in dataset [1], transfers were separated in training and testing datasets. The training dataset contains all the transfers created between June, 8th 2019 and July, 4th 2019. The testing dataset contains all the transfers created between July, 11th 2019 and July 29th 2019. The training dataset contains 121680 transfers and the testing dataset 76093 transfers. The results for 300 different solutions of *rate*, *overhead*, and *disk_limit* were found. Each solution was found by choosing at random 500 points in the size/rate plane from the training set, and by solving the least squares for Equation 7.2 on page 65 using the `least_squares` function from the `scipy.optimize` package. The number of random points selected was proven sufficient to allow the least squares algorithm to converge. Initial values for the unknowns were set to the mean rate of the training set for the *rate* variable, 1 for the *overhead* variable, and 100 to the *diskrw_limit* variable. These values are in the order of magnitude of the expected results. The least squares fit was constrained to

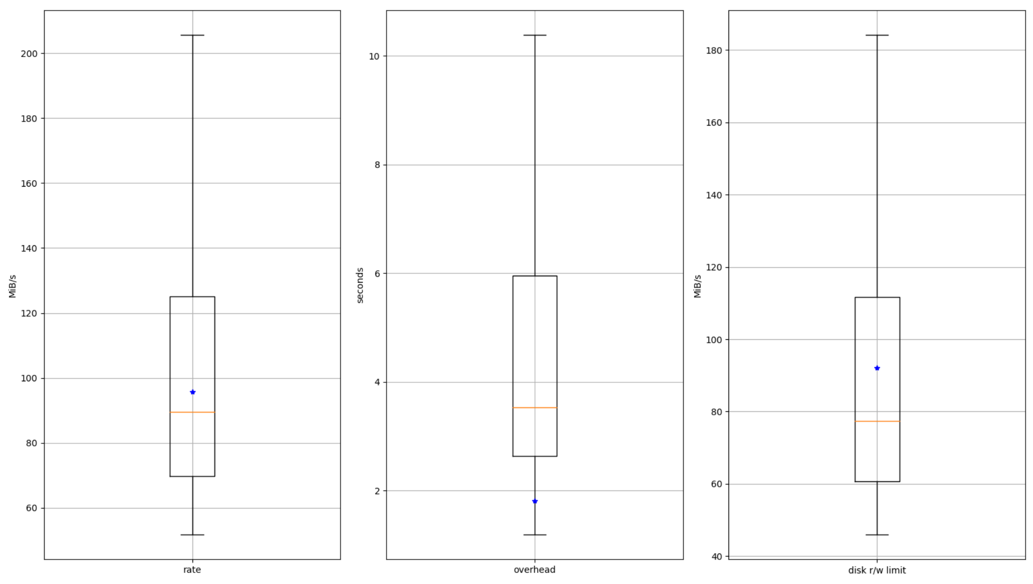


Figure 7.2: Distribution of the results for multiple fits of each variable. Ranges of the solutions are (51.79 MiB/s, 1670.88 MiB/s) for the rate, (1.19 s, 17.00 s) for the overhead, and (46.02 MiB/s, 299.14 MiB/s) for the storage read/write limit. Blue asterisk show the values for the fitted variables which predict best the training. These values are 95.72 MiB/s for the rate, 1.81 seconds for the overhead, and 92.04 MiB/s for the storage bandwidth limit.

allow only positive values. The distribution of the results for each variables is shown in Figure 7.2 on page 66.

In order to determine which set of results is able to make the best predictions of the Network Time based on the size of the transfer, the FoGP$(y, \hat{y}, \tau = 0.1)$ was calculated for each set of results. In this case, y is the observed Network Time for the transfers in the training set, and $\hat{y}$ is the predicted Network Time, obtained after computing $\frac{size_{xfer}}{rate_{xfer}}$, where the $rate_{xfer}$ is approximated using Equation 7.2 on page 65, replacing the variables for a particular set of results. Blue asterisks in Figure 7.2 on page 66 show the set of results that obtains the best FoGP, 95.72 MiB/s for *rate*, 1.81 seconds for *overhead*, and 92.04 MiB/s for *diskrw_limit*. Using these values, the FoGP$(y, \hat{y}, \tau = 0.1) = 0.3024$ means that 30.24% of the Network Time predictions have less than 10% relative error. If the same analysis is done using the observed rate instead of the Network Time, then the same set of results is returned as best but the FoGP metric is around 1% lower. This set of results were used to predict the Network Time for the transfers in the testing dataset, obtaining a FoGP$(y, \hat{y}, \tau = 0.1) = 0.2838$.

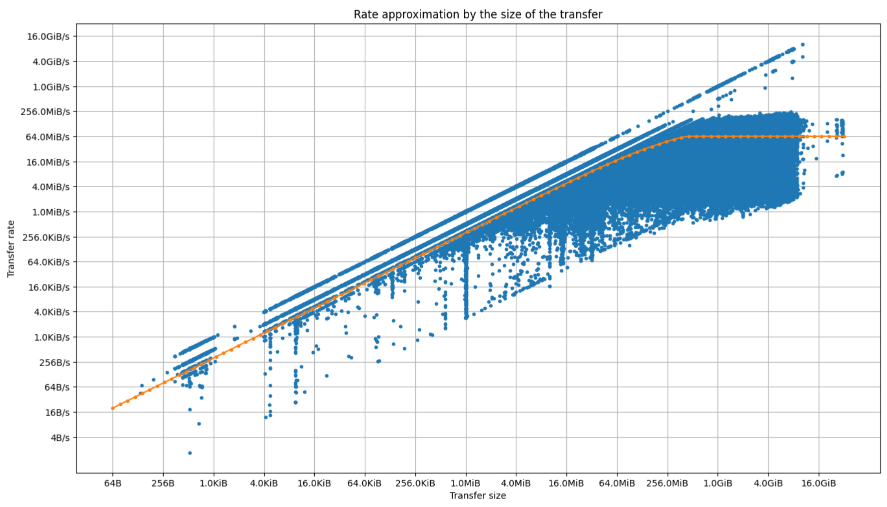


Figure 7.3: Transfer rate as a function of the transfer size. The blue dots represent individual transfers of the link `BNL-ATLAS__AGLT2`. The orange dotted line represents the best fit of the model of Equation 7.2 on page 65.

Figure 7.3 on page 67 shows the observed size and rate of each transfer in the test dataset. The orange dots are the approximated rate for a given size using the best set of results following the FoGP criteria to replace the values in Equation 7.2 on page 65.

The same experiment was repeated for the 10 links with a higher number of transfers. Table 7.1 on page 68 summarizes the results. The columns show the link, the results for the best solution according to the FoGP criteria, the highest FoGP obtained over the training dataset, and the FoGP obtained for the testing dataset. The split for training and testing set has been done by creation date of the transfer request, using the same dates interval, from 2019-06-08 to 2019-07-04 for the training dataset, and 2019-07-11 to 2019-07-29 for the testing dataset.

7.2 Network Time as a Transfer TTC and Rule TTC estimator

As the ultimate goal is to predict the Transfer TTC and Rule TTC, this section studies the possibility to use the Network Time of the transfer as a predictor. First, the observed Network Time was used to predict the Trans-

link	*rate*	*overhead*	*diskrw_limit*	FoGP (train)	FoGP (test)
55ada41..	16.11MiB/s	8.25 s	12.92MiB/s	0.6479	0.4986
69fda49..	4486.83MiB/s	2.27 s	1281.95MiB/s	0.0760	0.0064
ab50c34..	15.74MiB/s	7.60 s	11.89MiB/s	0.6970	0.3853
5d0adae..	579.06MiB/s	1.09 s	21.40MiB/s	0.3408	0.3278
7b65b5d..	59.91MiB/s	2.83 s	83.78MiB/s	0.1342	0.0472
9b06f74..	29.48MiB/s	1.83 s	15.78MiB/s	0.2515	0.0552
38043af..	12.55MiB/s	4.35 s	7.70MiB/s	0.2050	0.0989
0a88060..	196.29MiB/s	4.00 s	71.39MiB/s	0.1445	0.0241
c64d4d3..	53598.56MiB/s	1.01 s	4042.04MiB/s	0.8519	0.0000
6343052..	25.42MiB/s	2.78 s	24.05MiB/s	0.3767	0.3463

Table 7.1: Summary of the results for the 10 most popular links. Link hashes correspond to `CERN-PROD-->TRIUMF-LCG2`, `IFIC-LCG2-->SMU_HPC`, `BNL-ATLAS-->TRIUMF-LCG2`, `CERN-PROD-->IN2P3-CC`, `CERN-PROD-->INFN-T1`, `CERN-PROD-->NDGF-T1`, `BNL-ATLAS-->CERN-PROD`, `CERN-PROD-->CERN-PROD`, `UNI-BONN-->wuppertalprod`, and `CERN-PROD-->BNL-ATLAS`.

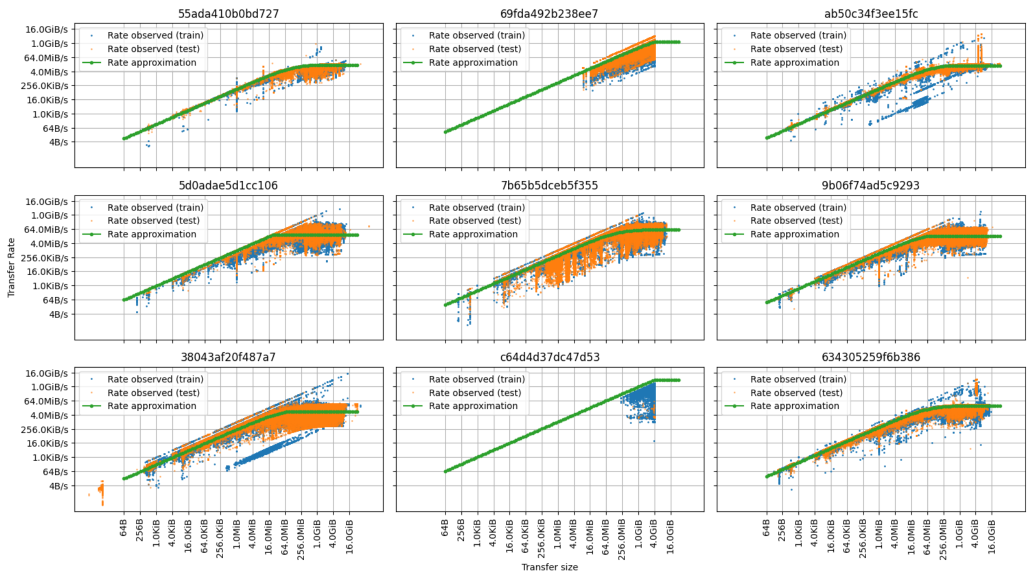


Figure 7.4: Transfer rate as a function of the transfer size for the transfers of the most popular links. The `CERN-PROD-->CERN-PROD (0a880..497131)` link was excluded.

fer TTC in order to study the properties of the estimator. For each transfer request for the 10 most popular links, the FoGP$(y, t_{NT}, \tau = 0.1)$ was computed, being y the Transfer TTC, the total time of the transfer request need to finished since its creation until it finishes, and t_{NT} the observed Network Time. The Transfer TTC can be obtained by subtracting the ending timestamp from the creation timestamp, but also can be obtained trough the sum of Rucio Time, FTS Queue Time, and Network Time. Table 7.2 on page 70 shows the results of predict the Transfer TTC using the Network Time in the column y, and the same prediction using the Rucio Time plus the Network Time, and FTS Queue Time plus the Network Time in the columns $y\prime$ and $y\prime\prime$ respectively. The accuracy of the prediction in column y is low in comparison with the other two columns. 7.4 show the transfer rate as a function of the size for the most popular links. The `UNI-BONN-->wuppertalprod (c64d4..c47d53)` link lacks statistics in test dataset which explains the 0.0 FoGP on Table 7.1 on page 68. Figure 7.5 on page 71 shows the distribution of these times against each other for the 9 more popular links. Here it is possible to see how the Network Time is small with respect to the Rucio time or the FTS queue time, and with respect to the Transfer TTC in general. This indicates that the Transfer TTC is dominated by the other components but the Network Time. The FTS Queue Times have a distribution where the median indicated with the orange line, is generally away from the mean value indicated with a green triangle, meaning the distributions are heavy tailed.

In the case of Rules TTC the same analysis was made. The Network Time approximation using polynomial fitting of Equation 7.2 on page 65 cannot be applied directly as not always all the transfers in a rule use the network of the same link. However, the observed Network Time can be computed for historical rules and this observed time can be used to predict the same rules in order to check if this is a good predictor for the Rule TTC. The Rule Network Time r_{NT} is defined as the difference between the maximum ending timestamp and the minimum starting timestamp of all those transfers with the same `rule_id`, that is all the transfer of the rule.

This definition of rule Network Time could not represent the actual time that the transfers of a rule are using the network. Imagine the case of a rule with two transfer requests. Both transfers are submitted to FTS at the same time but for some reason, the first starts but the second does not. If the first transfer ends before the second transfer starts, then the Network Time defined this way will overestimate the real Network Time. There will be gaps in time when the rule is not using the network, because transfers are queued in FTS that will be computed as Network Time.

Figure 7.6 on page 72 shows the FoGP measured for different τ thresholds,

link	y	$y\prime$	$y\prime\prime$
55ada410b0bd727	0.0151	0.0257	0.2751
69fda492b238ee7	0.0000	0.0000	0.0037
ab50c34f3ee15fc	0.0230	0.0393	0.1651
5d0adae5d1cc106	0.0078	0.0283	0.1401
7b65b5dceb5f355	0.0088	0.0199	0.0940
9b06f74ad5c9293	0.0074	0.0242	0.0806
38043af20f487a7	0.0468	0.0768	0.1497
0a880607b497131	0.0025	0.0135	0.0389
c64d4d37dc47d53	0.0000	0.0000	0.0003
634305259f6b386	0.0072	0.0131	0.2322

Table 7.2: Summary of the FoGP($y, t_{NT}, \tau = 0.1$), which is the percentage of predictions with less than 10% relative error, using the observed Network Time as predictor of y. The variables correspond to y the Transfer TTC, $y\prime$ the Rucio time plus the Network Time (`RTIME + NTIME`), and $y\prime\prime$ the FTS queue time plus the Network Time (`QTIME + NTIME`). Link hashes correspond to `CERN-PROD-->TRIUMF-LCG2`, `IFIC-LCG2-->SMU_HPC`, `BNL-ATLAS-->TRIUMF-LCG2`, `CERN-PROD-->IN2P3-CC`, `CERN-PROD-->INFN-T1`, `CERN-PROD-->NDGF-T1`, `BNL-ATLAS-->CERN-PROD`, `CERN-PROD-->CERN-PROD`, `UNI-BONN-->wuppertalprod`, and `CERN-PROD-->BNL-ATLAS` from top to bottom.

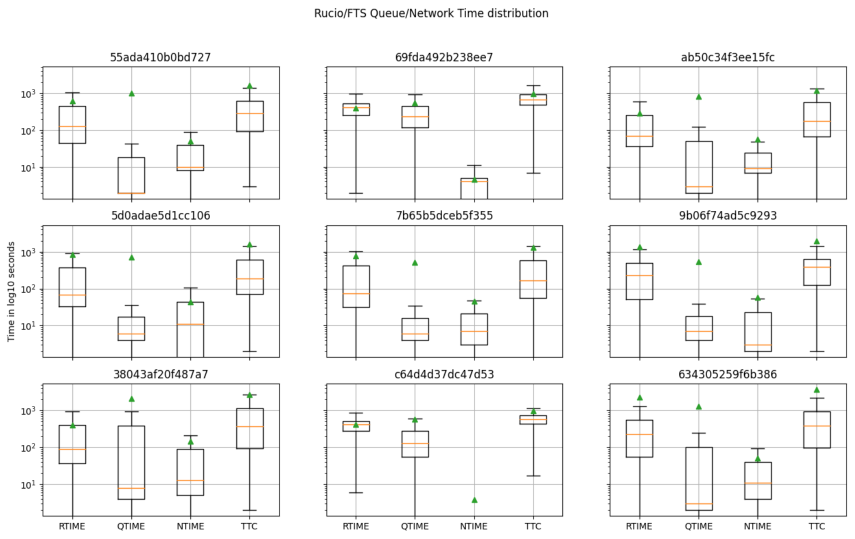


Figure 7.5: Rucio, FTS Queue Time, Network Time, and Transfer TTC distribution for the 9 more popular links. The `CERN-PROD-->CERN-PROD` link was excluded. The `UNI-BONN-->wuppertalprod` link presents an abnormal population of transfers with Network Time of 1 second for 69% of all the transfers sampled.

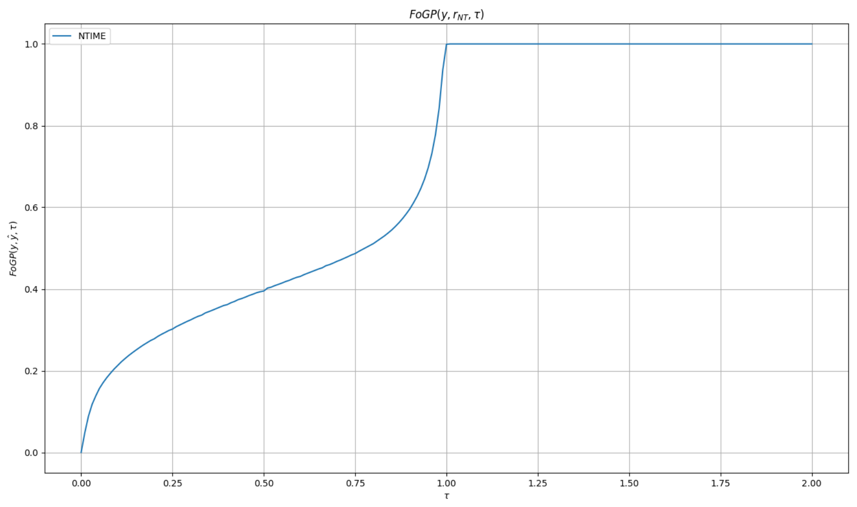


Figure 7.6: FoGP for several τ thresholds, when the r_{NT} estimator is used to predict the Rule TTC y, for all the rules in the dataset [1].

in which the observed Network Time of the rule is used to predict the Rule TTC for all the rules in the dataset. The results for $\tau = 0.1$ is 0.2128, meaning the 21.28% of the predictions have less than 10% relative error.

7.3 Results

The results for the best *rate*, *overhead* and *diskrw_limit* vary for every link. For some, the amount of overhead seems to be excessive. This could be due to lack of statistics in some size ranges or due to too broad distributions. If the network throughput is effectively constant, then the transfer rate will be dependent on the history of the active transfers during the Network Time. That is, the transfer will share the link, and thus the throughput, with several transfers that will be done in parallel. Moreover, the number of active transfers will not be constant, as during the Network Time of one transfer, some other transfers can start and finish. From the Rucio database it is possible to reconstruct the history of active transfers for finished transfers. Through the study of Rucio data it should be possible to check if the link network throughput converge to a value. But in order to apply any method to real time, FTS or other transfer tool should provide the timestamp the transfers request is submitted to the network at the moment of the submission, instead of at the ending transfer time in order to know the real number of active transfers in the network per link in real time.

On the other hand, the usage of the Network Time as predictor of Transfer or Rule TTC is limited, as demonstrated in Section 7.2 on page 67. Even if a model to predict the Network Time with 100% of accuracy could be used, then using this value will allow to predict the Rule TTC only around 21% of the time within 10% or less relative error.

Chapter 8

FTS Queue Time to predict Transfer TTC and Rule TTC

The whole is greater than the sum of the parts.

Aristotle

8.1 FTS queue modeling

The presentations cited in [22] and [4] address the problems in the prediction of the FTS Queue Time of the transfers, using the data available in Rucio database. The distribution of FTS Queue Time is not normal but heavy tailed as shown in Figure 8.1 on page 75.

Several issues arose during the study of the FTS Queue Time. There are several FTS server instances. From the data stored in Rucio it is possible to identify which transfer is served by which FTS server instance by the field `external_host`. However, while Rucio handle transfers request from ATLAS Virtual Organization (VO) exclusively, some FTS servers attend to transfers from ATLAS and other VOs. Transfers for other VOs will be opaque to Rucio and will not appear in the data collected from the Rucio database. If the number of transfers in the queue of FTS is being approximated from the number of transfers submissions by Rucio, the real number of requests queued could be severely underestimated.

In addition, the FTS scheduler algorithm in charge of selecting the transfer to be served next affects directly the queue time of those transfers already in FTS queue. The algorithm FTS uses to dispatch the transfers is as follows: The transfer requests submitted to FTS are assigned with a random unique

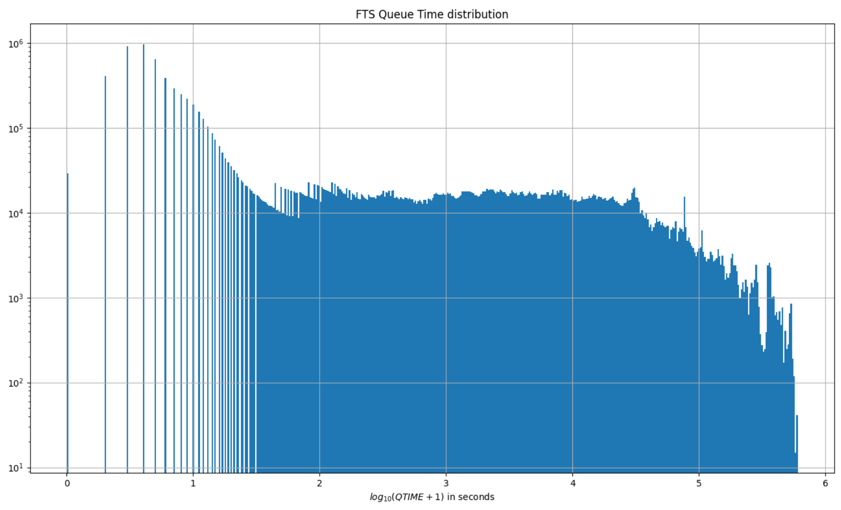


Figure 8.1: Distribution of the FTS Queue Times for the first 10 million data points in the transfers dataset. The minimum observed value is 0 seconds. The maximum value is 604747 seconds or 7 days. This threshold corresponds to an undocumented FTS configuration that boosts those requests that for some reason are still in the queue after 7 days.

identifier and then added to the FTS queue, a table in a centralized database per FTS server. This identifier is used to split the jobs between several FTS nodes that dispatch the requests in parallel. There are typically between 3 and 40 nodes per server. The nodes compete for access to the FTS database. The scheduler algorithm runs in a closed loop. For every run, the list of links with pending transfers is retrieved from the database through a SQL query. The list is randomly shuffled to avoid starvation. The first link is selected. From the transfers pending for that link a set of transfers are picked with certain probability that depends on the activity share of the transfer itself. The activity share is a number between 0 and 1 that guarantees that the transfers of all the activities make progress. The bigger the share, the more probable that the transfers of that activity will be processed. Table 8.1 on page 76 shows the different activity share values for the different activities defined in Rucio. The transfers are picked from FTS queue with a probability equal the the share of its activity, until the limit of active transfers in that link is reached. When all the transfers for the link were processed or the limit of actives transfers is reached, the next link is processed. Once all the links are processed, a new list of links with active transfers is retrieved

ACTIVITY	SHARE
Analysis_Input	0.15
Data_Brokering	0.3
Data_Consolidation	0.2
Data_Rebalancing	0.5
Express	0.4
Functional_Test	0.2
Production_Input	0.25
Production_Output	0.25
Recovery	0.4
Staging	0.5
T0_Export	0.7
User_Subscriptions	0.1

Table 8.1: Activity share per activity.

from the database and the cycle repeats.

As was mentioned earlier, the FTS scheduler pick transfers from the queue with a probability equal to the share of its activity. The higher the probability of the activity share, the most probable for a transfer to be selected to start. If the activity share for the activity A is 0.9 and for activity B is 0.5, and for activity C is 0.2, the scheduler will prefer transfers of the activity A over B and C. The link limit of active transfers per link is dynamic and managed by the FTS Optimizer algorithm. It depend among other things on the minimum and maximum number of active transfers configured set per source, per destination, per link, the throughput of the link, and the success rate of previous transfers. Details of the algorithm and its implementation can be found in [23].

8.2 Modeling the FTS queue from Rucio data

Despite the complexity of the FTS scheduler, it is possible to approximate the number of transfers in the FTS queue and thus the FTS Queue Time using the data stored in the Rucio database. Uncertainties about the real number of queued requests will be related to the number of active transfers. Active transfers are the ones FTS already dispatched at `started` timestamp. This timestamp is not communicated to Rucio in real time, but when the transfer ends, along with the `ended` timestamp. Therefore, considering all the transfers that have been submitted from Rucio to FTS as queued transfers in FTS is an overestimation. Some of these transfers may already have been started, but the `started` timestamp may not be in the Rucio database yet.

The proposed method consists in simulating the process after the transfers are submitted to FTS. A simulator was created to emulate the characteristics of FTS. The code of the simulator is available in the git repository of the thesis[24].

The simulator works as follows. The FTS state dictionary, a structure that represents and keep track of the internal state of the FTS server, is set to zero and the clock of the simulator is set to the minimum submitted timestamp of the transfers of the sample. A transfer streamer object that returns all the submitted transfers for the next simulator clock tick is created. The simulation resolution is 1 second. The simulation process loop begins. The following items summarize the steps in the loop.

1. Get events from streamer object.
2. Simulate transfers submission to FTS.
3. Simulate transfers activation in FTS.
4. Simulate transfers progress.
5. Save FTS state structure.
6. Increment the simulation clock by one second.
7. Go to 1

The streamer object returns the list of transfers that have been submitted at simulation clock time. This list is obtained by filtering the dataset to get all the transfers that have been submitted in a given second, according to the Rucio database.

The submission to FTS is simulated. The internal state of FTS queues are updated to reflect that new transfers have been submitted and there are links with pending transfers in the queue. The simulated submission time is the observed submission time of the transfer in the Rucio database. The Rucio Queue Time is not simulated.

The activation of transfers follows the same algorithm the FTS scheduler uses to select what transfers are dispatched next. The list of links with pending transfers is randomised. The transfers for a link are activated in FIFO order until the link limit of active transfers is reached. The activity shares are not simulated. After the first link is processed, the following link is selected and its transfers are activated. If all the transfers of a link are activated, the link is removed of the list of links with pending transfers. Once a transfer is activated, it exits the FTS queue and the transfer starts its Network Time. The simulated activation time can be different from the observed started timestamp of the transfer.

The simulation of the transfers progress is determined by the network throughput of the link. As this value is not known the observed rate of the transfer is used. When a transfer is activated the remaining bytes to be transferred is set to the size of the transfer. At each simulation tick, for all the active transfers, the observed transfer rate is subtracted from the remaining bytes to be transferred. When the remaining bytes to be transferred is zero, the transfer ends.

In order to save the FTS state, all the values of the dictionary structure are converted to integers and the dictionary is flattened to a Numpy Array.

Finally, the clock is increased by one second and the loop starts again. The simulation finishes when all the simulated transfers are processed.

This simulation model is know as Model B in [22] and is the best known model to simulate FTS queues. However, the rate of the transfers must be known in order to simulate the transfers Network Time progress. Other models in [22] address the problem to simulate the transfer progress when the rate is not known, with significantly less success. This approach is easier to implement over one in which mock FTS server, network and storage endpoints are used.

Failed transfers are also not simulated. Failed transfers can reduce dramatically the dynamic limit of active transfers for a given link after a failure. Future studies should investigate further the influence of failed transfers.

8.3 Using FTS Queue Time as a Transfer TTC and a Rule TTC predictor

FTS Queue Time can be used as a predictor of the Transfer TTC and Rule TTC. This value cannot be known at request creation time or rule creation time. In this section we discuss the use of FTS Queue Time as a predictor for Transfer TTC and Rule TTC, assuming that the FTS Queue Time can be known or be predicted with 100% accuracy.

For transfers, the FTS Queue Time can be computed directly from the dataset, subtracting the `started` timestamp to the `submitted` timestamp. When the observed FTS Queue Time of transfer (t_{QT}) is used to predict the Transfer TTC (t_{TTC}), the FoGP($t_{TTC}, t_{QT}, \tau = 0.1$) = 0.0927.

For rules, the FTS Queue Time can be defined in 3 different ways. The

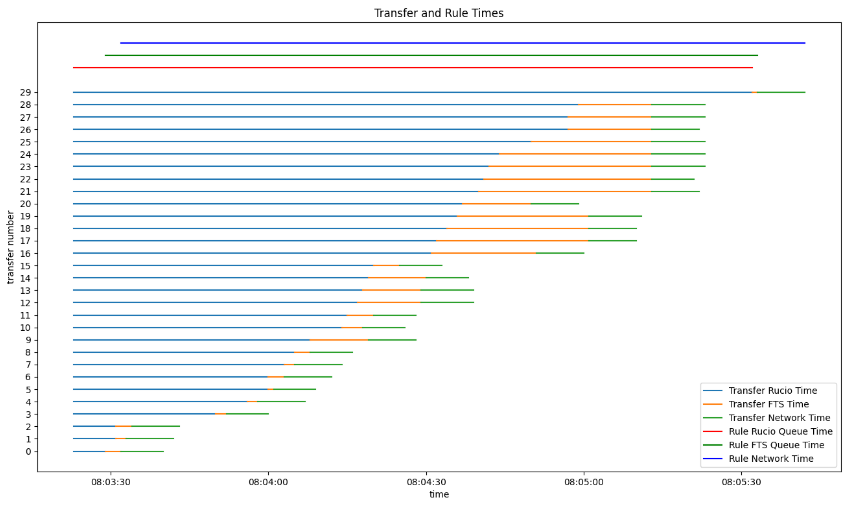


Figure 8.2: Times in life cycle for all transfers in one particular rule. First three colour lines top to bottom correspond to total rule Network Time (blue), FTS Queue Time (green) and Rucio Time (red). For each of the 30 transfers in this rule, Rucio Time is shown in blue, FTS Queue Time is shown in orange and Network Time is shown in green.

FTS Queue Time of the rule r_{QT} can be defined as the interval between the minimum submission time and the maximum started time of all the transfers with the same `rule_id`. If this is the case, then same as in rule Network Time, the FTS Queue Time can be overestimated. Figure 8.2 on page 79 shows the different times in the life cycle of a transfer, for all the transfers of a particular rule. Transfers are numbered from 0 to 31 in the order they appear in the dataset. All the transfers were created at the same time at 08:03:23 on July 3rd 2019. Rucio presents no idle time, as all the transfer are created at the same time. Total FTS Queue Time according to Equation 8.1 on page 80 is 123 seconds. In total, during 42 seconds there were transfers ready in Rucio but not submitted to FTS (FTS idle time). The effective FTS Queue Time according to Equation 8.3 on page 80 is 81 seconds.

A second definition of the rule FTS Queue Time, $r\prime_{QT}$, is to calculate a function of the FTS Queue Time of the individual transfers of the rule. The third definition, $r\prime\prime_{QT}$ calculate the effective time that transfers of the rule spend in the FTS queue. From all the Rule TTC, the slices of time where no transfers are in the FTS queue are discounted from the r_{QT}. Thus, $r\prime\prime_{QT}$ should represent the real load of the rule in FTS system. Formally, the first

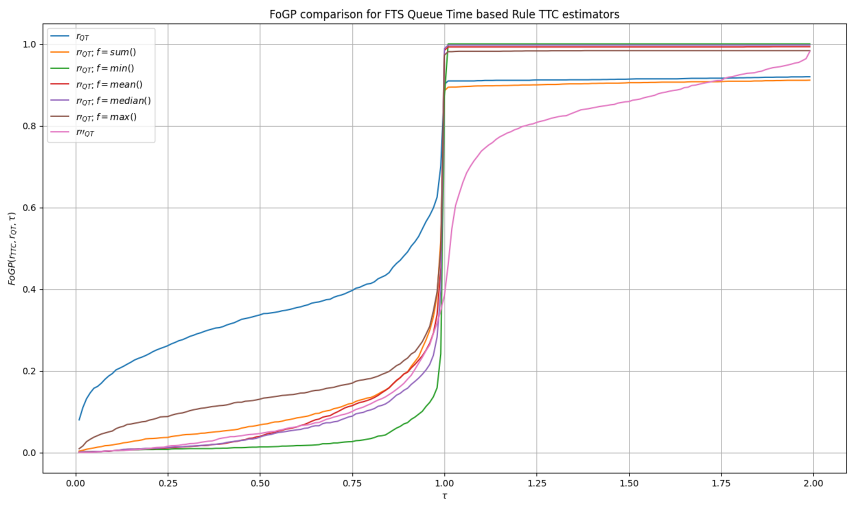


Figure 8.3: FoGP for different τ thresholds to compare the accuracy of the proposed predictors for Rule TTC based on the FTS Queue Time estimators. r_{QT} shows the best performance. Next best is $r\prime_{QT}$ with $f = max()$, that is, the model that predicts the Rule TTC using the FTS Queue Time of the transfer that will spend the most time in the queue.

definitions is as in Equation 8.1 on page 80, where $t_{started}$ and $t_{submitted}$ are the respective timestamps of a transfer t that belongs to the rule r. The second definition is as in Equation 8.2 on page 80, where f is a function of the FTS Queue Time of the transfers of the rule, i.e., the mean of the FTS Queue Time of the transfers of the rule will approximate the FTS Queue Time of the rule. The third definition is as in Equation 8.3 on page 80, where q_{IDLE} is is the number of seconds between $min(t_{submitted})$ and $max(t_{started})$ of which there is no transfers of the rule in FTS queue. Figure 8.3 on page 80 shows the performance of this predictors.

$$r_{QT} = max(t_{started}) - min(t_{submitted}) \tag{8.1}$$

$$r\prime_{QT} = f(t_{QT}) \tag{8.2}$$

$$r\prime\prime_{QT} = r_{QT} - q_{IDLE} \tag{8.3}$$

The different definitions of the rule FTS Queue Time were used as predictors for the Rule TTC and the results were compared using the FoGP metric. When using predictor $r\prime$ only the $min()$, $mean()$, $median()$, and $max()$ functions were considered.

This results indicate that the FTS Queue Time is a poor predictor for both Transfer TTC and Rule TTC. Individual transfers FTS Queue Time is, in general, overlapping with the Rucio Time, FTS Queue Time, and Network Time of other transfers in the rule, making it more difficult to find a function f that makes Equation 8.2 on page 80 useful to predict the FTS Queue Time of the rule. Even if a model to predict the FTS Queue Time, of both transfers and rules, with 100% accuracy is found, its utility to predict the Rule TTC is doubtful. The results obtained with the r_{QT} estimator are the best ones, but the estimator is consistently overestimating the real FTS Queue Time. These results suggest that the FTS Queue Time of the rules have very little impact on the Rule TTC and in the Transfer TTC. This supports the evidence provided by the results summarized in Figure 7.5 on page 71, in which is clear that the contribution of FTS Queue Time to the total Rule TTC is low in average and when compared with the Rucio Queue Time and in some cases with the Network Time.

Chapter 9

Results and conclusion

Real knowledge is to know the extent of one's ignorance.

Confucius

9.1 Models summary

We have presented a series of models to predict the Rule TTC at rule creation time and a metric to compare them. The summary of the performance obtained by the models according to the FoGP metric at $\tau = 0.1$ is shown in Table 9.1 on page 82. A summary of the models and their most important characteristics are presented in the following sections, among with a general discussion of the models, and the possible extensions of this work.

Model	FoGP$(y, \hat{y}, \tau = 0.1)$
κ	0.1323
β_{median}	0.2066
β^{*}_{max}	0.1547
γ_{median}	0.0502
δ_{32}	0.1098
$\delta\nu\nu_{32}$	0.1409

Table 9.1: Summary of FoGP$(y, \hat{y}, \tau = 0.1)$ for the models presented in this work.

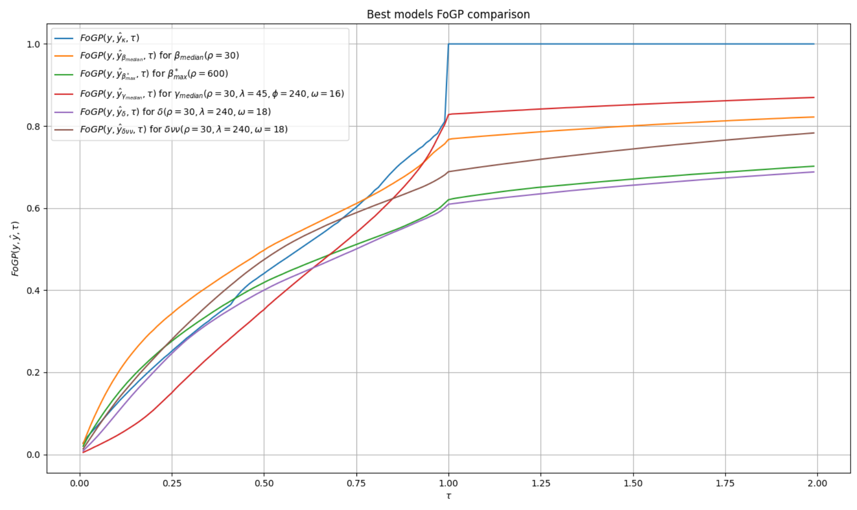


Figure 9.1: FoGP comparison over a τ range from 0.01 to 2 for the best models known, which were presented in this work. Predictions for all the models where made for all the rules created between 2019-07-11 and 2019-07-29.

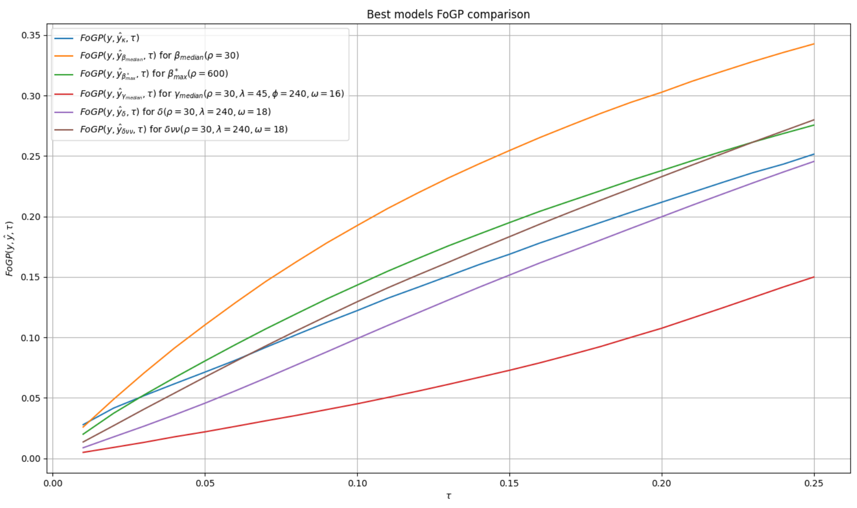


Figure 9.2: FoGP as in Figure 9.1 on page 83 but zoomed over a τ range from 0.01 to 0.25 for the best models.

9.2 Model κ

The κ model, which always predicts a constant value, allows us to put a lower bound for the performance of the models over a range of interesting τ values. Optimizing the constant in order to maximize the FoGP results in a model that is surprisingly difficult to improve upon, both at high and low τ values. By its simplicity, and because its performance is comparable to other more sophisticated models, it should be the preferred to be implemented, for example, in order to give feedback to users about the TTC of their transfers. If that is the case, the upper bound of a confidence interval would be interesting for users.

9.3 Model α

Model α is the only model of the studied ones that is not directly comparable with the other models due to inability to make predictions at the Rule creation time. Model α needs at least two transfers within the rule to finish in order to fit and forecast when the other transfers probably will finish. This makes the model suitable to give feedback to the users. But this model will not be helpful to improve the scheduler as the decision about where to send the transfers will need to be done at rule creation time and before any transfer is submitted but not finished. The model shows the non-linearity of the progression of the transfers, giving insights of the nature of the rules and their behavior. The time between transfer submissions for the transfers of a rule is not constant. Rucio may consider that FTS has a high enough number of transfers already and decide not to submit more transfers until some of those active transfers finish, increasing the Rucio Queue Time for part of the transfers of the rule. This will impact directly in the Rule TTC and this model will not be able to forecast these future delays.

9.4 Models $\beta_\mu(t_0, \rho)$ and $\beta^*_\mu(t_0, \rho)$

Models $\beta_\mu(t_0, \rho)$ and $\beta^*_\mu(t_0, \rho)$ make a prediction calculating a function μ over the Rule TTC of those rules created in the last ρ seconds. The difference between β_μ and β^*_μ is that β^*_μ excludes those rules that end after t_0. The β_μ model cannot be implemented with real time data as it calculates the μ function over the Rule TTC of all the rules that have started at some point in the past, including the ones that have not finished yet. This information from the future added to the model makes the two models radically different. One could assume that if the μ function could be predicted with 100% accuracy, then FoGP of the model β_μ represents the theoretical limit for of FoGP of the model β^*_μ, as the first include more information than the second. Yet, this statement does not hold in general, for example, for the function that takes the maximum, including more information in the model does not make it more accurate. The β_{max} model makes a prediction by calculating the maximum Rule TTC of all the transfers created between t_0 and ρ. The bigger the ρ is the bigger is the chance that there exist a very slow rule. But β^*_{max} filters out those transfers that have finished after t_0, and thus the Rule TTC is throttled to the value of ρ. For this reason, the $\text{FoGP}(y, \hat{y}_{\beta^*_{max}}, \tau = 0.1)$ presents a peak when ρ is near 600. This value is close to the best value for the model κ at $\tau = 0.1$, which is 562. β^*_μ models with other parameters presents lower FoGP values than β^*_{max} at $\tau = 0.1$, and thus are considered inferior models.

Figure 9.1 and Figure 9.2 on page 83 shows that $\beta^*_{max}(\rho = 600)$ outperforms model κ in the τ range between 0.04 and 0.22. This is the best model known to date in that range. It is not possible to implement the β_{median} model without known the Rule TTC of rules that didn't finish yet. If a model for a perfect prediction of the median of the Rule TTC exists, then the $\beta_{median}(\rho = 30)$ shows the best performance across a wide range of τ.

9.5 Model $\gamma_\mu(t_0, \rho, \lambda, \psi, \omega)$

The γ_μ model family is the first approach to solve the problem using time series analysis. The γ_μ is an auto-regressive (AR) model where the input is the time series of the Rule TTC. The function μ is calculated in bins of ρ seconds. The input for the AR model consists of λ lags. The model is fitted using ψ lags and the look ahead of the model is ω lags. The best model was obtained by scanning the parameter space and maximizing the FoGP, as detailed in Section 5.3 on page 45. Model $\gamma_{median}(t_0, \rho = 30, \lambda = 45, \psi 240, \omega = 16)$ achieved the best FoGP at $\tau = 0.1$. If this model would predict the median with 100% accuracy then its results should be comparable with those obtained with the β_{median}. The results show that the γ_μ model is not as good, especially at low τ. The γ_{median} model is better than β_{median} only for $\tau > 0.91$. This model seems to be not accurate enough and other more complex models are worth to try, as discussed on Section 10.2 on page 91. Integrated models were discarded after verifying that the time series show no trend, ergo there is no need for differentiation. Moving Average models are used after verifying that the time series are not stationary, which is not the case for long runs of Rule TTC time series. A straightforward check showed that the standard deviation from the mean changes over time, and thus a General Auto-Regressive Conditional Heterokedasticity (GARCH) model is more appropriate.

9.6 Models δ and $\delta\nu\nu$

The δ model is the first attempt to solve the forecast problem with neural networks using a modified model proposed by F. Chollet. This approach was shown to be ineffective but its accuracy is higher than the accuracy of model γ_{median}. Model δ includes the information of the past state of the system in the form of time series but it does not include information of the present. Information from the rule that is know at the creation time like the number of transfers of the sum of bytes the system must process in order to complete the rule are not included in model δ. This observation leads to the $\delta\nu\nu$ model, a deep neural network model with multiple inputs that includes the time series from δ model, but also the number of bytes, number of transfers, and the links affected by the rule. $\delta\nu\nu$ model is the best practical model in the τ range from 0.22 to 0.77, but more important, it is the easiest model to extend. We expect that this model would benefit enormously if information about failed transfers per link, history of transfers submitted to FTS, and history of the rate of the link were available and could be added as inputs.

9.7 Models based on individual transfers

The prediction of the Rule TTC based on the Transfer TTC presents several problems. First, the random process involved in the shuffling of the links to be served in FTS introduces noise in the FTS Queue Time of the transfers. Second, the FTS Optimizer reduces the number of transfers submitted to a link in ways that depend on the configuration of the link. This information however is opaque to Rucio, but could be inferred from the number of actives per link and the number of failed transfers, and ultimately, from FTS server configuration. But third and more important, even if the Transfer TTC is known with 100% certainty, due to the overlaps between the Rucio Queue Time, FTS Queue Time and Network Time of the individual transfers, it is still not clear how this information could be embedded in a model to predict the Rule TTC.

The model to simulate the number of queued transfers in FTS works very well in several scenarios, but discrepancies between the model prediction and observed values need to be addressed. Any model to should include the number of failed transfers per link in order to simulate the optimizer algorithm correctly, and again, as some of this configurations are link based and Rucio does not know about them, additional sources of data must be included.

9.8 Conclusion and final remarks

The distributed data management for the experiments using the Worldwide LHC Computing Grid forms a complex ecosystem with dynamic interactions. Since its commissioning in 2014, Rucio has become the de-facto standard for scientific data management, even outside the CERN community.

> "For many scientific projects, data management is an increasingly complicated challenge. The number of data-intensive instruments generating unprecedented volumes of data is growing and their accompanying workflows are becoming more complex. Their storage and computing resources are heterogeneous and are distributed at numerous geographical locations belonging to different administrative domains and organisations. [...] But ATLAS is not alone, and several diverse scientific projects have started evaluating, adopting, and adapting the Rucio system for their own needs. As the Rucio community has grown, many improvements have been introduced, customisations have been added, and many bugs have been fixed. Additionally, new dataflows have been investigated and operational experiences have been documented. In this article we collect and compare the common successes, pitfalls, and oddities that arose in the evaluation efforts of multiple diverse experiments, and compare them with the ATLAS experience. This includes the high-energy physics experiments Belle II and CMS, the neutrino experiment DUNE, the scattering radar experiment EISCAT3D, the gravitational wave observatories LIGO and VIRGO, the SKA radio telescope, and the dark matter search experiment XENON".[25]

The accuracy of the models to forecast its behavior will be bound for the amount of data about the system available at a given moment and for the stochastic processes involved in certain parts of the system. Rucio's importance and the rich amount of data gathered about the transfers and rules life cycles, make its study paramount.

From Rucio's point of view, the data about the transfers at any given moment is rich, allowing the reconstruction of the state of the DDM system in great detail. However, some aspects of the system remain hidden in these data. First, at the moment of taking the sample of the data during the months of June and July 2019, the Rucio instance was ATLAS VO specific, whereas FTS accepted transfers requests from other VOs as well. How many non-ATLAS transfers have been submitted to a specific FTS server in a given moment remain hidden to Rucio. Second, the FTS scheduler introduces randomness through the link shuffling and job identifier designation, transforming the output of the scheduling task into a stochastic process. Third, Rucio can identify which FTS server will serve each transfer, but not how many nodes, or even which node, in an FTS server will attend to the transfer request. This is opaque to Rucio. Fourth, Rucio can identify if the source of the transfer is a tape endpoint. These transfers represent a small but significant amount of the total requests. When a transfer source is tape, the tape system recalls the file to a disk buffer before transfer is made to the destination. This is known as Staging Time and can represent an important fraction of the total transfer time. Rucio in versions before 1.21.6 is not able to differentiate between FTS Queue Time and Staging Time, accounting as FTS Queue Time the sum of both. After version 1.21.6, the `staging_start` and `staging_finished` timestamps were included to the Rucio's REQUEST table allowing to reconstruct staging times for tape transfers. Fifth, Rucio's Conveyor daemon manages the number of transfers that are submitted to FTS or queued in Rucio at a given time. This daemon is configured dynamically to avoid overwhelming the FTS servers. These thresholds are set by the DDM Operations team, based on several sources that the team uses to monitor the DDM system. These settings can impact the Rucio Queue Time of the transfers. Given the contribution of the Rucio Queue Time to the Transfer TTC and Rule TTC, further study of the Rucio Queue Time is needed.

Further analysis will be required in order to determine how this factors affect the Transfer TTC and Rule TTC and the accuracy of future models.

Several models were presented and evaluated during this work, especially for Rule TTC prediction. All presented models except model α can make predictions at the rule creation time. Model $\beta_{median}(\rho = 30)$ outperforms all the models. However, the real median of previous Rule TTC needs to be known for the model to work and this information is not available at the time to make a prediction. The β^{*}_{max} is the best model following the FoGP criteria in the intervals of τ between 0.03 and 0.22. Model $\delta\nu\nu$ is the best in the intervals of τ between 0.22 and 0.77 and is the model with greatest potential to be extended. The performance of all the models are comparable with the performance of Model κ, and for its simplicity, should be preferred. The accuracy of these models does not allow them to be used to improve the schedule of the transfers or rules. The expected threshold that would make these predictions useful is a FoGP$(y, \hat{y}, \tau = 0.1)$ of around 0.95. Even if the accuracy of the models presented here is not enough for scheduling purposes, excluding model α, these are the best models known to date for Rule TTC prediction at rule creation time. On the other hand, in order to give feedback to the users about how much time a particular set of transfers will take, these models could only provide an educated guess.

This work lays the foundation for future models and establishes the metric by which they should be compared.

Chapter 10

Future work

What is coming is better
than what is gone.

Arabic proverb

10.1 Possible extensions to the $\delta\nu\nu$ model

The $\delta\nu\nu$ model is promising. The following section explain the best way to extend the model, ordered by difficulty with respect to data pre-processing and coding, from easiest to hardest.

The number of failed transfers per link, aggregated over a the last 10 to 30 minutes or some recent history in time series form could improve the performance of the model. The FTS Optimizer penalizes the links if there are failed transfers. Those links without failures in the recent history are preferred over the ones with failures. Thus, the transfers pending over affecting the links with failures will be delayed more than the transfers that will use a link without failures. The easiest way to add this information to the model is aggregated as an array with one value per link in the embedding branch. Another way, but computationally more expensive, is to add another time series to the time series branch, but this time series needs to be computed at link bases and it is not clear how the information needs to be summarized for rules that affect multiple links. The number of failed transfers per link can be calculated from the transfers dataset, searching for those transfers with `state == 'F'`.

The $\delta\nu\nu$ model could also benefit from knowing how many transfers have been submitted to the links that the target rule will affect. The easiest way to include this information is also summarized. The proper way to calculate this from the transfers dataset is to get all the transfers that have been submitted, but did not finish yet, ignoring the started timestamp, due to Rucio only knowing when a particular transfer has been started after it ends. At that moment, the FTS server will make available both timestamps, started and ended.

The number of transfers that have been submitted to the FTS server and did not finish yet can also be included aggregated as a time series in the time series branch, or summarized in the embedding branch. However, early studies found no evidence of linear correlation between the number of active transfers and the Rule TTC.

10.2 More complex auto-regressive models

General Auto-Regressive Conditional Heterokedasticity (GARCH) models have been used in the past in time series where the variance is not constant over time. In [26], the authors introduce a representation space called Complexity-Entropy Causality plane, based on the Bandt-Pompe[27] probability patterns for time series. Preliminary studies of the time series used in this work suggest these series are compatible with white noise, but this results were not published nor the results should be considered conclusive. In [28], the authors introduce a method to determine the predictability of a time series. This methods should be applied to the time series studied in this work before more complex time series analysis models are implemented.

Thanks

Mario Lassnig für alles. Diese Arbeit wäre ohne seine Hilfe nicht abgeschlossen worden, was immer weit über seine Verantwortung als Direktor hinausging. Danke.

A Fernando Monticelli, por todo. Este trabajo no se hubiera terminado sin su ayuda, la cual siempre fue mucho más allá de sus responsabilidades como director. Gracias.

A Javier Díaz, por dejarme elegir el camino y por bancarme siempre. Gracias.

A Teresa Dova, por abrirme las puertas de ATLAS, con todo lo que eso implica. Gracias.

An Armin Nairz, für seine Hilfe bei der Finanzierung meiner Aufenthalte am CERN und dafür, dass ich mich wie zu Hause gefühlt habe, als ich weiter weg von zu Hause war als je zuvor. Danke.

An Karl Jakobs, für seine Hilfe bei der Finanzierung von Reisen und Aufenthalten am CERN. Vielen Dank.

Ad Alessandro di Girolamo, per la sua pazienza, e per la sua generosità nel condividere con me il suo tempo e la sua conoscenza dell'esperimento. Grazie mille.

An Thomas Beermann, für seine Hilfe bei der Extraktion und Vorverarbeitung des in dieser Arbeit untersuchten Datensatzes. Danke.

An Martin Barisits, für die unzähligen technischen Konsultationen zu Rucio, für die er Zeit zum Diskutieren fand. Danke.

A Cédric Serfon, pour son aide dans la sélection de la fenêtre temporelle de l'ensemble de données étudié dans ce travail. Merci.

A Vincent Garonne, pour son aide et son accompagnement lors des premières étapes de ce travail. Merci.

A Jona y Aimé, por facilitarme el acceso al hardware. Gracias.

A Elio y Mimi, Mauro, Eliana, Gino y Pier y a los Reicher, por recibirme en las navidades durante los últimos 6 años y por haberme hecho sentir como en casa cuando estaba más lejos de casa que nunca.

A mis viejos Luis y Beatriz, por llevarme y traerme a Ezeiza las veces que pudieron, y a Joaco y Monti por haber ido las otras veces. Gracias.

A Lau, Fede y muy especialmente a Pau. Por las innumerables veces que nos juntamos a estudiar, a darnos ánimos y a avanzar con la carrera. Gracias.

A la gente de la oficina de postgrado, especialmente a Alejandra Pizarro, por estar 24/7 para todo. Gracias.

To the ATLAS Secretariat, especially to Nathalie Schwarzbauer, for her help with the arrival at CERN and E-Planet Grant paperwork. Thank you.

A Adjani y a Arthur, por su ayuda con las traducciones. Danke. Merci.

Thank you. Danke. Merci. Grazie. Gracias.

Acronyms

AGIS	ATLAS Grid Information Service.
BNL	Brookhaven National Laboratory.
FoGP	Fraction of Good Predictions.
FTS, FTS3	File Transfer System (version 3).
LHC	Large Hadron Collider.
LSTM	Long Short Term Memory.
RSE	Rucio Storage Element.
TSA	Time Series Analysis.
TTC	Time To Complete.
VO	Virtual Organization.
WLCG	Worldwide LHC Computing Grid.

Glossary

Fraction of Good Predictions Metric that, when applied to a set of predictions made for a model, reflects the fraction of those that present less than a certain relative error τ. This metric is defined and discussed in detail in Section 3.2.7 on page 27.

GRU Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks. The GRU is like a long short-term memory (LSTM) with a forget gate,[2] but has fewer parameters than LSTM, as it lacks an output gate..

Link The pair of source and destination RSEs for a transfer or rule.

Link Homogeneous When all the transfers of the rule go through the same link.

LSTM Long short-term memory (LSTM) is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feed-forward neural networks, LSTM has feedback connections. This kind of artificial neurons present some benefits over the Gate Recurrent Unit (GRU) described in [19] but take considerably more time to train.

Rucio Replication Rule Rucio abstraction over a set of data identifiers that allows the data management system to enforce a replication policy. Every time a rule is created, the system create transfer requests to fulfill the replication rule.

Rule Time To Complete Rule Time To Complete for a rule is defined as the difference between the maximum `ended_at` and minimum `created _at` timestamps of all the transfers with the `rule_id` equal to `ruleid`.

Throughput The rate at which something is processed. Usually in bytes.

Virtual Ortanization In WLGC context, a Virtual Organization define a logical group of activities and sites that work for a particular experiment. This allows make partitioning of resources from bandwidth to storage to job slot allocation among the experiments. All ATLAS experiment activities are part of the in `atlas` VO.

Bibliography

[1] J. Bogado, M. Lassnig, F. Monticelli, J. Díaz, and T. Beermann, "Atlas rucio transfers dataset," *Zenodo*, Dec. 2020. DOI: https://doi.org/10.5281/zenodo.4320937.

[2] M. Lassnig, W. Toler, R. Vamosi, and J. Bogado, "Machine learning of network metrics in atlas distributed data management," *Journal of Physics: Conference Series*, vol. 898, p. 062009, 10 2017.

[3] V. Begy, M. Barisits, M. Lassnig, and E. Schikuta, "Forecasting network throughput of remote data access in computing grids," *Journal of Computational Science*, vol. 44, p. 101158, 2020.

[4] J. Bogado, F. Monticelli, J. Diaz, M. Lassnig, and I. Vukotic, "Modelling high-energy physics data transfers," in *2018 IEEE 14th International Conference on e-Science (e-Science)*, pp. 334–335, 2018.

[5] CERN, *Worldwide LHC Computing Grid Public Site*, (accessed August 4, 2020). https://wlcg-public.web.cern.ch.

[6] A. Kiryanov, A. Álvarez Ayllón, M. Salichos, and O. Keeble, "Fts3 - a file transfer service for grids, hpcs and clouds," in *International Symposium on Grids and Clouds 2015*, p. 028, 03 2016.

[7] CERN, *File Transfer Service*, (accessed August 4, 2020). http://fts.web.cern.ch/fts/.

[8] M. B. on behalf of the Rucio Team, *Welcome, Introduction and Rucio Status (presentation)*, (accessed August 5, 2020). Slides available in https://indico.cern.ch/event/773489/contributions/3275021/attachments/1803610/2942599/Welcome_Introduction__Rucio_status.pdf.

[9] M. Barisits, C. Serfon, V. Garonne, M. Lassnig, G. Stewart, T. Beermann, R. Vigne, L. Goossens, A. Nairz, and A. M. and, "ATLAS replica management in rucio: Replication rules and subscriptions," *Journal of Physics: Conference Series*, vol. 513, p. 042003, jun 2014.

[10] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing, V. Garonne, A. di Girolamo, L. Goossens, W. Guan, J. Guenther, T. Javurek, D. Kuhn, M. Lassnig, F. Lopez, N. Magini, A. Molfetas, A. Nairz, F. Ould-Saada, S. Prenner, C. Serfon, G. Stewart, E. Vaand ering, P. Vasileva, R. Vigne, and T. Wegner, "Rucio - scientific data management," *arXiv e-prints*, p. arXiv:1902.09857, Feb 2019.

[11] P. Calfayan, Z. Dongsong, and V. Garonne, "Usage of Message Queueing Technologies in the ATLAS Distributed Data Management System," Tech. Rep. ATL-SOFT-PROC-2011-010, CERN, Geneva, Jan 2011.

[12] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[13] J. Reback, W. McKinney, jbrockmendel, J. V. den Bossche, T. Augspurger, P. Cloud, gfyoung, Sinhrks, A. Klein, M. Roeschke, S. Hawkins, J. Tratner, C. She, W. Ayd, T. Petersen, M. Garcia, J. Schendel, A. Hayden, MomIsBestFriend, V. Jancauskas, P. Battiston, S. Seabold, chris b1, h vetinari, S. Hoyer, W. Overmeire, alimcmaster1, K. Dong, C. Whelan, and M. Mehyar, "pandas-dev/pandas: Pandas 1.0.3," Mar. 2020.

[14] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679 – 688, 2006.

[15] A. Zheng, *Evaluating Machine Learning Models*. O'Reilly Media, Inc., 2015.

[16] S. Makridakis, "Accuracy measures: theoretical and practical concerns," *International Journal of Forecasting*, vol. 9, no. 4, pp. 527 – 529, 1993.

[17] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[18] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[19] F. Chollet, *Deep Learning with Python.* USA: Manning Publications Co., 1st ed., 2017.

[20] G. h.-j. The dataset recorded at the Weather Station at the Max Planck Institute for Biogeochemistry in Jena, *Weather archive Jena Air temperature, atmospheric pressure, humidity, recorded over seven years*, (accessed August 4, 2020). https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip.

[21] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach.* Pearson, 6 ed., 2016.

[22] J. I. Bogado Garcia, M. Lassnig, F. Monticelli, and C. Serfon, "Poster: Estimating Time To Complete for ATLAS data transfers," in *CHEP 2018 Conference - National Palace of Culture, Sofia, Bulgaria.*, Jul 2018.

[23] CERN, *FTS 3 Optimizer Documentation*, (accessed February 10th, 2021). https://fts3-docs.web.cern.ch/fts3-docs/docs/optimizer/optimizer.html.

[24] J. Bogado, *The source code of the FTS simulator used in this thesis.*, 2021. https://github.com/jwackito/ftssimu.git.

[25] M. Lassnig, M.-S. Barisits, P. Laycock, C. Serfon, E. W. Vaandering, K. Ellis, R. Illingworth, V. Garonne, and G. G. Fronze', "Rucio beyond ATLAS: Experiences from Belle II, CMS, DUNE, EISCAT3D, LIGO/VIRGO, SKA, XENON," Tech. Rep. ATL-SOFT-PROC-2020-017, CERN, Geneva, Mar 2020.

[26] O. Rosso, H. Larrondo, M. Martin, A. Plastino, and M. Fuentes, "Distinguishing noise from chaos," *Physical review letters*, vol. 99, p. 154102, 11 2007.

[27] C. Bandt and B. Pompe, "Permutation entropy: A natural complexity measure for time series," *Physical review letters*, vol. 88, p. 174102, 05 2002.

[28] J. Garland, R. G. James, and E. Bradley, "Quantifying time-series predictability through structural complexity," *CoRR*, vol. abs/1404.6823, 2014.