Faculty of Mathematics and Natural Sciences
Department of Physics
University of Wuppertal

A popularity prediction and dynamic data replication study for the ATLAS distributed data management

Dissertation

by

Thomas Beermann

Version July 5, 2017

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20170707-130937-5
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20170707-130937-5]

Abstract

ATLAS (A Toroidal LHC Apparatus) is one of several experiments of at the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland. The LHC is the largest and most powerful particle accelerator in the world, which is able to operate at unprecedented energy levels. Because of this, ATLAS is able to observe physical phenomena and massive particles that were not observable before.

The detectors at the LHC itself create vast amount of data that need to be accessible to physicists for their analysis. For this reason a worldwide computing grid (WLCG) was created that connects hundreds of computing centres across the planet. The experiments constantly create new data but older data has to be kept as well. The available resources are limited, which requires a smart management of the storage space.

This thesis presents a method to dynamically create new replicas and delete unused replicas based on a prediction of data popularity to improve user waiting times.

The first part gives an general introduction of the LHC, ATLAS and the WLCG, a description of the computing model and systems used by the ATLAS experiment and finally the motivation for this work.

The second part concentrates on the popularity prediction, introducing how the access data from the grid can be transformed to be used with different prediction methods. The evaluation describes typical usage patterns followed by a discussion of the advantages and disadvantages of the prediction algorithms, which then leads to the hybrid prediction, where two methods are combined to improve the results.

The third part then first introduces the redistribution algorithms that then uses the popularity prediction to delete and add new replicas. After that a grid simulator is described that was developed to study the impact of the redistribution on different workloads. Finally, the evaluation shows the impact of the redistribution on waiting times for user analysis jobs on the grid.

The last part summarises the results and gives an outlook for further developments.

Acknowledgements

This work was supported by the Wolfgang Gentner Programme of the Federal Ministry of Education and Research in Germany.

First, I would like to thank my university supervisor **Prof. Peter Mättig** for giving me the opportunity to obtain a PhD. He encouraged me to apply for the doctoral program at CERN, which was a great experience. A also a big thank you to the group in Wuppertal who helped me a lot establishing the connection to CERN.

Next, many thanks have to go to my CERN supervisor **Graeme Stewart**, who has help me a lot throughout the years. His input in our regular discussions guided me in the right direction.

Also many thanks to my colleagues at CERN. They made the job much more enjoyable. A special thanks to **Mario Lassnig** who always supported me when I needed some time to write and is not only a great colleague to work with but also has become a very good friend over the years.

Then, I would also like to thank **Prof. Nikolaus Wulff** from the university of applied sciences in Münster. Without him the connection to Wuppertal would not have been established and I might not have had the chance to work at such a great place as CERN.

Finally, I am very grateful for the continuous support from my family over the years, especially through some of the more difficult times.

Contents

Part I

Introduction

Chapter 1

The LHC, ATLAS and the WLCG

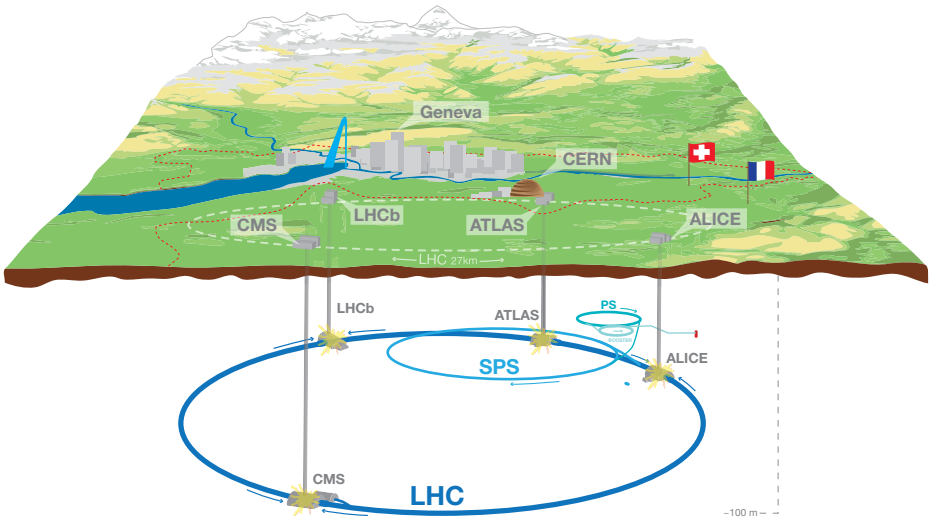
1.1 General

The work that is presented in this thesis was done for the ATLAS collaboration which based at CERN. This chapter gives a general introduction about CERN and the LHC and also provides information about the ATLAS detector and the data which it produces, followed by an introduction of the worldwide LHC computing grid.

1.1.1 CERN and LHC

The **European Organization for Nuclear Research (CERN)** is a European research facility founded in 1952 at the Franco-Swiss border near Geneva, Switzerland. Besides other interests, the main area of research at CERN is particle physics. It therefore provides the needed particle accelerators and other infrastructure. One particularly famous byproduct of this is the World Wide Web which was invented at CERN as a more effective system for communication between scientists. Most activities at CERN are based around the **Large Hadron Collider (LHC)** [1]. The LHC is the largest and most powerful particle accelerator in the world. Figure 1.1 shows an overview of the LHC, including the smaller accelerators and the experiments. It lies around 100 metres underground, spans over both Switzerland and France and the main ring has a circumference of 27 kilometres. There are several smaller accelerators that first accelerate the particles to higher and higher energies until they can eventually be injected into the main ring, which consists of superconducting magnets to keep the beams on track. The main ring then speeds up counter-rotating beams of protons to high ultra-relativistic energies before they eventually collide. The maximum energy is 7 TeV per beam, which results in a collision energy of 14 TeV. The products of these collisions can be used to help understand how the subatomic world is built and which laws of nature

Figure 1.1: Overview of the LHC and its experiments. [2]



apply. The particle bunches collide at specific places around the accelerators. At each of these points detectors are placed that are able to record the collisions and the particles that are created. One of these is the **ATLAS detector**.

The LHC is operated in so-called Runs and Long Shutdowns. During a Run the LHC continuously operates and events are created and collected. Run 1 took place from March 2010 until February 2013. During this run the LHC was operated at energies of 4 TeV per beam. After February 2013 Long Shutdown 1 (LS1) was started to prepare the collider for higher energies. The data taken for this study originates from Run 1 and LS1 and therefore, for the remainder of this introduction, the systems used during this period will be described.

1.1.2 The ATLAS Experiment

ATLAS (A Toroidal LHC Apparatus) [3] is a general-purpose detector of one of the 4 major experiments at the LHC. It is built out of several subdetectors that register and measure different properties of the subatomic particles, which are produced in proton-proton-collisions. The data from the subdetectors is combined and interpreted to reconstruct the interaction that happened when the proton bunches intersected and then

some of the protons collided. A general overview of ATLAS can be found in Figure 1.2. It is 44 metres long, has a diameter of 25 metres and weights around 7000 tons. The essential parts from the inside to the outside are the **Inner Detector**, the **Calorimeters** and the **Muon Chambers**. Each of these detector systems can only detect a certain set of particles so that all these different subdetectors are needed to see the whole picture.

The Inner Detector [4] consists of three detector subsystems which are closest to the beam pipe - the small, vacuumised tube where the particle bunches travel. Huge magnets create a strong magnetic field and because of the proximity to the collisions there is also a lot of radiation, i.e. these parts of the detector have to be very resistant to radiation. Also the transport of the collected data to the outside is a big engineering problem. The detector has several tracker systems that can determine the particle's path and its curvature, which is important information to reconstruct the electric charge and the momentum of the particles.

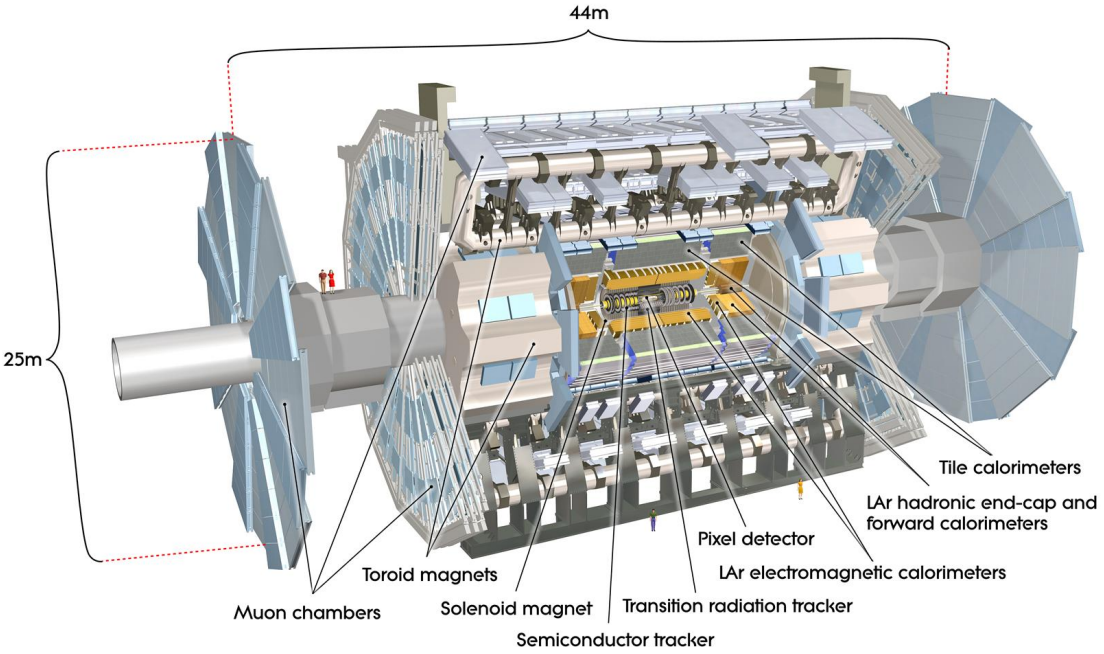
Next around the Inner Detector are the solenoid magnets that create the magnetic field for the inner detector and around them the Calorimeters are placed. These are used to measure the energy of the particles by absorbing them. There are two kinds: The Electromagnetic [5] and the Hadronic Calorimeter [6]. The first stops light particles participating in the electromagnetic interaction, which are electrons and photons, and measures their kinetic energy. Other, high energy particles, like hadrons and muons, cross this first layer, leave little energy there and continue to the second calorimeter. The Hadronic Calorimeter then stops the hadrons, leaving only the muons to continue on. Both systems together help classifying some of the particles created in the collisions.

The outer-most layer, the Muon Chambers [7], finally detects and measures muons that are travelling through the calorimeters without being stopped because of their high mass.

1.1.3 Trigger

The data that is recorded by the different sub-detectors add up to 25 MB raw data per collision. There are 40 million beam crossings per second in the centre of the detector, so in the end the detector can produce up to 1 petabyte of raw data per second. This huge amount of data cannot be stored as at this rate the currently available storage for ATLAS (around 160PB) would be filled up in less than three minutes. The data would have to be sent around the world to distribute it over the grid, but already the transport to the Tier-0, which is located at CERN, would not be possible at this rate. Fortunately only very few of the events contain interesting information and the **Trigger system** [9] is able to distinguish between the interesting events, that will then be stored, and the

Figure 1.2: The schematics of the ATLAS detector. [8]

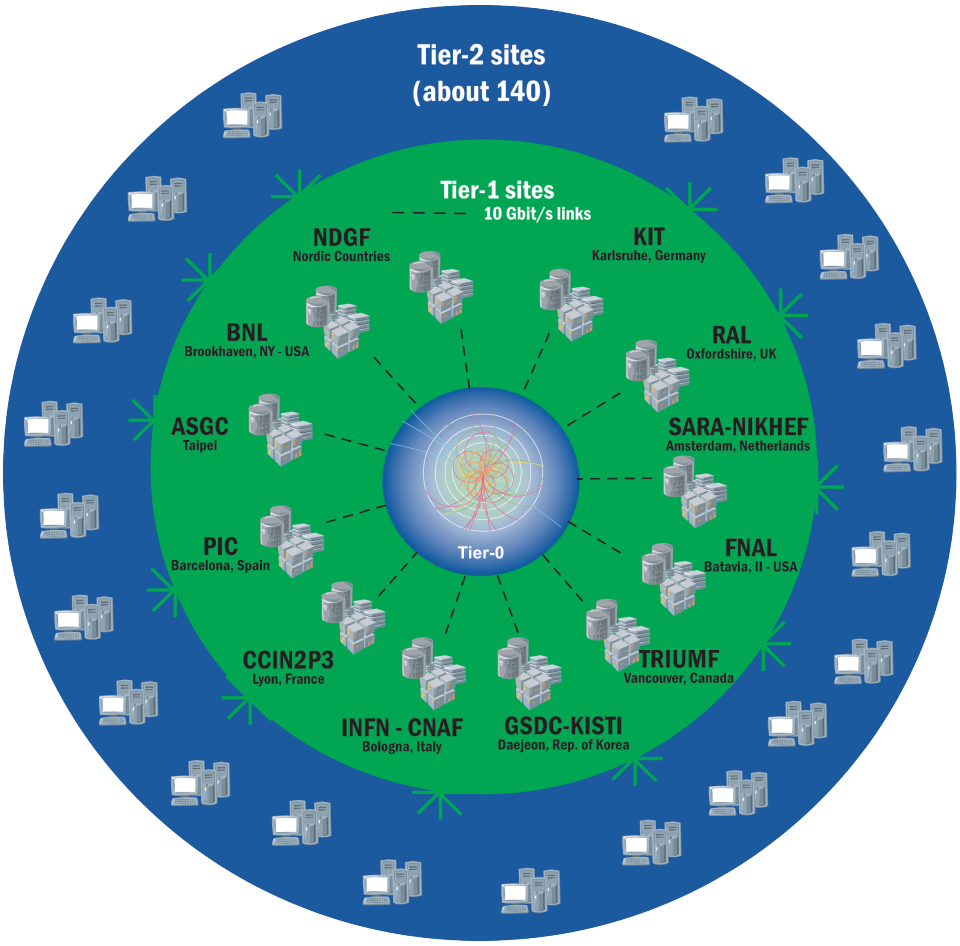


rest, which will be discarded. It used three different levels of triggers. The first trigger is implemented in hardware (FPGAs) and reduces the number of events to 100,000 per second. The next two levels are software-based and run on different computing farms. The second level reduces the event rate to 3000 per second and the last level reduces the rate again 15 times to the final rate of 200 events per second. Overall this results in a reduction factor of 180.000 or 320 MB per second, which adds up to 3.2 PB per year.

1.1.4 WLCG

The data that is produced by the detector needs to be made available for analysis by physicists. As the members of the ATLAS collaboration are spread all over the world at many different participating institutes the data has to be distributed so that all users can accesses it; so at CERN the **World Wide LHC Computing Grid (WLCG)** [10] was developed. The WLCG is a global computing infrastructure that combines the computing and storage resources of more than 170 computing centres in 40 countries in Asia, Europe and the Americas. The sites are organised in a tier system as shown in Figure 1.3. The CERN data centre forms the Tier-0 together with the Wigner Research Centre for Physics in Budapest. These sites are connected with two 100GB/s data links with a latency of around 30-35ms, which are used to synchronise both sites. The responsibilities of the

Figure 1.3: WLCG overview. [12]



Tier-0 are the first pass reconstruction of detector data, the distribution of raw and reconstructed data to the Tier-1s and the long-term storage of raw data.

The Tier-1s are twelve large computing centres based in different countries around the globe. The ATLAS experiment only uses 10 of these, not including FNAL and GSDC. Each of them stores a proportional share of raw and reconstructed data, runs large-scale reprocessing jobs, provides long-term storage on tape systems and distributes data further to the Tier-2 sites.

The Tier-2s are usually located at universities and other scientific institutes and provide storage and computing power to run most of the user analysis jobs. Furthermore they can also be involved proportionally in the event simulation production and reconstruction.

Like the detector itself also the computing systems evolve during and between the different runs. The before-mentioned model was developed at the beginning of Run 1. During the Run the model changed to make more flexible use of the sites [11].

Chapter 2

ATLAS Computing Model and Systems

2.1 ATLAS Computing Model

The previous chapter gave a general overview how the WLCG is designed. This section now introduces the **ATLAS Computing Model**[13], which describes the specifics of the ATLAS share of the WLCG. This includes the types of data that is stores on the grid, the different types of storage and the replication policies.

2.1.1 Data Types

Physics events are stored in different data formats that are derived from each other. Starting from raw data coming from the detector or simulation, the events are processed and converted into different formats that are more suitable for analysis. There are differences in the formats between detector data and simulation for the first steps. Later in the chain they converge to the same data types, where the last step in the chain is then the format used by physicists for their analysis.

Detector Data

- **RAW Data:** Events that are output of the Event Filter (the last step of the high-level trigger) and are ready for reconstruction. The data comes from the event filter as a bytestream and is transferred to Tier-0 in files. Each file contains events for a one or more runs, trigger streams and luminosity blocks.

Simulation

- **Event Generation (EVNT):** Event generation files in POOL[1] format, describing an underlying LHC event.
- **HITS:** The simulated readout of the ATLAS sub-detectors to an event.
- **Read-out Data Object (RDO):** The simulated response of the ATLAS sub-detectors to an event.

Combined

- **Event Summary Data (ESD):** Output of the reconstruction process. The events are stored in an object-oriented representation in POOL ROOT files.
- **Analysis Object Data (AOD):** Reduced event representation derived from ESD. Smaller and more suitable for analysis. Also stored in POOL ROOT files.
- **Tag Data (TAG):** Event-level metadata.
- **Derived Physics Data (DPD):** A selection of the event data from AODs, usually stored as a flat ROOT ntuple for fast access.

2.1.2 Cloud & Tier Structure

The ATLAS computing grid is arranged in a tier structure as introduced in 1.1.4. The first tier is the Tier-0 computing centre at CERN. Then there are multiple Tier-1 centres distributed around the world. Each Tier-1 has a cloud of multiple Tier-2s associated to it.

- **Tier-0:** The main task of the Tier-0 is the first reconstruction, archival and distribution of detector data to the Tier-1s.
- **Tier-1s:** The Tier-1s have multiple purposes. They archive the RAW data but each only a subset of it. They perform the reprocessing of the RAW data and provide access to the ESD and AOD datasets produced in the reprocessing. Also user analysis can be done on a Tier-1.
- **Tier-2s:** Tier-2s mainly run user analysis jobs but also do some reconstruction and simulation. Unlike Tier-1s they do not provide archival facilities.

[1] A hybrid technology store for C++ objects[14]

During Run-1 there were one Tier-0, 10 Tier-1s and around 100 Tier-2s in the ATLAS computing grid.

2.2 ATLAS Computing Systems

After a general introduction and the description of the ATLAS computing concepts this section will focus on the actual system that is used for the data and workload management in ATLAS.

2.2.1 DDM

The **Distributed Data Management (DDM)**[15] system keeps a catalogue of all ATLAS data on the WLCG and provides a central access point for users when they need to access data on the grid. The responsibilities of the DDM system include the organisation, transfers and the management of ATLAS data. The system that was used during Run 1 was named **Don Quijote 2 (DQ2)** and was running in production since 2006. A new DDM system called **Rucio**[16] was developed to remove some of the shortcomings of DQ2 and add new features in preparation for the next big data-taking run started in 2015. The studies in this thesis are still based on the old system and therefore on the next pages the concepts of DQ2 will be introduced, but the ideas presented in this thesis can also be applied to the new system.

Data Model

Most of the data in ATLAS is based on physics events which are represented by C++ objects. This objects are persisted in files. These represent the smallest unit of operation for the DDM system. One file alone usually is only a subset of data of interest for the user of the system.

For that reasons DQ2 has the concept of datasets which allow the aggregation of multiple files in one logical unit. These datasets represent the operational unit for replication in DDM, i.e., only datasets may be transferred between sites whereas single files may not. The physical copies of files of a dataset on a site are called replicas. Datasets may overlap, i.e., the same file can belong to multiple datasets.

In DQ2 there is a second level of aggregation called containers, which is a collection of datasets. Containers cannot be directly transferred but allow larger blocks of data, which cannot be stored on the same site, to be represented in the system.

Data management responsibilities

The DDM system has different responsibilities in the management and the organisation of data. These include:

- Register and catalogue ATLAS data
  - Registration of dataset and containers
  - Registration of files into datasets
  - Registration of datasets into containers
- Transferring data between sites
  - Registering the data transfer request
  - Allowing requests to be queried and cancelled
- Delete replicas from sites
- Ensure dataset consistency on sites
- Enforce ATLAS Computing Model policies
- Monitor ATLAS data activities

2.2.2 Automated Data Replication

Three main mechanisms handle the automatic data replication between the different computing centres. The described tools are the front-ends to inject transfer requests into the DDM system. The actual data movement between the sites is then performed by DQ2.

1. **RAW data from Tier-0 to Tier-1:** The distribution of RAW data from Tier-0 to different Tier-1s is managed by a tool called **SantaClaus** [17]. It respects the RAW data replication policy (explained later) and acts accordingly.
2. **Replication from Tier-1 to Tier-2:** For the automatic replication of data from Tier-1 to their associated Tier-2s **AK47** is used.
3. **User or group requested replicas:** Every user can request additional replicas to particular sites with **DaTRI** [18].

2.2.3 Space Tokens

The storage provided by sites is split in different areas that serve different purposes and that are identified by **Space Tokens**. Common space tokens are:

- **DATADISK:** to store officially produced real data events and Monte Carlo data. It is managed centrally.
- **PRODDISK:** cache used to stage file for production jobs. Managed by the Production System.
- **SCRATCHDISK:** stores the outputs of user analysis jobs temporarily so that users can then download them to their local machine. It is also managed centrally and old datasets are deleted if space is needed for new datasets.
- **GROUPDISK:** stores group data. Managed by space manager of each activity group.
- **LOCALGROUPDISK:** stores locally managed datasets for individual users. The data is still registered in the central data catalogue for all user to be found but the site can decided when and what to delete.

2.2.4 Data Retention

There are different types of dataset replica retention defined as following:

- **Custodial:** this is an archive copy and has to be kept always.
- **Primary:** has to be kept when in this state. Can be changed to secondary.
- **Secondary:** can be deleted if space is needed.
- **Throwaway:** must be deleted even if no space is needed.
- **Group:** dataset copies in GROUPDISK therefore not deleted centrally.
- **Local:** dataset copies in LOCALGROUPDISK. As for Group.

2.2.5 Data Replication Policies

The data replication as defined by the computing model is as follows. It is just the baseline and not fixed and can be adapted anytime by the ATLAS Computing Resource Management (CREM [19]).

- A custodial copy is made for every RAW dataset from the detector, archived on tape at the Tier-0 and distributed over the Tier-1s according to their shares. There will also be a primary copy made onto DATADISK on the Tier-1s. If RAW data is needed for calibration and alignment studies by users it has to be copied first to SCRATCHDISK or LOCALGROUPDISK from a Tier-1.

- The ESD data produced in the initial processing at Tier-0 will be archived on TAPE at CERN and copied to a DATADISK on one of the Tier-1s as a primary copy. A second primary copy may be made on any other Tier-1. There are no copies of ESD made automatically to Tier-2s. The group based analysis is made at Tier-1s. For user analysis an extra copy has to be made to a Tier-2.

- AOD produced at the Tier-0 are archived on Tape at CERN. A primary copy will also be made at the Tier-0 for local use. After the production at the Tier-0 the AOD are copied to Tier-1 DATADISK to be able to replicate to the corresponding Tier-2s. Two primary replicas are kept at the Tier-1s. All other are marked secondary. Additional secondary replicas are made to the Tier-2s for the user analysis.

2.2.6 WMS

The next important system in the ATLAS distributed computing infrastructure is the **Workload Management System (WMS)**. The WMS is responsible to schedule jobs on the grid based on available resources. The current implementation is the **Production and Distributed Analysis System (PanDA)** [20]. It takes care both for scheduling jobs for centrally managed production for ATLAS and independent user analysis jobs.

Architecture

To inject workload into the system tasks are submitted via a central client interface. The server then receives these tasks and puts them in a central task queue. These tasks are then processed by a brokerage module that prioritises and schedules them based on the type of the task, the priority, the input data locality and available computing resources. The tasks are split into jobs which are sent to pilots [21] running on the computing sites. The pilots are small jobs that are sent to the sites and then verify that the worker node (WN) environment, the local disk space, the available memory etc. are suitable for an ATLAS job. If a job slot on the WN is available the pilot asks the PanDA server for a new job, which it then executes and monitors.

Scheduling

The scheduling of jobs differs between production and analysis tasks. The production tasks are centrally defined based on physics needs and available resources in ATLAS. The tasks are split into multiple jobs based on available disk space, the data locality, available and computing resources. Those parameters are taken into account when to decide where to send the jobs. If not all needed data is complete at one site, data transfer requests can be submitted to the DDM system. The jobs then wait in the task queue until the DDM system notifies PanDA that the data has been transferred.

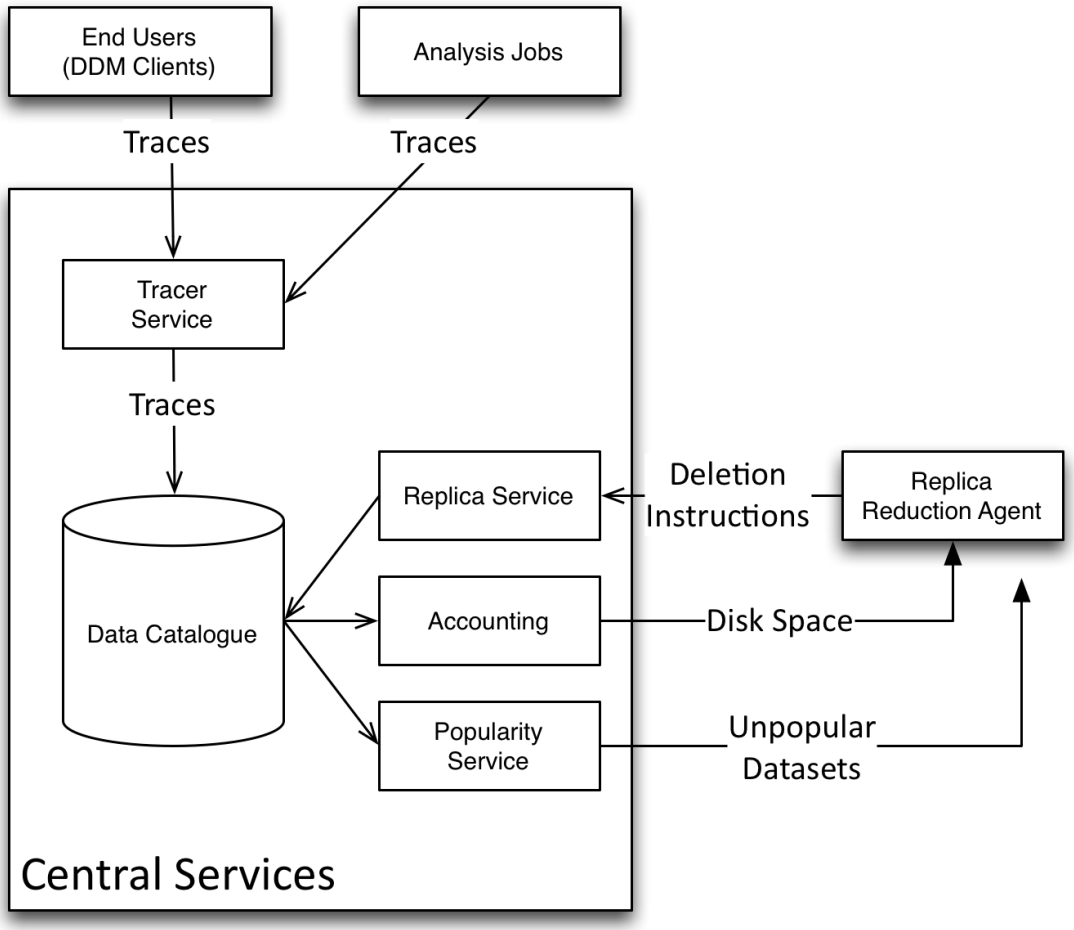
For analysis tasks, however, the workflow is different. The tasks are not defined centrally, but are sent in more chaotic ways directly by the users who want to run an analysis on the grid. Unlike production tasks analysis tasks will not trigger data transfers and are therefore always sent to sites where the input data is available. The workflow starts when the user submits a task to the PanDA server. This task is split into several jobs based on the input dataset locality and the available CPU resources at sites. These jobs are sent to the pilots on the sites. On each site first a compile job will be run that receives the source files as defined by the user and compiles them. After that multiple execution jobs are triggered when the input data has been transferred to the local storage of the WN. The execution jobs then run the compiled code on the input data on the WNs in parallel. The output of the jobs on site will be collected into an output dataset and the different output dataset will be placed in one output container than can then be accessed by the users using the DQ2 tools.

2.2.7 Tracer

The DDM and WMS are the core systems that are used for all interactions with ATLAS data on the WLCG. There are systems built on top of them to provide additional services, which will be important for this work, like the **Tracer** [22].

Every data access made on the grid, either through the WMS or directly through the DDM system, is tracked by the Tracer system. To do this clients send messages to a central server for every single file that has been accessed. The records are then stored in an Oracle database. All the fields that are recorded by the Tracer can be found in Table 2.1. The use cases for this data are diverse, e.g. analysing past transfers, spotting errors during transfers, etc., hence the variety of fields. But most important for this study are the time, the event type, the user id, the dataset and the involved sites. With this information it is possible to analyse past data access patterns and also the popularity of datasets.

Figure 2.1: Overview of the tracer and popularity components.



Attribute	Description
UUID	Unique identifier shared by traces that originate from the same operation
Event Type	Application type initiating download, e.g., an analysis job
Tool Version	Release version of grid application
Remote Site	Site where data is being downloaded/uploaded to
Local Site	Site where data is being downloaded/uploaded from
Time Start	Time operation started
Time End	Time operation ended
DUID	Unique dataset identifier
Version	Version of dataset
Client State	Final state of client, e.g., done or error code
File Size	Size of files
GUID	Unique identifier of file
Catalog Time	Time stamp when file catalogs were queried
Transfer Start	Time stamp when first byte arrived
Validation Start	Time stamp when validation began
Application ID	Operation ID, assigned by the application
Hostname	Name of the host
IP	Host's IP address

Table 2.1: Tracer fields.

2.2.8 Popularity System

The tracer system stores 6 million records per day and has already aggregated over 7 billion rows in total[2] which makes online analysis directly on this data very slow. For this reason the **Popularity service** was developed [23].

The Popularity service works by aggregating tracer entries into access summaries that contain information about how often datasets were requested on the grid. These include both the number of file and dataset accesses. The aggregation is based on the following tracer attributes: dataset name, access day, eventtype, dataset id, local site, remote site and user. This aggregation allows queries like how many times was a dataset accessed by a specific user or how many accesses were made on a particular site. Because of the aggregation by day the result is much smaller and more condensed and allows for much faster access to the results.

[2] as of June 2014

2.2.9 Victor

A current use case for the Popularity system is the automated replica reduction agent **Victor** [24]. The objective of Victor is to clean up sites when they become full of data. In order to do this the agent gathers replica popularity information from the Popularity system and identifies unpopular replicas that can be deleted to clean up space. The process has two steps that run regularly:

First the agent has to identify sites that are about to run full. Next, to get the current space usage of a site the Accounting service[3] is queried. Victor then checks if the space has reached a certain threshold. The threshold has to be smaller than the actual storage space of the sites. This is important because the deletion of replicas takes some time and the sites would be blocked for writing until space has been freed.

When a site has been selected the system starts identifying possible replicas for deletion. This is done in multiple steps. First the replicas are sorted from oldest to newest. Next, all replicas that are not secondary and which are not younger than 15 days are filtered out. The time period of about two weeks is taken, because it usually takes some time after the replica creation until it is accessed the first time. This list of replicas is then passed on to the deletion agent of DQ2, which then will take are of the actual deletion of replicas.

2.2.10 PD2P

The **PanDA Dynamic Data Placement (PD2P)** [25] system tries to distribute data by taking into account the popularity, locality, usage patterns of the data and available resources. The goal is to replicate popular datasets to make better use of CPU and storage resource distribution. The main motivation of this service was the fact that the policy-based data distribution of ATLAS of two copies at Tier-2s filled the sites with replicas that were never used. PD2P tries to overcome this problem by making extra replicas only when there are a certain amount of jobs for the same dataset waiting in the queue. After the replica has been created successfully some of the waiting jobs can be rescheduled to take advantage of the new replicas.

For the replication PD2P does not consider datasets that are output of user analysis jobs, but only official datasets that are the result of production jobs. The service is only triggered when end-users submit jobs to sites that are dedicated to analysis activities and production and testing jobs are ignored.

[3] DQ2 service that gives detailed information of how much space is used at a site and what kind of data is stored there, based on replica information from the data catalogue.

Chapter 3

Motivation

Data distribution in ATLAS currently relies a lot on static policies that are defined for long time periods and on manual creation of extra replicas. This has lead to a state where a lot of data that is stored on the grid is only rarely used or is not used at all. On the other hand, popular datasets could be using this space to make a broader distribution of often used replicas. Analysis jobs can only be sent to a site if the needed replica is available. If a popular dataset is only available on sites which are running at full capacity the job then has to wait in a queue. Other sites might have free computing resources, but not the needed replicas and therefore could partly run idle. If more replicas of these popular datasets would be available the scheduler would have more options to choose from and would be able to spread the workload over more sites, leading to a better utilisation of the computing resources. Information about popular datasets is available and can be used for this. The PD2P systems tries to address this problem but only by retroactively creating new replicas when the jobs are already submitted to the WMS.

This thesis therefore proposes a system that is able to anticipate potentially popular datasets and that will redistribute replicas to make better usage of the available resources. It uses the popularity information to make predictions of how datasets will be accessed in the future. Based on that information, unused replicas will be deleted and new replicas will be created. The benefits of such a system are measured by the time jobs have to wait before they start and the usage of computing resources. To be able to measure these metrics a grid simulator will be introduced, that is able to take historic workloads and then replay them on different data distributions.

Part II

Prediction

Chapter 4

Popularity Prediction

This chapter is about the prediction of dataset popularity, i.e., how often will a particular dataset be accessed in the near-term future. The chapter starts with a general introduction into time series prediction. It is followed by an overview of the available input data and how it is transformed to fit the time series requirements. After that the three different predictions methods that are evaluated in this thesis are described in turn followed by the introduction of the hybrid prediction algorithm. The chapter finishes with details about the implementation.

4.1 Introduction

This section gives a general definition of time series that will be used throughout the whole chapter. It furthermore defines classes of time series prediction.

4.1.1 Time Series

A time series is a series of vectors that depend on time t [26]. It is formally defined as follows:

$$\vec{x}(t), t = 0, 1, ...$$

If the vector consists of only one component it is called univariate; otherwise it is multivariate. For the remainder of this thesis only univariate time series are examined. A time series can be made of all kinds of observable objects that can be converted to numbers, e.g., the outside air pressure, the value of a company share or the amount of computers sold in a store. In theory $\vec{x}$ is a continuous function of a time variable t, but in most practical use cases it is handled in discrete time steps. That means there is an instance of $\vec{x}$ at some point in time t. How this point is measured depends again on the use case, in

some cases it is just the reading at a point in time, in other cases it is an aggregate or an average over a time interval, e.g., for air pressure the measurement reading at that point of time will be taken. For the amount of computers sold it is the sum of all computers sold between two points in time. The actual interval again depends of the application and can range from microseconds to years.

4.1.2 Time Series Prediction

There are different kinds of prediction of times series depending on the application. The two most common are:

- **Forecasting:** The prediction of future values of the time series, i.e., taking a number of values of past entries in a time series and calculating a future value. Formally this means finding a function $F : \mathbb{R}^n \rightarrow \mathbb{R}$ so that $x(t + d)$ can be obtained using the values of x for a time interval up to t. d is called the lag and typically $d = 1$, i.e., the next entry in the time series is predicted. It is then formally defined as:

$$x(t + d) = F(x(t), x(t - 1), ...)$$

- **Classification:** In some use cases the actual value of $x(t+d)$ is not important but rather if the value is lower of higher than the previous value. Or more generally to associate $x(t + d)$ to one of several predefined classes. This is then called a classification problem. Formally this means to find a function $F : \mathbb{R}^n \rightarrow \mathbb{C}$, where $\mathbb{C}$ is a set of predefined classes, so that:

$$F : (x(t), x(t - 1), ...) \rightarrow c \in \mathbb{C}$$

This study evaluates if it is possible to predict future dataset accesses, so for the remainder of this chapter only the forecasting method is considered.

4.2 System Design

The previous section described how time series prediction is defined formally. In this section it is described how the input data that is available can be transformed so that it can be used with any time series prediction algorithm. After that, the actual forecasting methods that were chosen and evaluated are introduced.

4.2.1 Input Data

The input data is taken from the tracer and popularity systems that were introduced in 2.2.7. The tracer system can at any point in time receive data points that represent file-level data accesses. This data now has to be transformed to get a time series for each dataset with data points that correspond to a number of accesses for this dataset for a chosen interval.

Original Data

The complete format of the traces data were already described before. For the predictions only a few fields are required:

- **Time Entry:** The timestamp when the system received the trace.
- **UUID:** A unique identifier that is shared among traces from the same operation.[1]
- **Event Type:** The type of application that sent the trace.
- **Dataset Name:** The name of the dataset that was accessed.

Transformation

Before the actual time series data can be built, the input data first has to be filtered for irrelevant data. The first step is to filter the data based on the time entry. The prediction is based on a certain time period in the past that will be used to predict a future value, so only traces that belong to this time period are needed and the rest are filtered out. The next step is to filter the input data based on the event type. There are two main categories for traces: `put` and `get` events. `put` events correspond to operations, where data is stored onto the grid, e.g., when a user uploads a file or when a finished job writes the output back onto the grid. Those events are irrelevant for the popularity prediction. `get` events correspond to operations, where user request data, e.g. when a user downloads a file or when a job uses data for analysis. Those event are the relevant ones for the popularity prediction, as they show which data is actually used. Generally, if there are more `get` events for a dataset, then it is more popular. So first the `put` events are filtered out of the data. Next, there are three main kinds of `get` events: one for user analysis jobs, one for local downloads and one for production system jobs. As

[1] As described before, a dataset is a set of files. For each single file access a trace is sent, but this study concentrates on dataset level accesses, i.e., the traces have to be aggregated. To do this each access to the same dataset shares the same UUID by which interrelated traces can be identified.

laid out before, the system is designed to improve waiting times for user analysis jobs, so accesses from the other two kinds are not relevant and are therefore filtered out.

After the filtering the data it can be transformed to the needed time series format. The time series will be on a per-dataset basis. So first, the data is split up based on the dataset name. As it will be a discrete time series the data then has to be split based on the time entry so that there will be a set of traces for each time period. The last step now is to aggregate all the traces for each time interval to get one single data point for this period. The data points that remain now per time period have the same event type and dataset name and the only difference is the file name that was accessed and the user who accesses it. Just counting the number of traces for the period would not measure popularity usefully. A dataset with a lot of files would have a lot more accesses than another one with just a few files. It is the popularity of a dataset that is interesting, not its size. Furthermore as datasets can only be handled as a whole in the DDM system, it is irrelevant which particular files in a dataset have been accessed. Therefore the UUID is important here. A new UUID is created for each coherent user operation, e.g., if a user requests all files from one dataset for a job all resulting traces will have the same UUID. Further if a user request multiple datasets, e.g., in a container, for the same analysis job, there will be traces with the different dataset names but the same UUID. So this means that in order to get the aggregated accesses for a time period all distinct UUIDs have to be counted.

This process, which is also illustrated in Figure 4.1, then results in time series for a period with n data points of the format:

$$A_D(t), t = 0, 1..., n$$

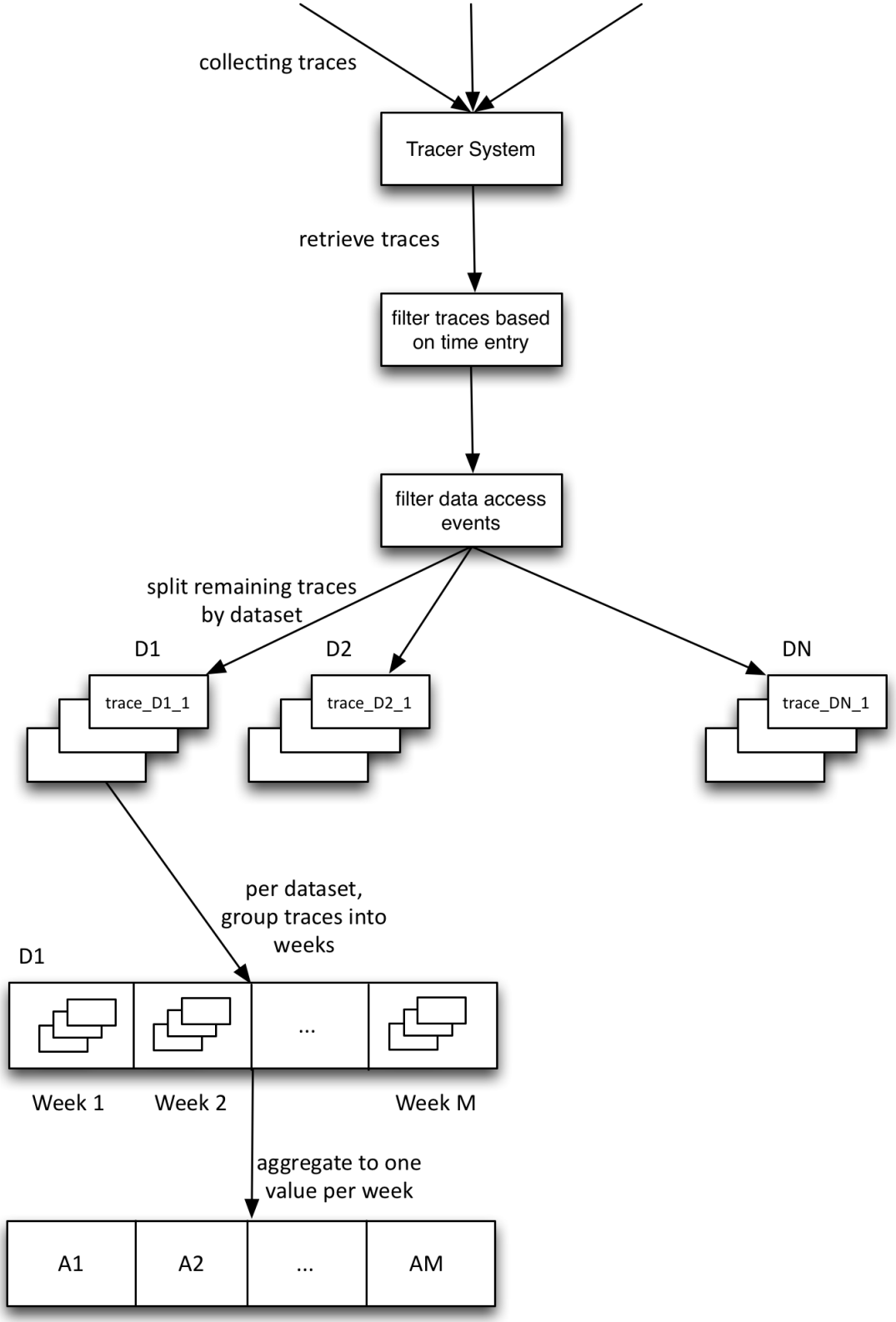
where $A_D(t)$ is the number of distinct operations accessing dataset D at time t.

Those time series are produced for all currently available datasets and will be used in the next step to predict future accesses.

4.3 Prediction Methods

After the formal definition of time series and how the tracer data can be transformed into time series this section introduces the actual prediction methods that will be used in the remainder of this study. There are a wide variety of algorithms available for time series predictions but this study concentrates mainly on three of them, the static prediction, the linear prediction and the neural network prediction. For reasons described later the static and neural network predictions are then combined into a hybrid method, in

Figure 4.1: The data preparation and transformation.



order to a achieve better accuracy. The interval between data points in the time series is chosen to be one week. This time period averages over daily peaks, smooths out weekday/weekend differences, and is also a good base for the redistribution, as it gives enough time to do the actual transfers needed. The time series is defined as before as $A_D(t), t = 0, 1..., n$ and the prediction has to find the value $A_D(n+1)$.

4.3.1 Static Prediction

One of the simplest time series prediction algorithms is static prediction. It is not really a time series prediction as it does not take the complete time series made out of several data points into account, but only the last point. It is defined as:

$$A_D(n+1) = A_D(n)$$

It works under the assumption that the accesses stay rather stable from one week to the other.

4.3.2 Linear Prediction

The next prediction algorithm is the linear prediction. Here the predicted value for the next week is the value for the past week as for the static prediction, but on top the difference between the values from the last week and the week before that are added, i.e., the increase or decrease between the two previous weeks it added:

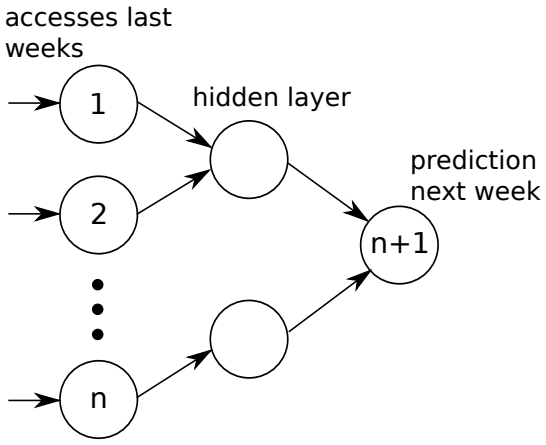
$$A_D(n+1) = A_D(n) + (A_D(n) - A_D(n-1))$$

So if there has been an increase of accesses from the second to last to the last week it is also reflect in the prediction as a linear increase and also the other way around if there has been a decrease.

4.4 Neural Network Prediction

Artificial Neural Networks (ANN) are a set of machine learning algorithms that are based on the way biological neural networks, like those in the human brain, work. They are a set of interconnected neurons that take in some input that is transformed in the possibly several inner layers to generate some output. They need to be trained based on some training data and are capable of picking up patterns, which they then can apply to new data, which matches those patterns. They can be especially useful in cases where the correlation between the input variables is not directly obvious and standard rule-based

Figure 4.2: A simple artificial neural network setup



algorithms would be difficult to use. This is the case in this application. ANNs have been the subject of a vast number of studies and have already proven to be useful for time series predictions and are therefore chosen in this study. The first studies using neural networks for time series prediction where presented around 1990 to estimate voice traffic demand [27]. Others used them to forecast financial and economic data [28]. They also have be proven to work well for chaotic time series [29] and in more recent work they have been used to forecast exchange rates [30].

4.4.1 Basics

Figure 4.2 shows an example of a simple neural network. It has several neurons at the input layer, one neuron at the output layer and some neurons between that are in the hidden layer. The neurons of the input layer have connections to the hidden layer and the output neuron is connected to the hidden layer. The connections between the neurons are called synapses and the arrow shows in which direction the data is sent through the network over the synapses. If information always travels in one direction through the neural network it is called a feed-forward neural network. If the information can also travel back to previous neurons it is called a recurrent neural network. The synapses have different weights that manipulate the data when it is sent through that synapse. The neurons themselves have activation functions that define if a transformed value received from a synapse will be sent over the next synapse or not.

4.4.2 Training

One of the most interesting things about ANNs is the ability to adapt the weights to fit specific problems and therefore store knowledge of this problem in the network. This

process to find values for the weights is called learning. There are mainly two different types of training:

- **Supervised:** Training with a dataset[2] with inputs and known outputs.
- **Unsupervised:** Training with a dataset without known outputs.

Unsupervised learning can be especially useful for classification problems, whereas supervised learning is useful for pattern recognition. Therefore in this study supervised learning is used.

In supervised learning a dataset is used with known patterns, i.e., the output value for a set of input values is known. This training dataset usually consists of a set of typical patterns for the problem to solve, that the network should be able to detect later. The dataset is split in a training dataset and a testing dataset. The input data from the training dataset is fed to the input nodes of the network multiple times, each time comparing the output of the network with the output given in the training data based on a pre-defined error function. After every run the weights are adapted accordingly to minimise the error. The process is terminated either when the algorithm arrives at the pre-defined maximum number of learning rounds or when the pre-defined error threshold has been reached. After the training, the testing datasets is used for evaluation of the quality of the newly trained network by feeding it to the network and comparing the outputs with the actual outputs from the dataset. In this way it can be made sure that the network is trained correctly to detect patterns for the whole problem and not only for the training dataset.

4.4.3 Dataset Access Prediction

After the introduction this part describes the general concept of how the ANNs are used to predict dataset accesses. The ANN is trained to pick up typical access patterns to apply them to new data. The idea is that typical access patterns repeat over time and therefore an ANN trained to detect previous accesses patterns can also detect those patterns in new data.

Example

Later in this section a general description will be given but first a simplified example of this process is described (Figure 4.3). In this example an ANN is trained using the

[2]Until now the term dataset has only been used to describe a set of physics data. To avoid confusions, from now on those will be called DDM datasets.

previous accesses of three different DDM datasets (Dataset 1, Dataset 2 and Dataset 3). The three datasets are in different phases in their lifetime, the three DDM datasets have quite different access patterns and those can be picked up by the neural network. The accesses for Dataset 1 vary a lot between each weeks, whereas Dataset 2 and Dataset 3 have more stable, but distinct, patterns. Datasets 1 and 3 are similar in the beginning as they get popular quickly, but as Dataset 3 stays popular for some time. The popularity for Dataset 1 goes down and up again two times until it becomes popular again in the target week. Dataset 3 on the other hand is not popular anymore in the last week. The popularity of Dataset 2 slowly builds up to hit its peak at the beginning of the last week, but then it is going to become unpopular again. The neural network is now trained with the accesses of those three datasets as input and the accesses of the last week as output. After the training, the neural network will be used to predict the accesses for another Dataset 4. The access patterns for Dataset 4 resemble those of Dataset 1, i.e., it goes up and down a few times until it is popular again in the last week. If the training of the neural network was successful it will pick up the pattern and produce a similar output for Dataset 4 as for Dataset 1.

General Description

The input and output values for the neural network prediction is based on the per-dataset time series as defined before:

$$A_D(t), t = 0, 1, ..., n-1, n$$

This data is now split into input and target datasets for training and into the input dataset for the actual prediction. The neural network will be set up to have $n-1$ input neurons and 1 output neuron.

For the training the input dataset is:

$$A_D(t), t = 0, 1, ..., n-1$$

The output is:

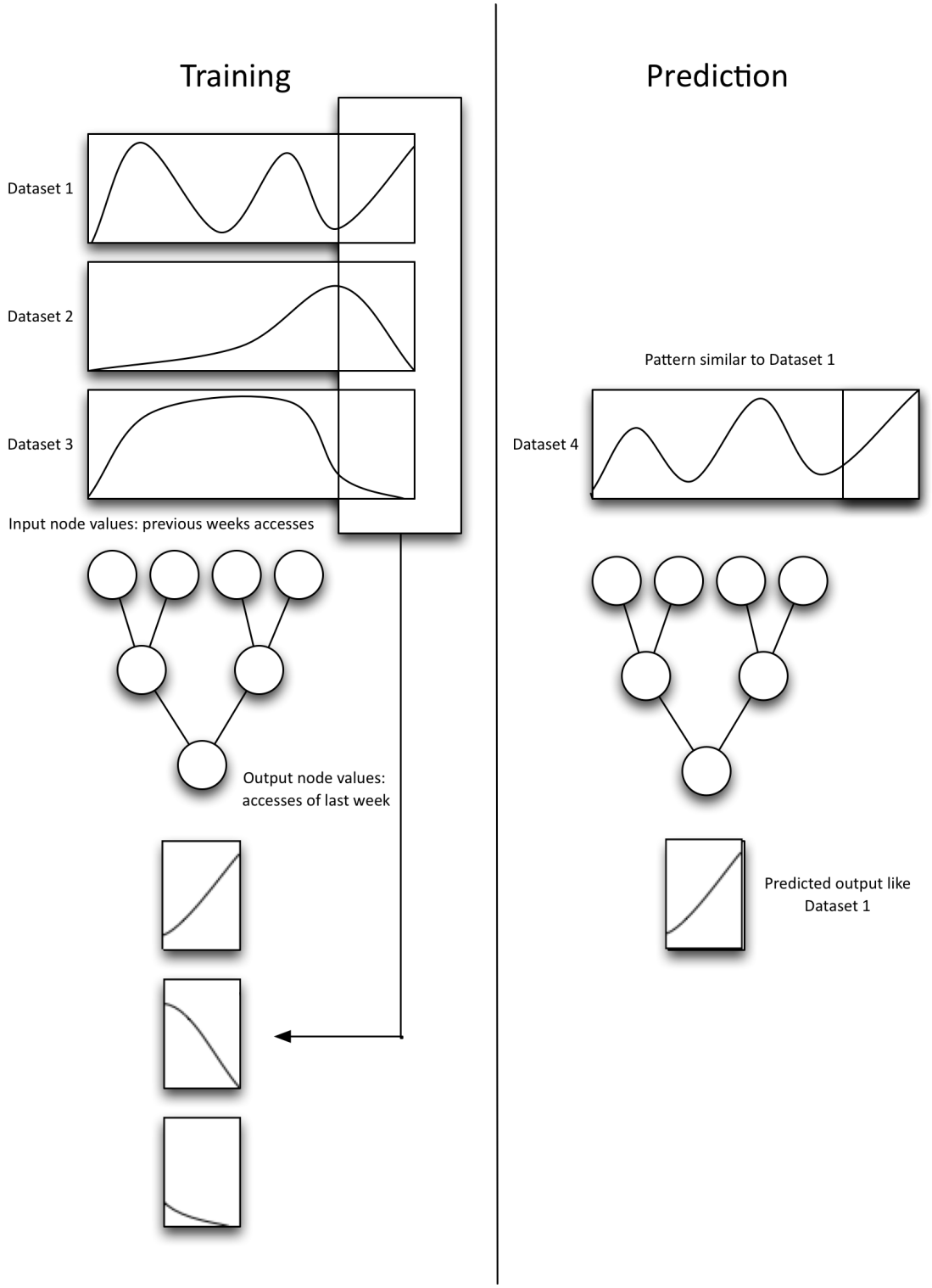
$$A_D(n)$$

25 percent of the data will be taken away and not used for the training. This is the test dataset which will be used after the training to evaluate the trained network.

After the training is done and the network has been evaluated the network will be used for training. For this the input data will be shifted to get the following access time series:

$$A_D(t), t = 1, 2, ..., n$$

Figure 4.3: The ANN training and prediction.



The output will be the predicted output for $n + 1$:

$$A_D(n + 1)$$

The underlying assumption in this method it, that the access patterns repeat over time and that the neural network picks up those recurring patterns to apply them to other datasets.

4.4.4 Neural network separation

The ATLAS DDM system stores a lot of different data. For this evaluation only the officially produced datasets from the detector and simulation are used. And also from those only the datasets that are in the data format used for analysis. But even for this limited set of DDM datasets there is still a lot of data that is used by different analysis groups and physicists resulting in a lot of potentially different access patterns.

Official datasets follow a naming convention as described in [31] and which is defined as follows:

project.[otherFields].dataType.version

The two most import fields in this case are the project and the datatype. Typical projects would be **mc12_8TeV** or **data12_8TeV** for 2012 Monte Carlo simulation data or detector data respectively. The datatypes have already been introduced in 2.1.1 and can be anything from **RAW** over **AOD** to **NTUP**. The NTUPs then again are split into many different sub-types for different types of events, e.g., **NTUP_SMWZ** for standard model W and Z boson events.

Using one big neural network for all those different project and datatype combinations does not work at all. As shown later in the evaluation at 5.2, the patterns are similar for the same project and datatype but can differ a lot across all DDM datasets and one big neural network has problems converging in this case. To circumvent this problem multiple neural network are trained, one for each project / datatype combination.

4.5 Hybrid Solution

The previous section described the different prediction algorithms separately from each other, but depending on the data it can make sense to combine the algorithms for better performance. The separate evaluation of the different prediction methods in the next chapter in section 5.3 shows, that overall the neural network prediction can produce better results than the other prediction mechanisms, but its performance can be biased

by the unpopular data. Therefore it is helpful to filter the input data to remove the unpopular data, so that the neural networks are used for the popular data and the static prediction for the unpopular data. This process is described in this section.

4.5.1 Prefiltering

Depending on the project and datatype there is the possibility of a lot of unpopular data, i.e., data that has been very rarely or not been accessed at all. When the neural network is trained with this data without filtering the unpopular data the network can be biased to the unpopular data with then results in under-predicting, as shown in 5.3.4. To mitigate this effect a pre-filtering step is introduced in this part that splits the unpopular from the popular dataset so that it later can be handled with separate prediction methods.

The filtering is based on two variables that are calculated for each DDM dataset in the input data:

- **Access Ratio:** This is the ratio of weeks with any accesses compared to all weeks. It is defined as follows:
$$U = \frac{1}{N} * \sum_{i=1}^{N} U(i)$$
where $U(i)$ is:
$$U(i) = \begin{cases} 1 & \text{if } A_D(i) \text{ != } 0 \\ 0 & \text{if } A_D(i) = 0 \end{cases}$$
- **Average Accesses:** This is average of all accesses for one dataset over
$$A_D = \frac{1}{N} * \sum_{i=1}^{N} A_D(i)$$

Those two metrics are used separately for each DDM dataset, both in the input data for the training and later for the input for the prediction. If the access ratio for a dataset is below 50%, i.e., it has no accesses for half on the period, than the static prediction will be used for this DDM dataset. Otherwise the average accesses will be calculated and if it has not been accessed on average more than 5 times a week, also the static prediction will be used. Everything else is fed to the neural networks.

4.5.2 Hybrid Prediction

After all the different components of the prediction have now been introduced the whole workflow, from the collection of the data until the final output of the predicted accesses, can be described. The process is also illustrated in Figure 4.4.

The first step is the collection of the raw traces from the tracer system. The traces are then transformed into a weekly format with aggregated accesses per week per dataset. The next step is then the pre-filtering of the data based on the popularity, so that the access data for unpopular datasets is removed from the training input for the neural networks. Then, in a next step the popular data for the neural network prediction is again split into different groups based on the project and data type of the dataset. The projects and data types are usually accessed in similar patterns as shown in 5.3. Splitting the data based on the project / datatype it is possible to handle data with similar data access patterns in the same neural network. After this step the actual training of the neural networks can start. Each of the neural networks is trained separately using the data resulting from the pre-filtering and grouping. The networks are trained and evaluated until they meet the error threshold. The training will be given a maximum number of iterations and an error threshold. If the error threshold is not met within the given number of iterations the training will be done again. Then, when the training is done the networks are used for the actual predictions. After this, only the predictions for the unpopular data is missing. For that the static prediction is applied. In a last step the results from the neural network prediction and the static prediction are merged to get the predicted accesses for all datasets.

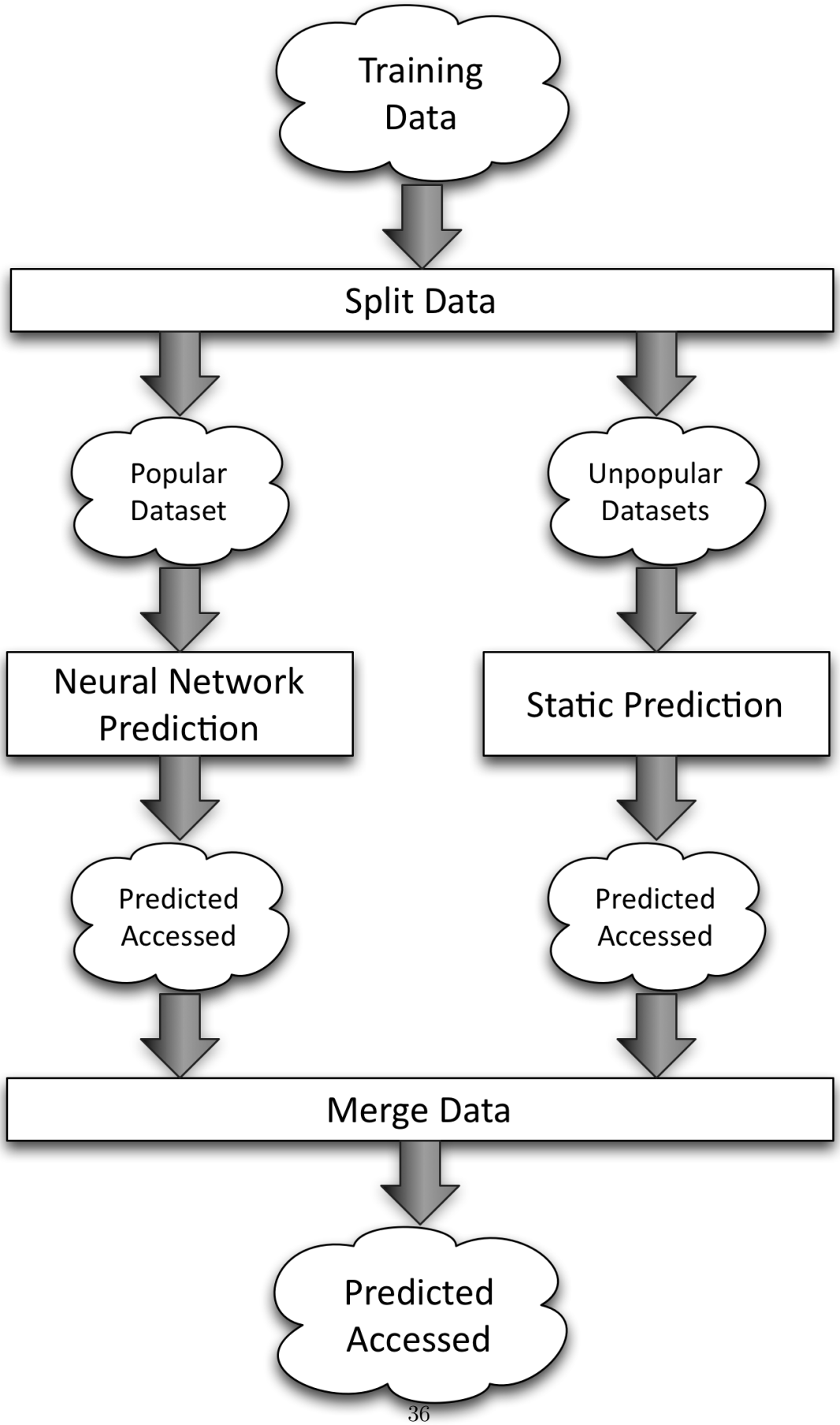
4.6 Implementation

The implementation is done in multiple parts using different tools:

- **Oracle:** The per file traces are stored in an Oracle database. To get the actual accesses per dataset an Oracle query is used that aggregates the traces per day and per dataset.
- **Python scripts:** From this daily aggregation the data has to be further aggregated to get the needed granularity. In a next step the data is then split per data type and project and pre-filtered using the criteria that were described before. For all this a set of Bash and Python scripts are used.
- **Python neural networks:** The actual neural network prediction is done with Python and the **neurolab**[32] package. This combination offers a wide variety of neural network implementations.

The first two parts are rather technical and it is not necessary to describe them here, but the last part will be explained in a simplified way in this last part of this chapter. Actual samples of the code being used can be found in the appendix section 10.1.

Figure 4.4: The Hybrid Prediction Process.



4.6.1 Python neural networks initialisation

In the first step the data has to be prepared and the neural network has to be set up. A class `Neural` is used for that. The class needs a matrix of dataset access data and the number of weeks that will be used for the prediction. A row of the input data represents the access data for one dataset. The columns represent the weeks. For the scale down and scale up the maximum and minimum of the input data has to be found. Then, in a next step the access data can be scaled down (the process is described below). The scaled down data is then split into a training input set, training target set and a prediction input set. For the training input all accesses back to a given number of weeks in the past until one week in the past are taken. The training target set is the accesses in the last given week. Then, for the prediction the training input set is shifted by one week to include everything from the number of weeks minus one until the last week. Next, the neural network is initialised by setting up the input neurons, i.e., creating as many input neurons as there are data point in the training and prediction set and setting the range of those neurons. Finally the network can be trained.

4.6.2 Scale Down/Up

The input neurons require a range of allowed input values but the actual access value can range from 0 to any positive integer. To make sure they fit in the needed range the following two function are used to down-scale and up-scale the data.

4.6.3 Training

The actual training uses three parameters: `goal`, `epochs` and `runs`. `goal` gives the maximum acceptable error, `epochs` are the number of iterations in the training phase and `runs` defines how often the training should be retried if the goal was not met.

4.6.4 Prediction

After the training the neural network can be used for the predictions. In a last step the data has to be scaled up and the results have to be associated to the dataset names again. So the result is a simple dictionary with the name of the dataset as key and the predicted number of accesses as value.

Chapter 5

Prediction Evaluation

This chapter is about the evaluation of the previously introduced prediction algorithms. This includes an overview of typical usage patterns, followed by the evaluation of each of the different prediction methods, including the hybrid prediction method, which combines static and neural network predictions. The prediction evaluation is done using a subset of the available data, which is enough to show both the strengths and weaknesses of the algorithms. For the redistribution evaluation, following later, the fully available data is used.

5.1 Prediction Input Data

First, to show different access patterns across the project / datatype combinations four different typical **NTUP** types across the **mc12_8TeV** and **data12_8TeV** projects and **AODs** from the **mc11_7TeV** project are used. The table 5.1 below shows how many DDM datasets each project / datatype combination contains.

Project/Datatype	Number of DDM datasets
mc12_8TeV.NTUP_SMWZ	5217
data12_8TeV.NTUP_SMWZ	1659
mc12_8TeV.NTUP_COMMON	4834
data12_8TeV.NTUP_COMMON	1452
mc11_7TeV.AOD	1177

The single file traces were aggregated into dataset-level accesses per week using the mechanism that was described before in 4.2. The time period includes 35 consecutive weeks starting from October 2013 until April 2014.

A first thing to notice is the difference in number of datasets between the **mc** and **data**

NTUPs. This is due to the fact that for the **mc** data there are different datasets produced for all possible signals of interest, whereas for **data** all signals are in the same datasets.

5.2 Data Access Pattern Examples

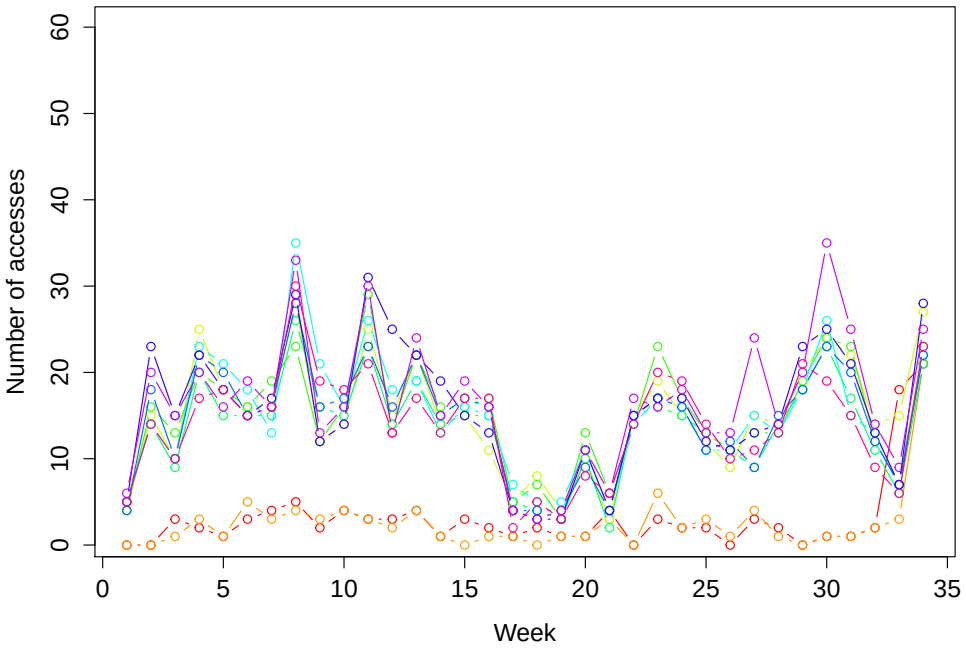
This part goes into detail about the different data access patterns. Overall, there are a lot of combinations of different projects and data types but some of them have only a small number of datasets and others are old and not used much anymore, so the following examples concentrate on only a few combinations that are sufficient to illustrate the different patterns. Two of the chiefly used projects from the time span that is used for this evaluation are **mc12_8TeV** and **data12_8TeV**. The first one corresponds to Monte Carlo simulation data and the second includes events collected from the detector collisions. As explain in the introduction chapter this data goes through different steps from the raw event data to the pre-processed slimmed/skimmed NTUP data types that are used by physicists for their analysis. Depending on the type of research there are different NTUPs created.

5.2.1 Highly popular project / datatypes

One of the more popular ones is the **NTUP_SMWZ** [33], which corresponds to standard model W and Z boson events. Figure 5.1 shows multiple access time series for the **data12_8TeV** project. Because of overlaps, every line in the plots can represent one or more datasets. In this plot is apparent that there are mainly two different types of datasets, unpopular and popular datasets. For the unpopular datasets the usage stays rather constant over time and there are only small changes. For the popular datasets this is different. The popularity goes up and down a lot and there a some recurring patterns. The interesting point is that the pattern is similar for most of the popular datasets. If the number of accesses goes up for one of the datasets it goes for all of them. Furthermore, there are some seasonal influences as well. The time of these plots spans from October 2013 to April 2014 and this also includes the Christmas season. This can be seen in the plots between week 15 and 20, as the popularity goes down for pretty much all datasets at the same time.

The next example is for the **mc12_8TeV** and is shown in Figures 5.2 and 5.3 respectively. Here it was chosen to split the data in two different plots as the patterns differ substantially. Whereas in the first plot the data becomes very popular around week 10, it goes down again around Week 17 and then stays rather unpopular, the data in plot 2

Figure 5.1: Weekly accesses for all datasets in the data12_8TeV project with NTUP_SMWZ datatype. Every line can represent one or more datasets.



stays unpopular for a longer time and then starts being popular in week 25.

5.2.2 Popular project / datatypes

After this very popular data type this section shows two examples of for project / datatype combinations which still are popular but not as much as the two before. Here, the **NTUP_COMMON** datatype has been chosen. This datatype contains common events that are enough for most analysis groups [34], and which is to replace other datatypes. But in the period, that is used in this evaluation, it is overall less popular than the NTUP_SMWZ datasets. Figure 5.4 shows the datasets from the data12_8TeV project and Figure 5.5 for the mc12_8TeV project. While for the highly popular combinations there have been at least two different sets of datasets, very popular and not popular, here all patterns are closer together.

5.2.3 Unpopular project / datatypes

Finally, one example of a project / datatype which is not popular at all. It can be found at Figure 5.6. The **data11_7TeV** project corresponds to detector data taken in 2011. The data itself is already older and the AOD datatype is usually not used in user analysis

Figure 5.2: Weekly accesses for part of the datasets in the mc12_8TeV project and NTUP_SMWZ datatype. Every line can represent one or more datasets.

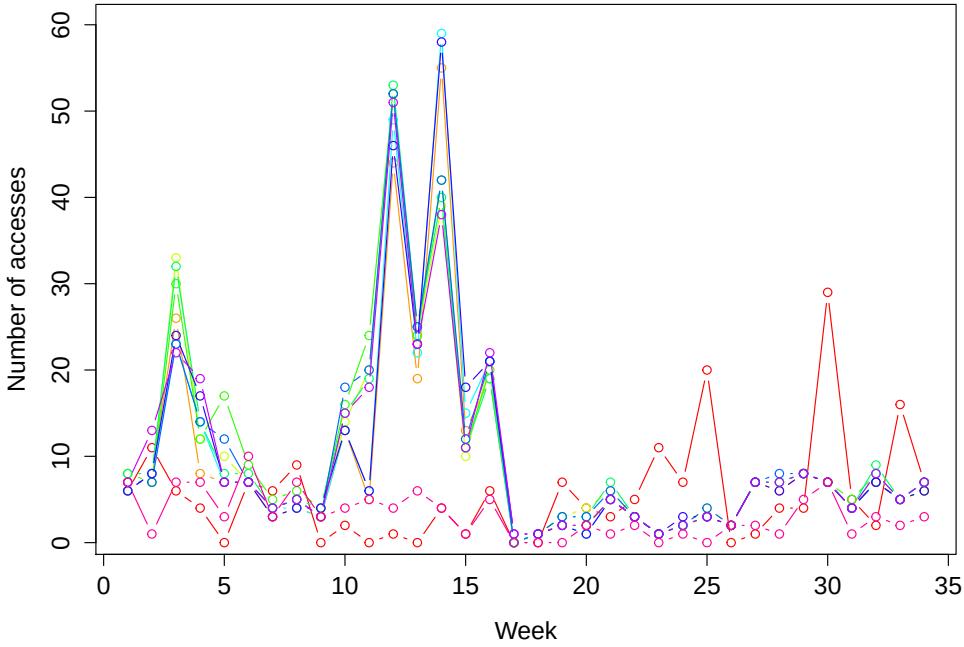


Figure 5.3: Weekly accesses for the other part of datasets the mc12_8TeV project and NTUP_SMWZ datatype. Every line can represent one or more datasets.

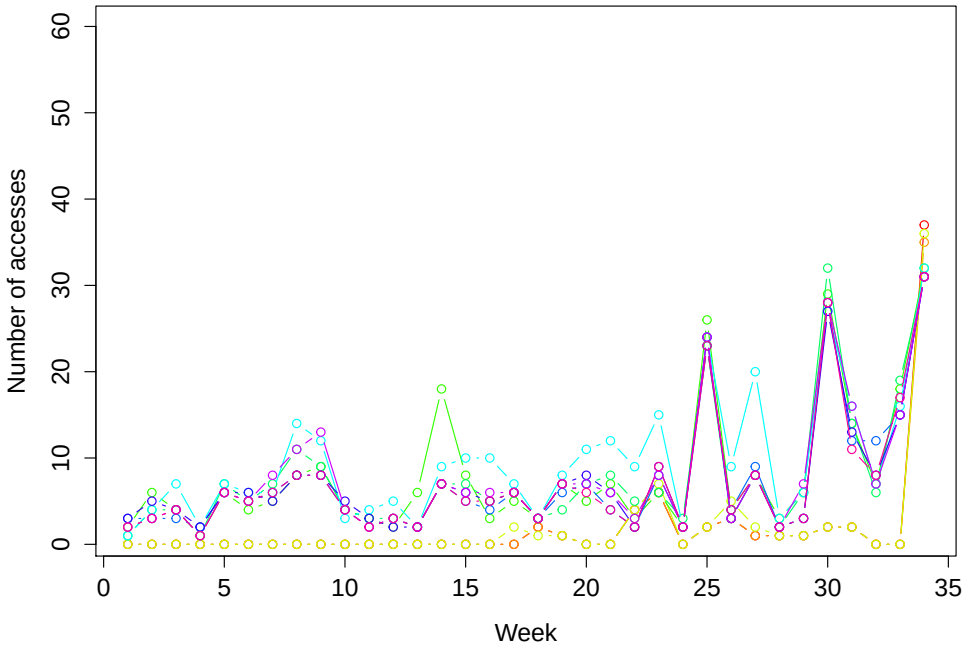
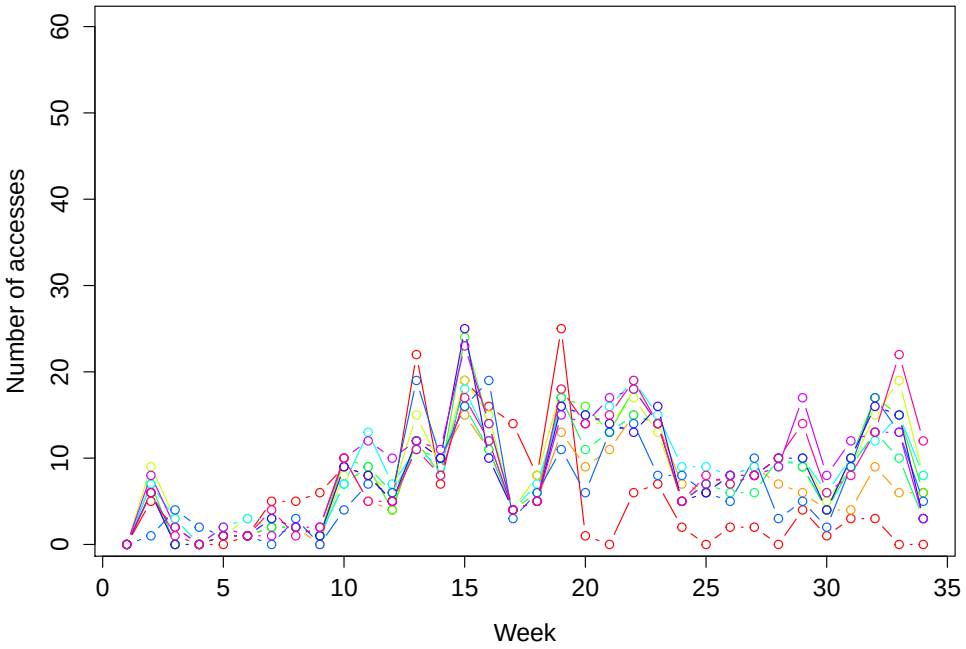


Figure 5.4: Weekly accesses for all datasets in the data12_8TeV project and NTUP_COMMON datatype. Every line can represent one or more datasets.



jobs which leads to this rather flat plot. As project / datatype combinations like this are not used very often in user analysis a dynamic placement is not needed and a prediction is not necessary at all and they can be disregarded.

5.3 Evaluation

This section now evaluates the different algorithms. The main evaluation concentrates on the two **NTUP_SMWZ** datatypes of the **mc12_8TeV** and **data12_8TeV** projects, i.e., separate discussions of each algorithm for each combination. For the main evaluation, only using those two combinations is enough to show the prediction performance for different access patterns. After that the results of hybrid prediction will be discussed, then also including results for the **NTUP_COMMON** datatype.

The access data presented before is used as input for the prediction. Out of the 35 weeks the algorithms have to predict the 31st week. The neural networks consists of 29 input neurons and 24 hidden layers, which have proven to give good results. The training and test sets consists of the data of the weeks 1 to 29 for input and week 30 for output with a 75/25 split for the training and testing sets. The evaluation set consists of the accesses

Figure 5.5: Weekly accesses for all datasets mc12_8TeV project and NTUP_COMMON datatype. Every line can represent one or more datasets.

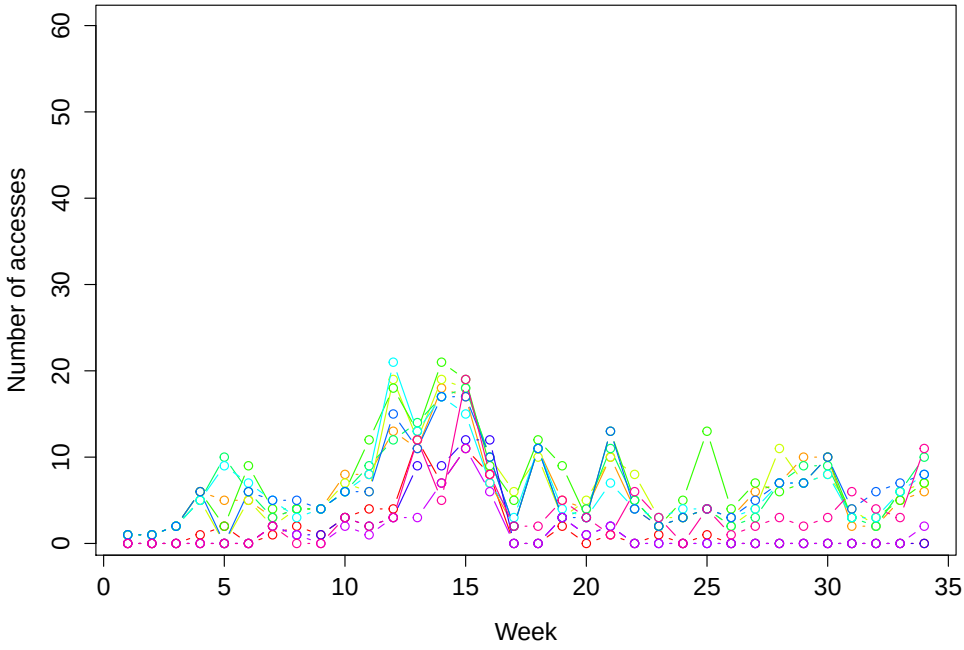
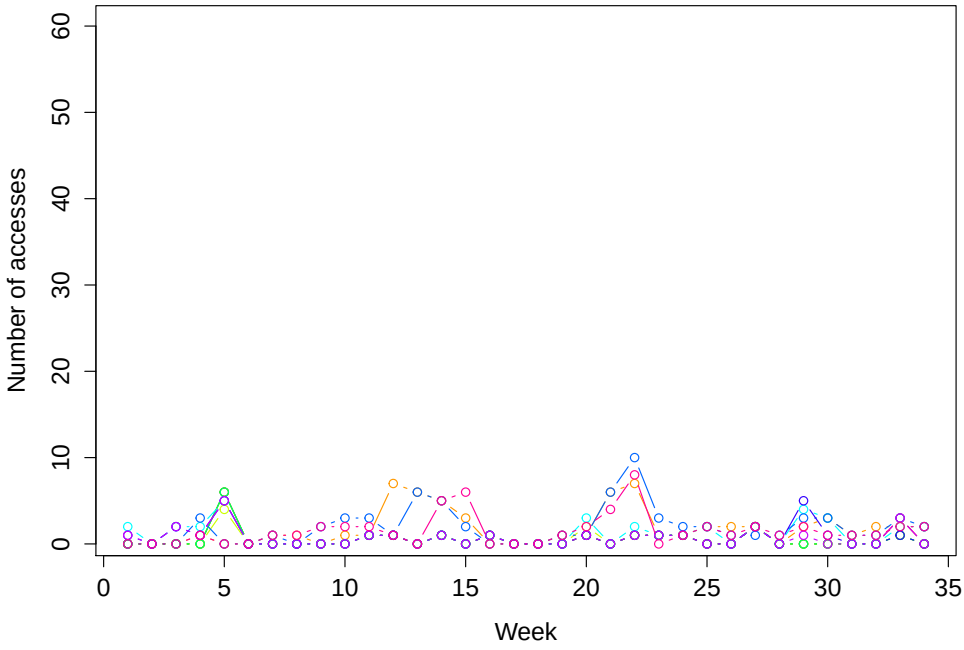


Figure 5.6: Weekly accesses for all datasets in the data11_7TeV project and AOD datatype. Every line can represent one or more datasets.



of weeks 2 to 30 as input and then predicts week 31 as output. An Elman model [35] has been used in this evaluation, which is a partially recurrent neural network and especially useful for time series data, where the output should not only depend on the current pattern but also on previous patterns in the time series. The maximum iterations have been set to 100 and the error threshold to 3%, which have proven to give good results.

5.3.1 Evaluation Metric

The evaluation is done by comparing the predicted accesses for each algorithm to the actual accesses. The main metric is the root mean square (RMS) of the differences between the predicted and actual values. For two sets of actual and predicted accesses a_i and p_i of N values the RMS is defined as follows:

$$RMS = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (a_i - p_i)^2}$$

It gives the standard deviation of the difference between the predicted and the actual accesses and is a good measure of the accuracy of the prediction. It is scale dependent and therefore cannot be used to compare the prediction across different project / datatype but it can be used to compare the predictions of the same project / datatype with different algorithms.

5.3.2 Static Prediction

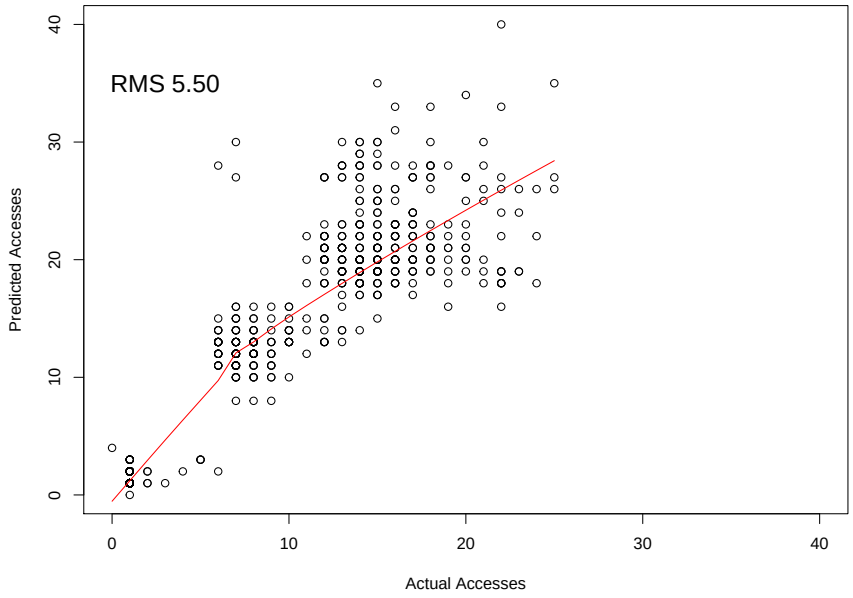
The first algorithm that is evaluated is the static prediction. As described already before this prediction method just takes the last available number of accesses and predicts this to be the accesses for the next week as well.

data12_8TeV project and NTUP_SMWZ datatype

Figure 5.7 shows the comparison of the actual to the predicted values.

- Overall the algorithm manages to predict the tendencies correctly, i.e., an actual popular dataset is also predicted to be popular in the prediction but the spread is big.
- This is also reflected when looking at the RMS of 5.5, i.e., on average the difference between the actual and predicted value is 5.5, which is big considering the biggest actual access is 27 and most accesses are around 6 to 20.

Figure 5.7: Actual vs predicted accesses for data12_8TeV project and NTUP_SMWZ datatype using static prediction. Each dot represents one or more datasets, and shows the predicted values compared to the actual value. The red line shows the trend of the prediction. A diagonal line would show a perfect trend, i.e., the predicted values match greatly with the actual values. In case the line goes above the diagonal the algorithm tends to overpredict, i.e., the predicted values are in general higher than the actual values and vice versa for a line lower than the diagonal.



- In the plot the effect of this can be seen, e.g., there are predicted accesses of 16 when the actual value is only 10.
- In general the prediction tends to over-predict the values.

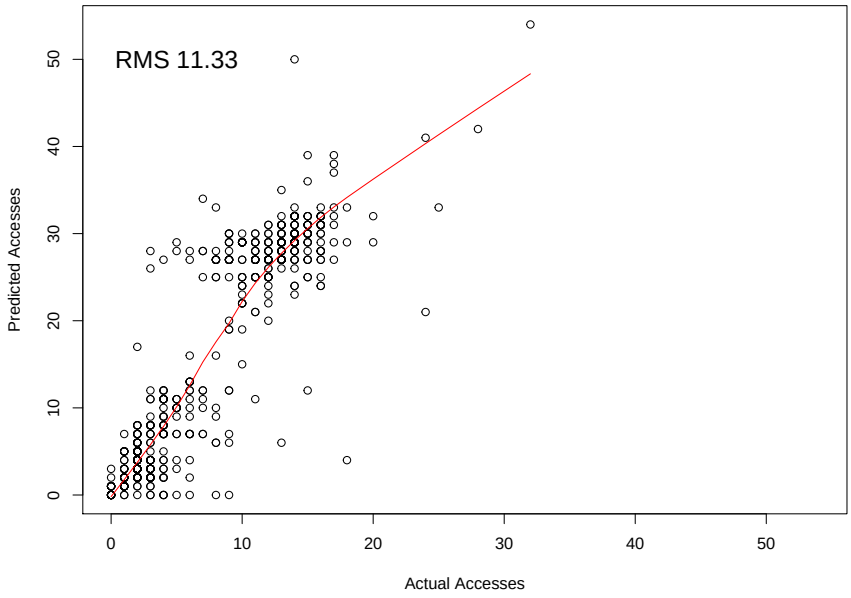
This can be understood when looking at access pattern in Figure 5.1. Overall the accesses are going down from week 30 to week 31 and as the static prediction is taking the previous value the predicted value will be generally too high.

mc12_8TeV project and NTUP_SMWZ datatype

For this project datatype combination the results are shown in Figure 5.8.

- Overall the prediction is worse compared to the previous combination.

Figure 5.8: Actual vs predicted accesses for mc12_8TeV project and NTUP_SMWZ datatype using static prediction.



- The RMS cannot directly be compared to the one before but the scale is not a lot different with a maximum value of 32 and most accesses between 0 and 20.
- Taking this into account the RMS of 11.33 is even worse than in the previous example.
- Overall the over-prediction is worse than before. There are cases when the actual value is 10 but the predicted value is 30.

Also here, when looking at the plot in Figure 5.3, showing the access patterns of the datasets that are popular around week 30, a drop from week 30 to week 31 can be observed, which then again results in over-predicting.

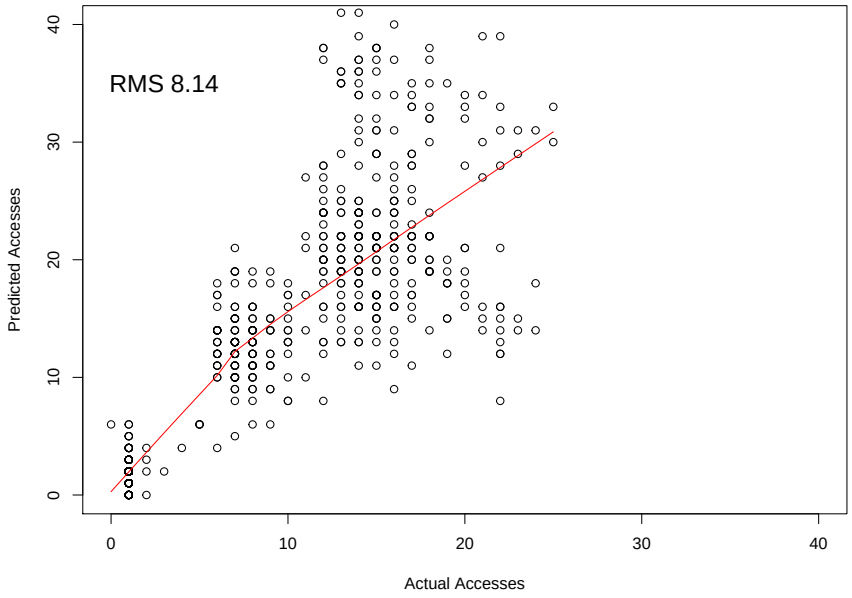
5.3.3 Linear Prediction

The next evaluated algorithm is a linear prediction, which just extrapolates the value based on the last two values.

data12_8TeV project and NTUP_SMWZ datatype

Figure 5.9 shows the comparison of the actual to the predicted values.

Figure 5.9: Actual vs predicted accesses for data12_8TeV project and NTUP_SMWZ datatype using linear prediction.



- Here the linear prediction behaves similarly to the static prediction as it generally also over-predicts the value.
- But this time it is worse, as here the RMS value 8.14 it is even higher than for the static prediction.
- Especially for the higher actual accesses the predicted accesses are spread a lot.

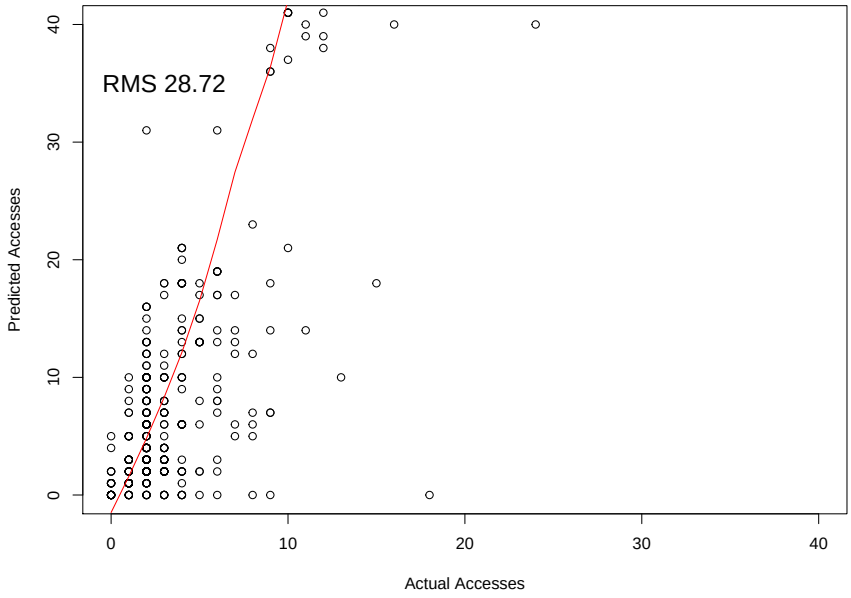
When looking at the accesses on week 29 and 30 in Figure 5.1 it can be seen that there is a slight increase from one week to the other resulting in even higher predicted values compared just taking the value in week 30. Therefore, the linear prediction tends to over-predict the values even more here.

mc12_8TeV project and NTUP_SMWZ datatype

The results for this combination are in Figure 5.10

- It can be seen that the linear prediction does not work at all for this combination.
- The RMS of 28.72 alone is already indicating how far the off the predictions are from the actual values

Figure 5.10: Actual vs predicted accesses for mc12_8TeV project and NTUP_SMWZ datatype using linear prediction.



- Looking at the distribution of the points in the plot it can be seen that it is greatly over-predicting for a lot of datasets.

Again as before, when looking at the access patterns in Figure 5.3 there is a big increase between week 29 and week 30. Extrapolating from there leads to very high number of accesses whereas for the actual accesses the values go down to week 31, which then results in this huge over-prediction.

5.3.4 Neural Networks

After the evaluation of the two simple methods now follows the neural network approach without any pre-filtering. As explained before the network is trained using the dataset accesses per dataset of between week 1 and 29 and then the accesses between week 2 and 30 are used to actually predict the 31st week. Every evaluation figure is split into three different plots. First the result of the trained neural network using the training set as input followed by the test set and the last plot is the result of the evaluation set, which then can be compared to the output of the other two prediction methods.

data12_8TeV project and NTUP_SMWZ datatype

The results for the first combination can be found in Figure 5.11.

- The first plot is showing the training set and it can be seen the model could very well converge on this data. The RMS is very low with only 0.48 and also the plot shows that there are only very small deviations.
- Similarly for the test set with an only slightly higher RMS of 0.49 and generally the trend is very good.
- The most important part is the evaluation set and here the error is a bit higher but better than the other models.
- Compared to the static prediction it is already down to half with an RMS of 2.65. And it is much better than the linear prediction.
- It has problems with the low accesses. For the accesses between 0 and 5 it over-predicts and for the accesses between 5 and 10 it is under-predicting. This could be optimised with filtering of the data.

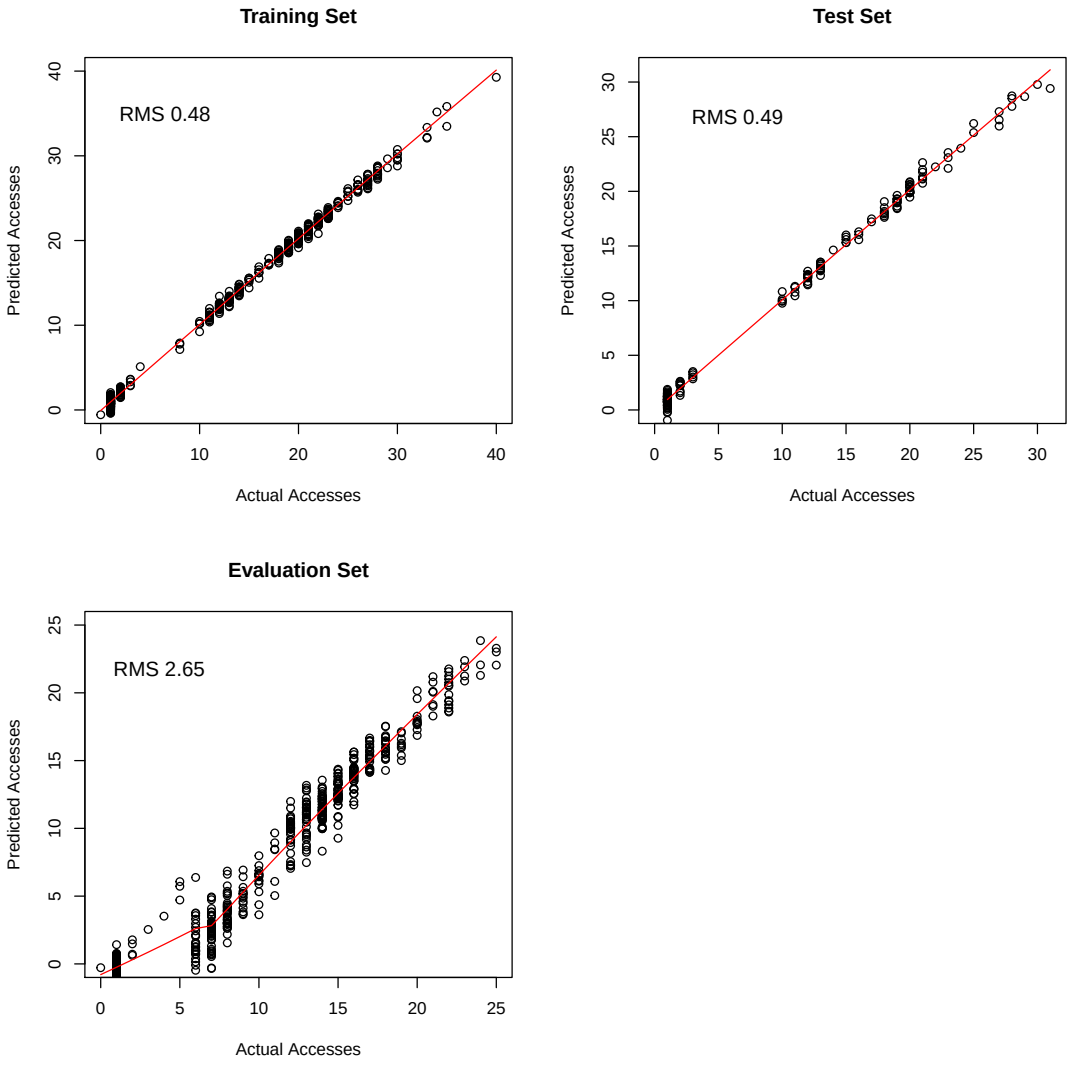
When looking at the accesses at Figure 5.1 that the for the popular datasets leading to week 31 are similar to previous patterns in the time series, which the neural network can pick up and apply in the prediction. For the unpopular data the pattern is not as obvious leading to a bigger error.

mc12_8TeV project and NTUP_SMWZ datatype

The outcome for this combination can be found in Figure 5.12.

- Overall the model has more problems in predicting the accesses for this combination.
- Looking at the first plot for the training set it can be seen that general trend is really good, but the single points are deviating. This is also reflected by the RMS of 2.94.
- For the test set is becoming worse with an RMS of 3.34 and also the trend is not as good anymore as it is over-predicting for the higher values.
- The prediction of the evaluation set is still better than for two simple prediction methods but with an RMS of 5.46 the prediction in generally too far deviating from the actual values.
- The trend shows that it is also generally under-predicting, i.e., it is affected by the unpopular data.

Figure 5.11: Actual vs predicted accesses for the training, test and evaluation set for data12_8TeV project and NTUP_SMWZ datatype using neural network prediction.



When looking at the patterns in Figure 5.2 and Figure 5.3 it can be seen that the popular datasets split in two groups. The ones that were very popular in the beginning of the period and the ones that only become popular at the end. The patterns for the latter ones between week 25 and 31 are similar to the other ones between week 10 and 16. But overall the spike is bigger and the drop is smaller. The model is trained on that and when applying this for predicting week 31 it under-predicts the values for the more popular datasets.

5.3.5 Summary

First, these examples show that the linear prediction does not work at all for these time series. Because of the non-linearity of the data it tends to greatly over-/under-predict the accesses.

The static prediction manages to generally get the correct trend, but especially for the higher accesses the results are deviating a lot.

The neural network examples show that the outcome of the prediction can vary a lot based on project/datatype combination. One of the reasons for the differences is the different project. Overall for the simulation (mc12_8TeV) data there are much more datasets available within the same datatype, because of the reason explained before, and this can lead to more different usage patterns within the same project / datatype combination. The network has more problems picking up all those different usage patterns and is also affected by unpopular data, which then leads to under-predicting. On the other hand for the detector data (data12_8TeV) is looks better as there are mainly two separate patterns in the data. One for the popular data and one for less used data and therefore the predictions are better. But also here there is some under-predicting caused by the unpopular data.

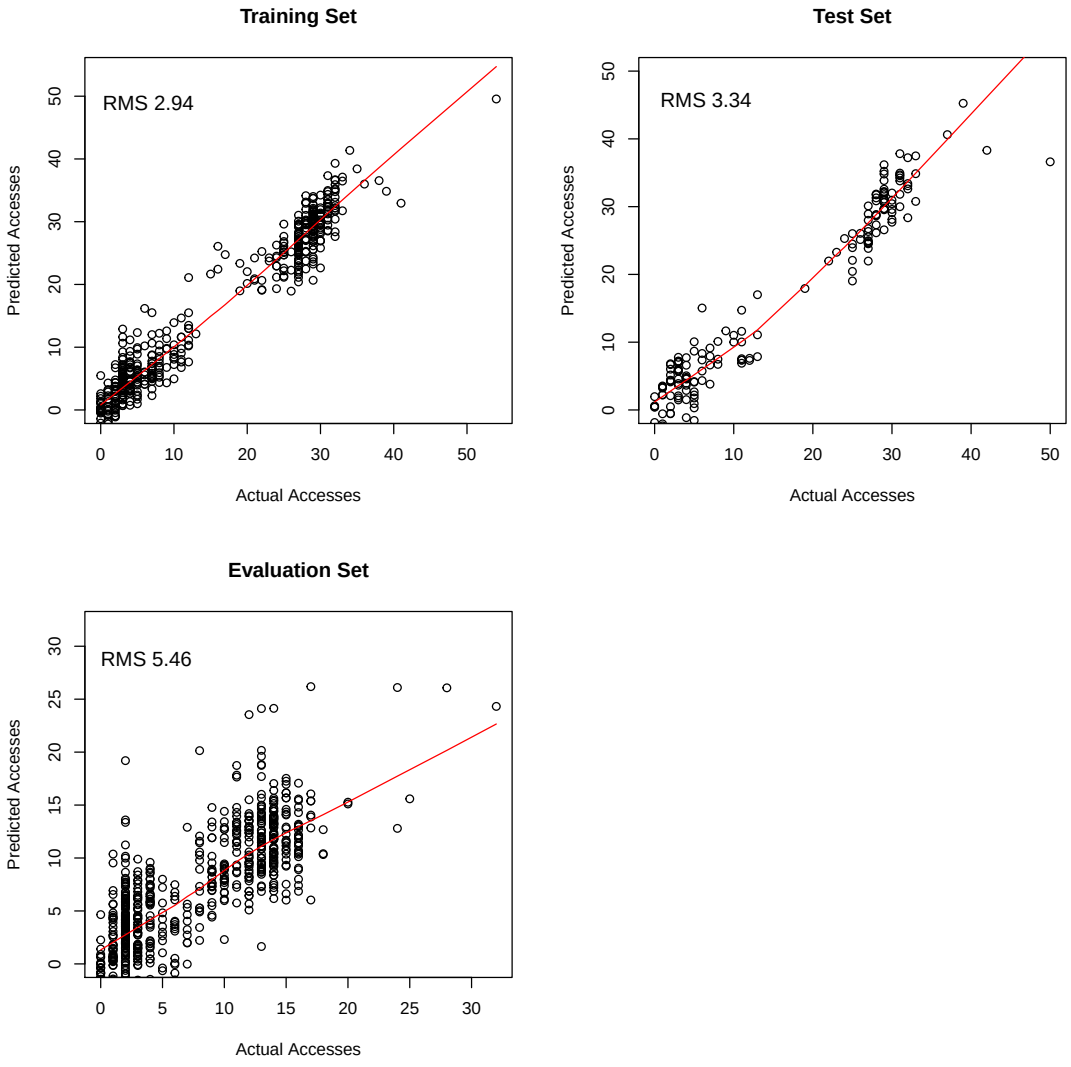
5.3.6 Hybrid Prediction

For the hybrid prediction the input data will be split into input data for the static prediction and for the neural network prediction. The filtering is done as explained in 4.5.1 before using both the access ratio and the total number of accesses. The values have been determined empirically.

data12_8TeV project and NTUP_SMWZ datatype

The results for this project / datatype combination can be found in Figure 5.13.

Figure 5.12: Actual vs predicted accesses for the training, test and evaluation set for mc12_8TeV project and NTUP_SMWZ datatype using neural network prediction.



- As the previous evaluation showed the neural network model had problems with the unpopular data, so the pre-filtering for this combination is done by removing the base line of less popular data.
- First, all DDM datasets that have not been accessed in more than 50% of the weeks are removed.
- Then, all DDM datasets that have less than on average 5 accesses in over all 29 weeks are removed from the input for the neural network training.
- Looking at the first plot for the training set it can be seen that the filtering removes everything below 8 accesses on week 30 and the result is very good with an RMS of only 0.36 and a very good trend.
- For the test set it looks similar with everything below 10 accesses removed and an RMS of only 0.39.
- For the important evaluation set the results are also much better than before. The RMS is down to only 0.84 and especially for the higher values the prediction is very close.
- For the low access the prediction is not as good but still acceptable.

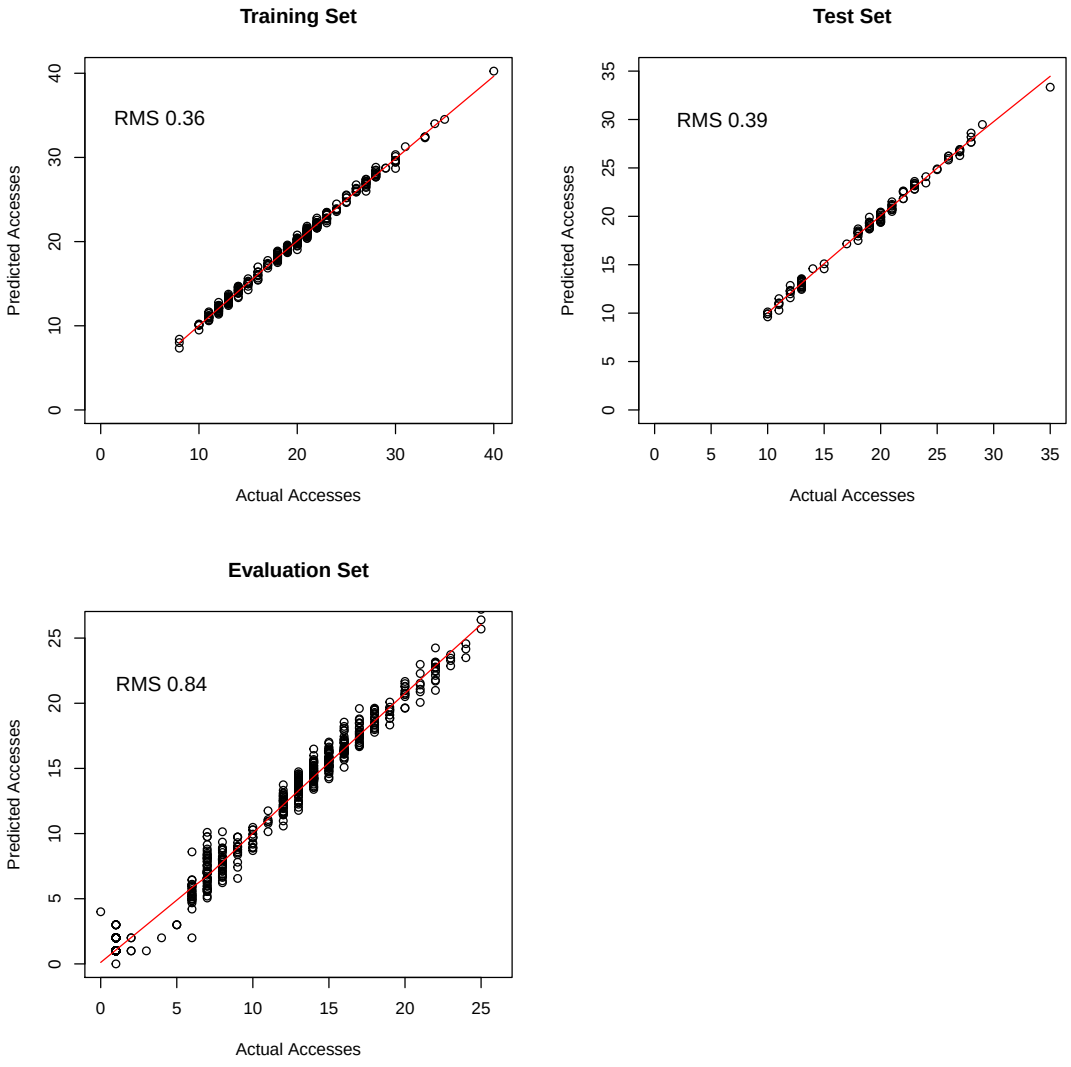
For this case, removing the unpopular data help training the model a lot. Overall the predicted accesses with the static prediction are not as good as the prediction for the more popular data. But the latter benefited a lot as now the model is not affected anymore as much by the unpopular data. Overall these results are very good as input for the redistribution, as there the accuracy for the unpopular data is not as important as for the popular data.

mc12_8TeV project and NTUP_SMWZ datatype

The outcome for this combination can be found in Figure 5.14.

- The first step here in pre-filtering is again to remove everything which has been accessed in less then 50% of the weeks.
- Then everything with an average number of 5 accesses in all 29 weeks are removed.
- As a last step in this case, also extreme outlier-datasets, which have at any point more than 50 accesses per week are removed. This removes the datasets that have the big spikes around week 15.

Figure 5.13: Actual vs predicted accesses for the training, test and evaluation set for data12_8TeV project and NTUP_SMWZ datatype using hybrid prediction.



- Now the model works much better for the training set shown in the first plot. The trend is very good the point do not deviate a lot with a small RMS of only 0.46.
- Also the test set is only slightly worse with an RMS of 0.79.
- The actual prediction of the evaluation set is a lot better than for the unfiltered neural network. Now the trend is very good and the predictions are overall much closer the actual accesses. The RMS of 1.98 is still a bit high as there are a few outliers, but overall it is much better than before.

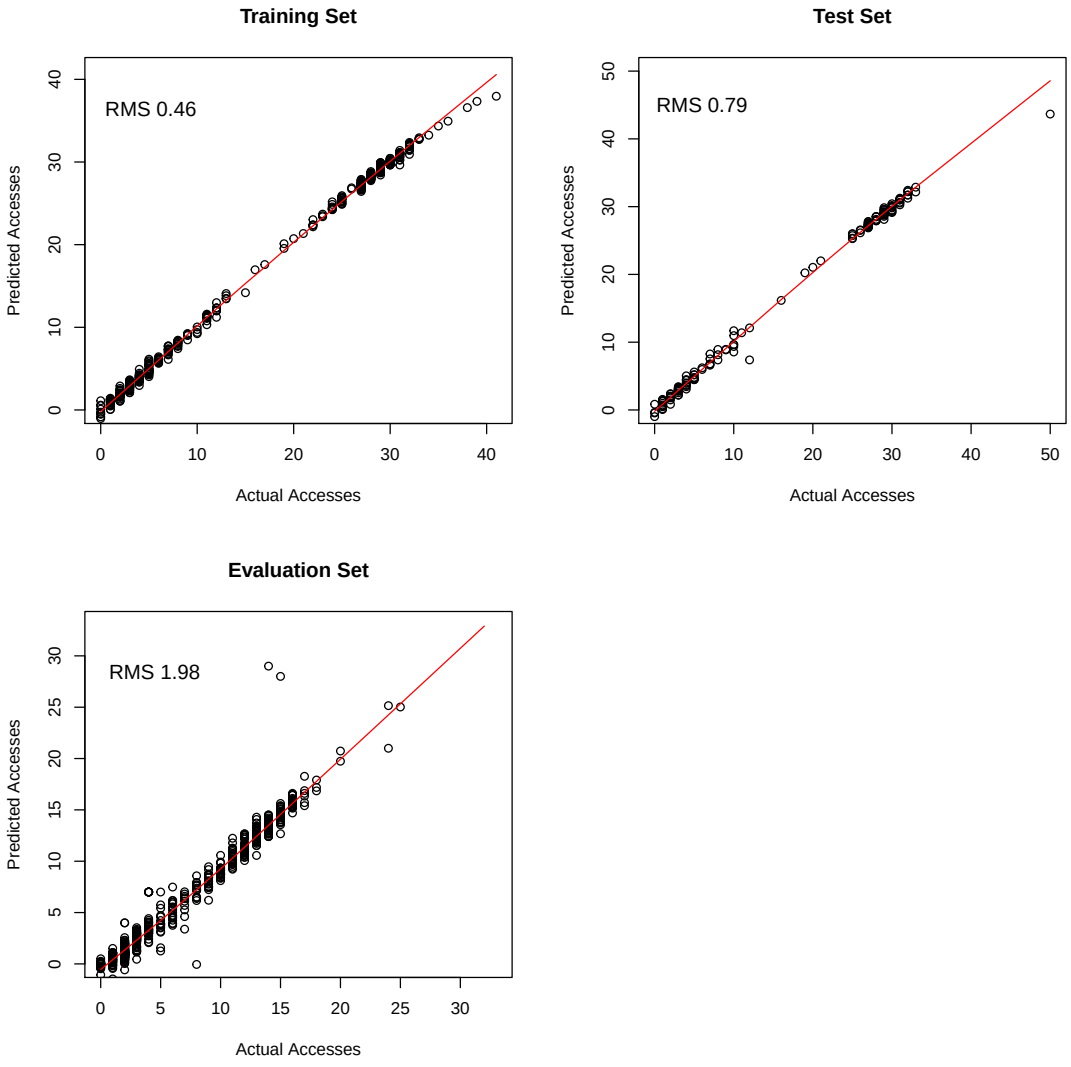
Overall the two criteria of the access ratio and the average accesses alone did not remove the most important part from the training set, as most dataset have been popular at some time. In this case removing the extreme outliers from the beginning of the period helped a lot, as now those do not affect the popular dataset from the end of the period anymore. On the other hand those outlier-dataset are not popular at the end of the period anymore and the static prediction works fine for them.

data12_8TeV and mc12_8TeV projects and NTUP_COMMON datatype

The last part of this chapter now evaluates the performance of the neural network prediction for the **data12_8TeV** and **mc12_8TeV** projects and the **NTUP_COMMON** datatype. Figures 5.15 and 5.16 show the results of the evaluation set for both combinations.

- As already discussed before and shown in Figures 5.4 and 5.5, overall there are less accesses for those datasets and the patterns are close together.
- For both projects the difference between the least used and the most used data is low.
- Because of this, filtering in this case is not necessary, as the bias in this case would not be a problem, and the input data has been directly fed to the neural network for training.
- The results for the **data12_8TeV** project show that the neural network gets the trend completely right. The predictions spread a bit but overall the RMS of 0.84 is good considering that the accesses range up to 15. There are a few outliers that are completely over-predicted but generally the results are good.
- For **mc12_8TeV** the RMS is even lower with 0.37 but this is mostly due to the smaller range of accesses. Overall, also here the trend is very good and especially for the accesses starting at 5 the predictions are very close to the actual accesses.

Figure 5.14: Actual vs predicted accesses for the training, test and evaluation set for mc12_8TeV project and NTUP_SMWZ datatype using hybrid prediction.



5.4 Conclusion

This evaluation showed that it is possible to train neural networks that are able to predict the future accesses. The first important part is splitting the input data into several groups based on the project / datatype, as the evaluation showed, that there can be greatly different patterns across all DDM datasets. But the patterns are similar for the same project / datatype combination. Depending on the overall popularity of the data and the difference between unpopular and popular dataset it can help the networks a lot when the unpopular datasets are removed for the neural network prediction. This helps the networks in converging on the right patterns and the performance of the static prediction is generally good enough for the unpopular data.

For some of the data it is possible that there are still some outliers but in general the models get the general trend right. This is important for the redistribution, that will be introduced in the next part.

Figure 5.15: Neural network prediction for the data12_8TeV project and NTUP_COMMON datatype.

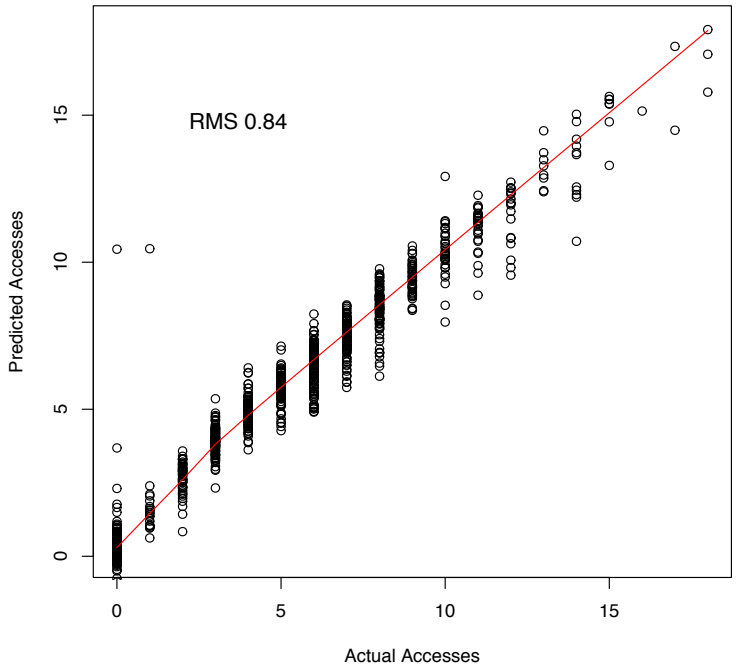
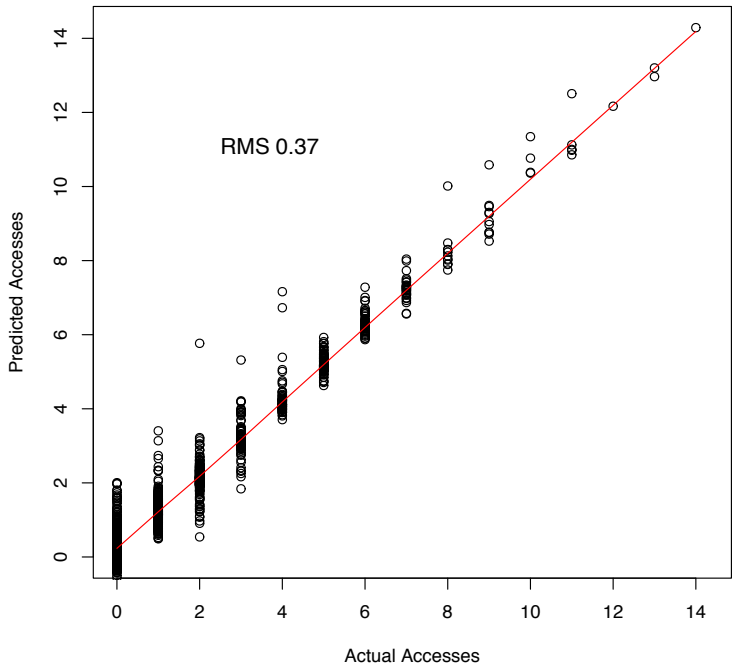


Figure 5.16: Neural network prediction for the mc12_8TeV project and NTUP_COMMON datatype.



Part III

Redistribution

Chapter 6

Data Redistribution

This chapter describes how the predicted accesses introduced before can be used in order to add and remove dataset replicas to improve the data distribution. The data redistribution is made of two parts that have to work hand in hand: the cleanup of space at sites and the creation of new replicas. The deletion procedure has to know exactly how many bytes it has to delete and the creation of replicas has to fill up the freed space as efficiently as possible. Furthermore, it has to work within certain constraints and a metric is needed to evaluate the performance. The chapter begins with an small example followed by requirements. After that the concept of the redistribution algorithm is introduced. The chapter finishes with details about the implementation.

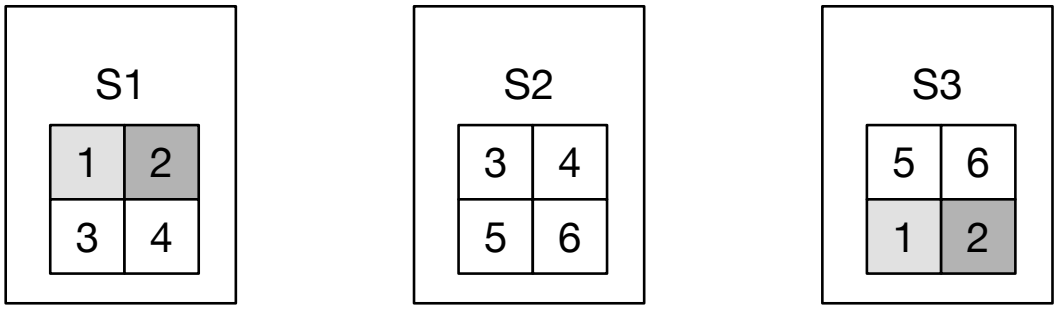
6.1 Introduction

In a distributed computing environment, where the job is sent to the data, good data distribution plays a crucial part in its performance. If the data distribution is not balanced according to available resources, e.g., all popular data is clustered at only a few sites, it is possible that computing resources lie idle, even when jobs are available, as the presence of the input data is a precondition for these jobs to run. This leads to unnecessary waiting times for users, which can be avoided when data is distributed according to its popularity. To illustrate the general problem this chapter begins with a simplified example.

6.1.1 Example

In this example there are three sites and six datasets as shown in Figure 6.1. The datasets are evenly distributed over the sites and for each dataset there are two replicas available. Each site has two job slots available. In the first step the workload is balanced and all

Figure 6.1: Site distribution example.



datasets are accessed equally, therefore the utilisation of all three sites is balanced. In step two workload starts to be biased towards two popular (1,2) datasets. Now the site utilisation changes and most of the workload goes to the sites S1 and S3. S2 is mostly empty. Now if new jobs arrive needing the popular datasets they will have to wait, even though there are jobs slots available on S2. The dataset replicas stored on S2 are mostly unpopular. In a dynamic data distribution environment this could be spotted and the popular datasets can be distributed more evenly over the sites to get a more balanced job distribution and therefore a lower average waiting time. E.g., by removing some of the unpopular data from S2 and transferring the popular data from the other sites to S2.

In general it can always come to bottlenecks when the number of replicas is smaller than the number of sites available. This is especially true for ATLAS, as there are around 100 sites, but in average there are much less replicas available. Therefore, optimising the data distribution can be very important to reduce possible hot spots.

6.2 Requirements

This section describes the requirements of the redistribution, starting with an overview of the data available to make decisions for the redistribution followed by the definition of the profit metrics, the constraints and the costs.

6.2.1 Input Data

There are four inputs available that the algorithm can use to make decisions about how to redistribute data:

- **Current Data Distribution:** This includes the location of replicas together with their size and creation date.
- **Available Computing Resources** The total number of job slots for each site.
- **Past Data Accesses:** For each dataset the summary of the last n weeks of user accesses per week.
- **Predicted Data Accesses:** For each dataset the predicted user accesses for the next week.

6.2.2 Profits

The redistribution of data yields mainly two main benefits which are closely related to each other:

- **Computing Resource Utilisation:** This is the ratio of running jobs to waiting jobs over time. In this case the ratio r at any time t is defined by the number of running jobs $j_r(t)$ and the number of waiting jobs $j_w(t)$ as follows:

$$r(t) = \begin{cases} j_r(t) & \text{if } j_w(t) = 0 \\ \frac{j_r(t)}{j_w(t)} & \text{if } j_w(t) > 0 \end{cases}$$

  If there are no waiting jobs the formula returns the number of running jobs. If there are waiting jobs than the ratio reduces. If the ratio is below 1 there are more waiting jobs than running jobs. This formula can be applied both to the whole grid or to single sites. In case of the whole grid $j_r(t)$ and $j_w(t)$ are the sum of those values for each single site in the grid.

  To now get the ratio over a period of time the average is used. The average ratio is then the sum of the instantaneous ratios at N different measuring points divided by the number of measurements:

$$U = \frac{1}{N} * \sum_{t=1}^{N} r(t)$$

  The higher this value the better.

- **Average Job Waiting Time:** This is correlated with the job ratio as the better the ratio, the less jobs have to wait and therefore the overall waiting time decreases. Each job has three timings in its lifetime: first the point when the job is submitted t_{submit}, next the point when the jobs actually starts t_{start}, and last the point when

the job finished t_{finish}. The waiting time is the interval between t_{submit} and t_{start}. The mean waiting time MWT for a set of N jobs is then defined as follows:

$$MWT = \frac{1}{N} * \sum_{i=1}^{N} (t_{i_{start}} - t_{i_{submit}})$$

6.2.3 Constraints

The algorithm has to work inside certain bounds that are either defined by the resource management policy or given by physical limits: Those constraints are:

- **Minimum amount of replicas:** Each dataset must have a minimum amount of replicas. The actual amount is based on a policy defined by the resource management policy. The algorithm must not delete replicas of a dataset when the minimum is reached.
- **Storage Space:** The total storage space for redistribution is limited and is defined as the sum of the individual storage spaces of each site of the grid. The algorithm must not use more than the maximum bytes available per storage endpoint.
- **Age of replicas:** To avoid the deletion of newly created replicas the algorithm must not delete any replicas younger than a certain age. The actual value can be set as needed.

6.2.4 Costs

The main cost for the algorithm is the amount of data that has to be moved. Each byte that has to be transferred between sites has a price. To take this into account the algorithm should be able to limit the redistributed data volume, i.e., the total number of bytes that are created and deleted.

6.3 Redistribution Algorithm

This section describes the actual algorithm that is based on the requirements described above and that is responsible for redistributing data based on the predicted accesses. Redistribution means that the datasets are moved between sites, which implies that datasets have to be transferred between sites and also deleted if additional space is needed. So the algorithm consists of two parts that have to work hand in hand: The cleanup of space at sites and the creation of new replicas. The deletion only needs to

run if space has to be freed and needs to know the exact amount of bytes it has to clean up and the creation of replicas has to fill this freed space as efficiently as possible. As described above the algorithm has work inside the constraints. It must not delete more than the minimum amount of replicas per dataset and has to add new replicas in a way so that future jobs can benefit from that.

6.3.1 Replica Creation

The first part of the redistribution algorithm is the creation of new replicas. It relies on the accesses predicted by the prediction algorithm. The general approach is to distribute the replicas evenly over the sites taking into account the possible accesses and the available computing resources. The first step is to assess the current data distribution and how it would handle the predicted workload. The metric for this is the accumulated accesses per site normalised by the total number of jobs slots at a site:

$$A_{site} = \frac{1}{s} * \sum_{i=1}^{n} a_i$$

Where s is the number of jobs slots at a site, n is the number of replicas at a site and a_i is the predicted number of accesses for a corresponding dataset. A_{site} then is the accumulated normalised number of accesses for one site. This metric approximates how much load a site is expected to have. The goal of the redistribution now is to balance this value for all sites, so that they have a similar predicted workload and therefore the WMS will have more choices when deciding where to schedule a job.

After the first calculation of the accumulated accesses the algorithm will begin to decide where to put new replicas. For this it will use the list of datasets predicted to be popular. It will work off this list one by one starting from the most popular to the least popular one. For each dataset it will take the site with lowest predicted workload (smallest A_{site}) and it will try to place a new replica at that site as long as there is not already a replica of this dataset there. If no space is available it will ask the deletion part to free up the space. If that is not possible it will continue with the next lowest workload until it finds a suitable site. If no site can be found this dataset will be ignored and it will continue with the next dataset. After the new replica has been added to the site A_{site} will be recalculated now counting in the predicted accesses of this new replica. After this the algorithm continues with the next dataset from the list.

6.3.2 Replica Deletion

The second part is the reduction of dataset replicas. The algorithm runs separately for each storage endpoint (SE). The only input for this algorithm is the number of bytes that are needed to be cleaned at a specific site. The algorithm starts by creating a list of all replicas on the SE. In the first run it filters out all replicas for which the corresponding dataset has recorded accesses in the last ten weeks. Those ten weeks give a good measure if a dataset will be accessed or not. It usually takes a few weeks after the creation of dataset until it will be first accessed. If there is still no access to the dataset in ten weeks time, then it becomes more unlikely that it will be accessed frequently in the future. But even if it would be accessed the algorithm keeps a minimum set of replicas and if it becomes popular there can be more replicas created again in the future.

The remaining replicas are then sorted by their size starting with the biggest. Then the actual deletion starts. The list is iterated and the replicas are deleted until the needed space is freed. Before a replica is actually deleted first it is checked that a minimum number of replicas remain on the grid. The actual number of minimal replicas is different depending on the data type and project of the dataset and is defined externally. If at some point no more replicas can be removed the list of replicas will be extended to also include the ones which have been accessed in the last 10 weeks with the exception of replicas that are younger than the minimum age of two weeks.

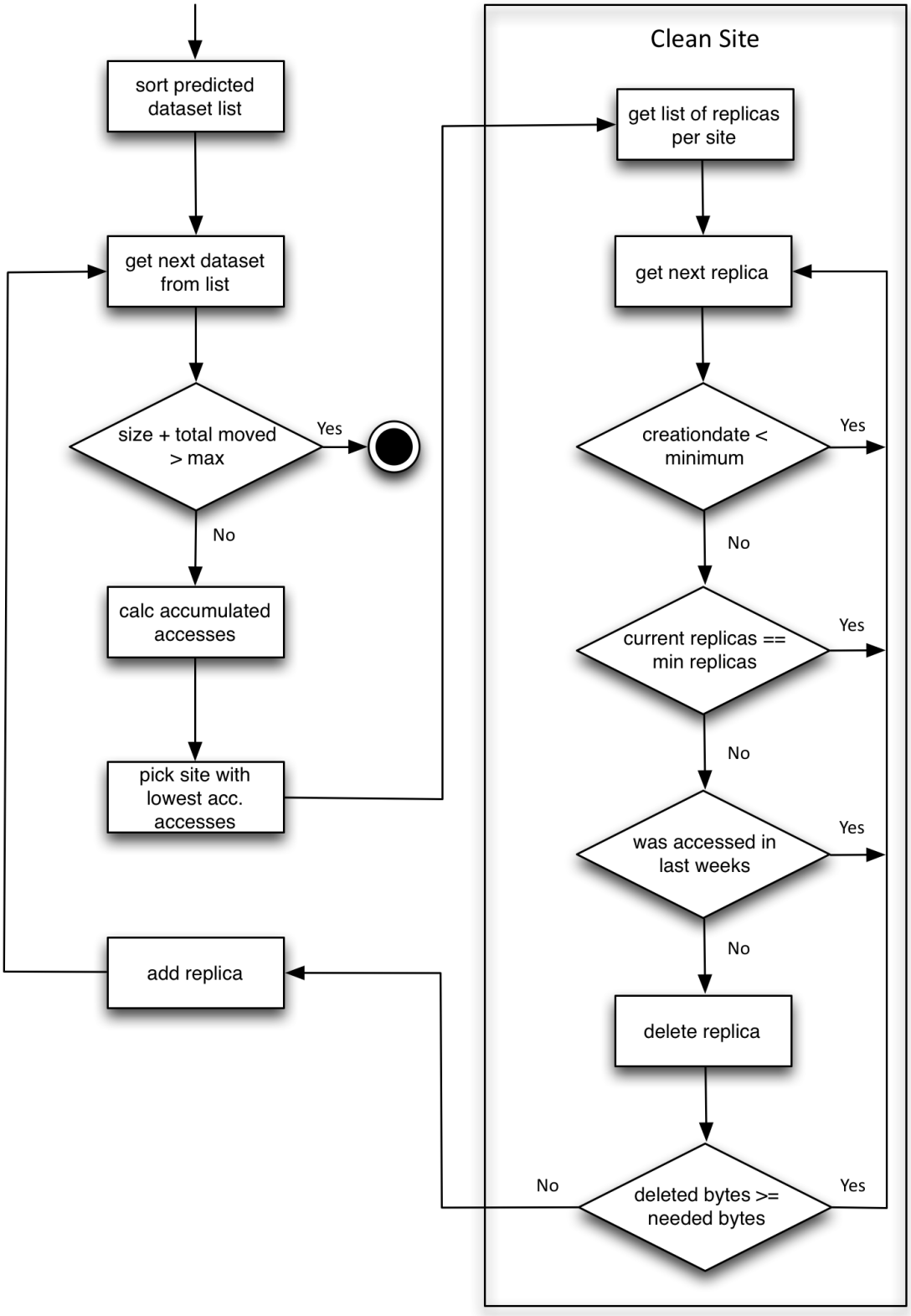
6.3.3 Combination

The last step is now to combine the two separate algorithms to get the full redistribution algorithm. The full process is illustrated in Figure 6.2. It starts with the list of predicted to be accessed datasets. It works off the list one by one as described before. For each dataset it checks if space is available at the chosen site and if not it runs the replica deletion algorithm to free up space. If not enough space can be freed it will continue with the next sites until it finds a suitable one. Following is a description of the inputs and the output of the algorithm.

Input

- **Current replica distribution:** A list of dataset replicas with their corresponding size and creation date per site.
- **List of predicted datasets:** A simple list containing the dataset name and the predicted number of accesses.

Figure 6.2: The complete redistribution workflow.



- **Site storage:** Total storage space per site in bytes.
- **Number of job slots per site:** The number of total jobs slot for each site.

Output

- **List of replicas:** A list of datasets and storage endpoints together with the required operation (add, remove).

6.3.4 Example

Following a simplified example to illustrate the algorithm. It shows how the algorithm decides where to replicate data and describes with a sample workload how this affects waiting time.

Setup

The setup of this example consists of three sites with different storage spaces and job slots as described in Table 6.1. Furthermore there are 5 datasets with different accesses as shown in Table 6.2.

Table 6.1: Site configuration.

Site	Storage Space	Job Slots
S_1	3	5
S_2	2	2
S_3	4	3

Table 6.2: Datasets.

Dataset	Size	Accesses
A	1	15
B	1	3
C	1	0
D	1	10
E	1	5

Initial Distribution

The initial distribution is given in Figure 6.3. The workload is described below:

1. In the first step 5 different jobs are submitted. 3 for dataset A, 2 for D and 1 for E. The first 3 will be sent to S_1 leaving 2 open job slots. The 2 for D will be sent to S_3 because it has more free resources compared to S_1 and the jobs for E will be sent to S_2
2. In the next step 5 new jobs come in for A and one for B. The jobs for A will be sent again to S_1 where 2 jobs can be started directly and 3 jobs have to wait in the queue. The job for B will be sent S_2.

Table 6.3: Initial Distribution.

Site	Datasets
S_1	A, C, D
S_2	B, E
S_3	B, C, D, E

After these two steps the job slots of all three sites are mostly filled leaving only one free space on S_3. This state is not optimal as on S_1 jobs are waiting in queue where at least one of these jobs could run on S_3 if it would have a suitable replica. The redistribution will fix this as described below.

Redistribution

1. Accumulated accesses: The redistribution begins with the calculation of the accumulated accesses. Site S_1 has replicas for dataset A, C and D with predicted accesses 15, 0 and 10 respectively. This results in overall 25 possible accesses which are divided by the number of job slots to get the normalised accumulated accesses $A_{S_1} = 5$. This is done for S_2 and S_3 as well resulting in $A_{S_2} = 4$ and $A_{S_3} = 6$.

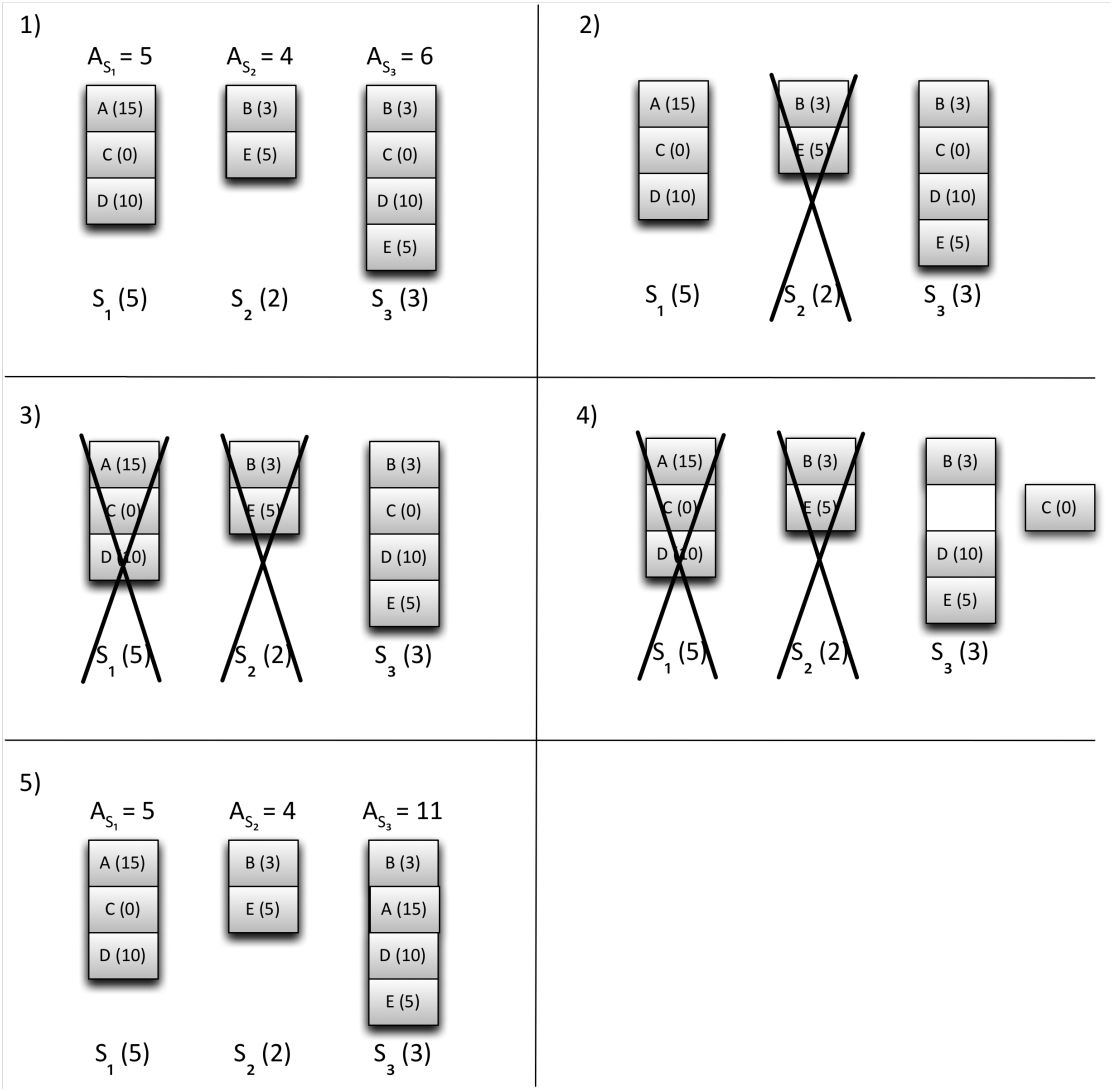
2. Try cleaning 1st site (S_2): The algorithm will now select the site with the lowest accumulated accesses, which is S_2 in this case, and will try to free up space there. Site S_2 has replicas for datasets B and E which both have predicted accesses for the next week so they will not be considered for cleaning resulting in no cleaning for this site.

3. Try cleaning 2nd site (S_1): The first could not be cleaned so the algorithm will continue with the site with the next lowest accumulated accesses which is S_1. This can also not be used because it already has a replica for dataset A.

4. Try cleaning 3rd site (S_3): This leaves only S_3. This site has a replica for C which does not have any predicted accesses. Furthermore C still has another replica at S_1 which makes it a suitable candidate for deletion. This frees up the needed space at S_3.

5. Adding new replica: The last step now is to add the new replica to S_3 and updating the accumulated accesses resulting in A_{S_3}

Figure 6.3: The redistribution example.



Final Distribution

The final distribution after the algorithm has run can again be found at Figure 6.3. The same workload runs again, but now the jobs can be distributed differently:

1. At first there are again 5 jobs as before. The difference starts already here: 2 jobs for A will be sent to S_1 and 1 job will be sent to S_3. The 2 jobs for D will be split between S_1 and S_3 and the one job for E will be sent to S_2.
2. Now 3 of the 5 jobs for A can be distributed over the sites. 2 jobs to S_1 and 1 to S_3. The jobs for B go to S_2.

So in the end there are only two jobs left waiting and all the sites resources are used.

This was only a very simple example to show how the algorithm works and how it can affect the job waiting time. The actual evaluation of the performance of the algorithm is done in Chapter 8.

6.4 Implementation

This section describes how the previously introduced algorithm is implemented. The implementation was done using Python together with an SQLite database. Code snippets of the most important parts can be found in the appendix in chapter 10.2. There are three main components involved:

- **Site:** The `Site` module keeps track of the currently used and the total space at a site. Furthermore it stores how many jobs slots the site has.
- **Data Catalogue (DDM):** The `DDM` module stores the location of all dataset replicas and their size. It provides an interface to query replicas for a dataset and also to get a list of all the replicas at a site.
- **Redistribution:** This is the main module which uses the other two modules to delete and create replicas.

6.4.1 Site Module

The `Site` module consists of one simple class that stores the storage space and the job queue size. One object of this class represents one site which can have multiple storage endpoints (DATADISK, LOCALGROUPDISK). The class has two simple methods to

add and remove replicas, which only take the storage endpoint (SE) and the size of the replica. The site itself has no knowledge about what data is stored on it but only keeps track of how much space is currently used.

It has the following interface:

- `add_storage_endpoint(name, storage_space)`: This adds a new storage endpoint to a site with the given storage space in bytes.
- `list_storage_endpoint()`: Returns a list of available storage endpoints at this site.
- `set_job_queue_size(size)`: Sets the total amount of job slots for this site.
- `get_job_queue_size()`: Return the total amount of job slots for this site.
- `add_replica(storage_endpoint, size)`: Adds a new replica with the given size in bytes to the given SE, fails if no more space is available.
- `remove_replica(storage_endpoint, size)`: Removes a new replica with the given size in bytes at the given SE.

6.4.2 Data Catalogue

The `DDM` module stores the location of each replica. It uses a simple SQLite database with one table and the following columns:

- **Name:** The name of the dataset.
- **Size:** The size in bytes of the dataset.
- **Site:** The site the replica is stored.
- **SE:** The storage endpoint on the site where the replica is stored.
- **Creation Date:** The date this replica was created at.

The Data Catalogue module now provides an interface for requesting dataset locality and for adding and removing replicas from the catalogue.

- `add_replica(name, size, site, se, creationdate)`: Add a new replica for the given dataset to a storage endpoint at a site. Fails if replica already exists or if no space is available.

- `remove_replica(name, site, se)`: Remove a given replica from a storage endpoint at a site. Fails if the replica does not exist.
- `get_dataset_size(name)`: Returns the size of a given dataset.
- `get_replica_creationdate(name, site, se)`: Returns the creation date of a given replica.
- `list_replicas_for_dataset(name)`: Returns a list of site, storage endpoint tuples where replicas for this dataset exist.
- `list_replicas_for_site(site)`: Returns a list of all datasets that have a replica on any storage element of the given site.
- `list_replicas_for_se(site, se)`: Returns a list of dataset that have a replica at the given storage endpoint at a site.

6.4.3 Redistribution

The Redistribution module is the main module and uses both the `Site` and `DDM` to determine where to add and remove replicas. To set it up it needs the following input parameters:

- **Site Information:** A dictionary with one entry per site, including the number of jobs slots, and list of storage elements and their total storage space.
- **Data Distribution:** A list of dataset replicas with their name, size, site, storage endpoint and creationdate.
- **Historic Accesses:** A list of all dataset and their accesses for the last n weeks.
- **Predicted Accesses:** A sorted list of datasets and their predicted accesses. Sorted in descending order by number of accesses.

Main method

With this information the module can set up the required sites and data catalogue. After that there is one main method, together with two helper methods, that will be called to do the actual redistribution: `redistribute(max_volume, minimum_age, storage_types)`. This method takes three parameters:

- `max_volume`: maximum number of data to be added and removed in bytes.

- `minimum_age:` minimum age of replica to be eligible for deletion in days.
- `storage_types:` a the storage type that should be used during the redistribution (e.g., DATADISK, LOCALGROUPDISK).

The methods starts by looping over all the predicted datasets, beginning with the ones having the most accesses. It keeps a counter of how much data has been moved already and maintains a list of replicas that will be added and removed as a result of the redistribution. First, it checks if adding the new replica would exceed the maximum data volume, and if yes, it will continue with the next dataset. Then it generates a list of sites which is sorted based on the accumulated accesses. It loops over those sites and tries to free up the needed amount of storage space to add a new replica. If not enough space can be freed it will continue with the next site. If the cleaning was successful it will add the new replica to the catalogue and extend the list with the to be deleted and added replicas.

Helper Method (Accumulated Accesses)

The first helper method is `generate_sorted_sites()` which takes the storage type as parameter. The method loops over all available sites and first checks if the site has an eligible storage element. If yes, it retries a list of all replicas at the site. It then keeps a counter with the accumulated accesses and adds the predicted accesses for each replica on that site. When all accesses are summed up the results will be divided by the number of jobs slots at that site to normalise the accesses. As a last step all sites will be sorted based on the normalised accesses starting with the lowest one.

Helper Method (Clean Sites)

The second helper method is `clean_storage_endpoints()` with the following parameters:

- `site:` The name of the site to clean.
- `se:` The storage element on the site to clean.
- `needed_space:` The amount of data that has to be removed.
- `minimum_age:` The minimum age of replicas to be allowed to be deleted.

This method loops over all replicas on a storage element of a site. For each replica it checks if is older than the minimum age, if it has at least one replica more than the

minimum replicas necessary. At last it checks if it is included in the historic accesses, i.e., if it has been accessed in the last n weeks. If it passes all test it is added to the list of eligible replicas. Then in the next step these replicas are sorted by size starting with the biggest, so that less replicas have to be deleted. The algorithm loops over these replicas and deletes one by one, always checking if the required storage space has be freed up.

Chapter 7

Grid Simulator

This chapter introduces the grid simulator, which was designed and developed to evaluate the effects of the previously described data redistribution strategies.

7.1 Motivation

There has been a lot of interest to simulate the grid in the past, that lead to a wealth of grid simulators. Still, it has been decided to develop a new simulator for this work.

A variety of grid simulators have been developed, like Monarc[36], ChicSim[37], SimGrid[38], OptorSim[39], MicroGrid[40] and GridSim[41]. None of those simulator is applicable in this case for the following reasons:

- MicroGrid and SimGrid were developed to simulate computational grids. They model job workloads in a heterogeneous set of computing resources, such as might be found in a grid of HPC centres. However, the simulation of data management aspects was not considered in these programs.

- The Monarc project addressed data grids more directly, in order to explore different computing models for the Large Hadron Collider (LHC) experiments. It is able to evaluate transfer times and bandwidth usage for different computing models but it is missing the infrastructure for replica management and optimisation, which it is now clear that the LHC experiments need.

- The original GridSim was focused on resource modelling and application scheduling for parallel and distributed computing and had no capabilities for data management. However this changes with the extensions presented in [42]. The data management is file based and allows for a complex hierarchy of global, regional

and local file catalogs. While such flexibility extends the scope of the application, it is overly complex for the modelling of the ATLAS DDM system.

- OptorSim is another grid simulator which was originally developed for European DataGrid, but was also used to simulate the LHC computing grid (LCG). In the LCG case it was aimed at studying different replication strategies and how they would affect mean job time and the effective network usage. The model relies on autonomous agents for the replica optimisation, which is currently not the model used by ATLAS.

For those reasons it has been decided to write a new, simple simulator for the ATLAS grid to be able to directly measure the impact of different data distributions on the job waiting times and computing resource usage. The remainder of this chapter describes the requirements and architecture of this new simulator.

7.2 Requirements and Design

Instead of using a synthetic workload the simulator runs on the workload history extracted from the production system. In this way it is possible to measure the impact the redistribution would have had if it were used in the real system. So the simulator has to be able to run on any given workload that is taken from the real system, defined by a starting time, execution time and dataset.

Because the workload is taken from the real system, the data distribution has to be as well. The simulator will be given a data distribution at the start of the workload and must keep a catalogue of data location and storage space occupied. Furthermore, it needs to provide the possibility to update the catalogue, i.e., add and remove replicas, at any time to simulate transfers and deletions. There is only a limited amount of storage space available on each site, so it also has to ensure that these limits are not exceeded.

Next, it has to simulate the utilisation of each site's computing resources. Each site has only a certain amount of CPU slots that can be used by jobs. One job always takes one CPU slot and blocks it, i.e., during this time no other job can use this slot. This leads to the next requirement, that the simulator has to hold a queue of jobs that currently cannot run because all possible slots are taken. When slots becomes available again, one of the waiting jobs in the queue can use it.

As a next requirement it has to be able to simulate the job scheduling of the real system. Jobs will be injected into the simulation as defined by the workload. The scheduler then needs to find a suitable site to run each job and dispatch it there.

Another important requirement is the possibility to stop the simulation at any given time, while saving the current status of all the job slots, the waiting queues and the data distribution. Then it needs to be able to restart the simulation again from this point.

Finally the simulation needs to run significantly faster time than the workload being simulated. It will be used to run multiple weeks of real workload to be able to simulated different data distributions to find an optimal one. If each simulation run for one distribution would run in real time this would be virtually impossible to do. An acceptable run time would be a couple of hours.

7.3 Architecture / Components

This section will give an overview of the main components of the simulator and their interactions. Based on the architecture of the real system the simulation uses three main components: the **Sites**, the **Distributed Data Management System (DDM)** and **Workload Management System (WMS)**.

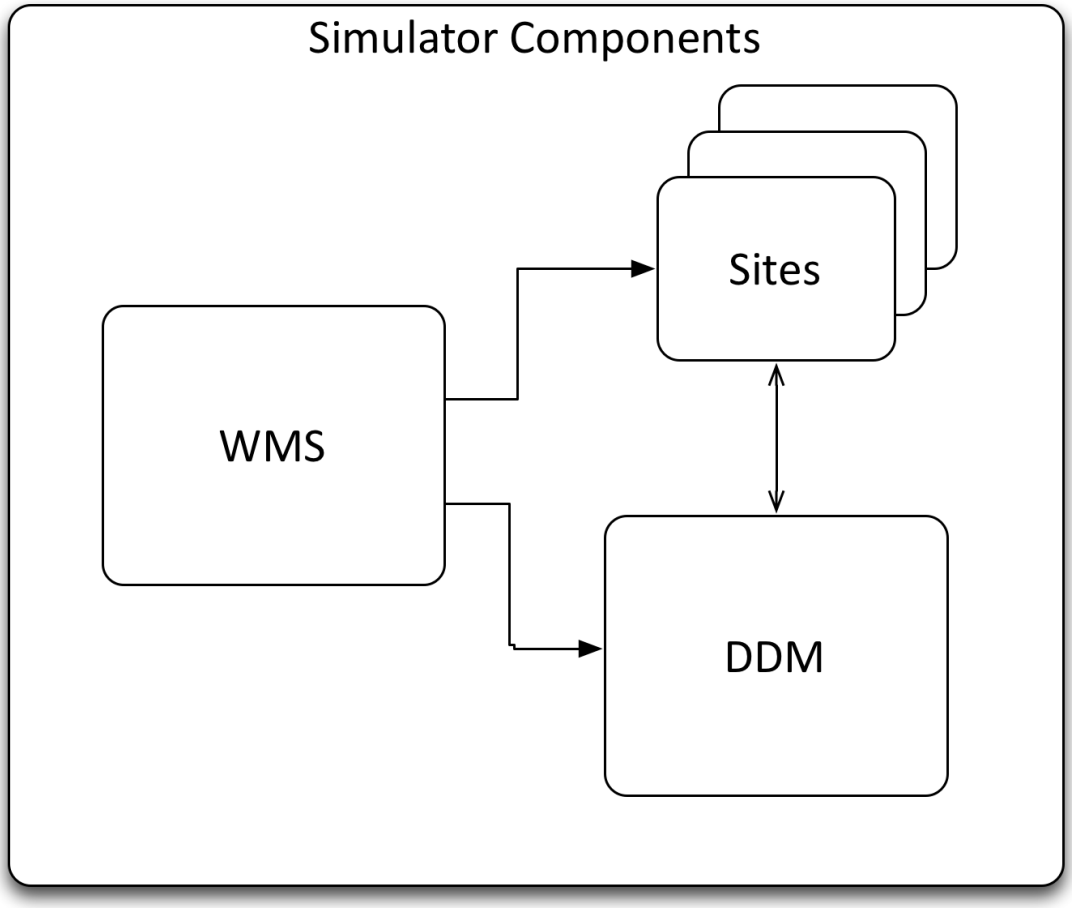
7.3.1 Sites

The site component has two main tasks: keeping track of the utilisation of site storage and the management of the computing resources. Like the real system it has different types of storage types, e.g., `DATADISK`, `GROUPDISK`, etc. and input datasets for jobs can be read from any of these disks.

Interface

- `add_replica(disk, size)`: Used to add a given number of bytes to a given disk. Will fail if not enough space is available.
- `remove_replica(disk, size)`: Removes the given number of bytes from a given disk.
- `run_job(execution_time)`: Runs a job for the given execution time.
- `get_running_jobs()`: Returns the number of currently running jobs.
- `get_capacity()`: Returns the total number of job slots.
- `get_waiting_jobs()`: Returns the current number of jobs waiting in the queue.

Figure 7.1: The simulation architecture.



7.3.2 DDM

The main functionality of the DDM component is to keep a catalogue of all replicas, their locality and their size. Interacts with sites to check if space is available.

Interface

- `add_replica(name, size, site, disk)`: Adds a new replica with the given name and size to the given site / disk.
- `remove_replica(name, site, disk)`: Removes the replica with the given name from the given site / disk.
- `get_replicas(name)`: Returns a list of sites / disks that have a replica for the dataset with the given name.

7.3.3 WMS

The WMS component manages the workload and schedules jobs at sites. It interacts with DDM to find the location of replicas and it interacts with the Sites to send the job for execution.

Interface

- `schedule_job(name, execution_time)`: Schedules a job for the given dataset name and the given execution time.

7.3.4 Inputs

Three inputs are needed to setup the simulation environment:

- **Site Configuration:** This defines the total storage space for the different space tokens and also defines the number of job slots.
- **Data Distribution:** Contains the location of all replicas at the start of the simulation.
- **Workload:** Consists of a list of jobs, where a job is a tuple containing the time the job will be sent to the WMS, the dataset it will use and the execution time.

- **Transfers and Deletions:** Consists of a list of datasets, their locations and the time during the simulation when they should be added or removed from the data catalogue.

7.3.5 Outputs

During the simulation two outputs are created:

- **Job Outputs:** Saves for each finished job the site it ran on, the time when it was sent to the WMS, the time when it finally started to run on the site, the execution time and the dataset.

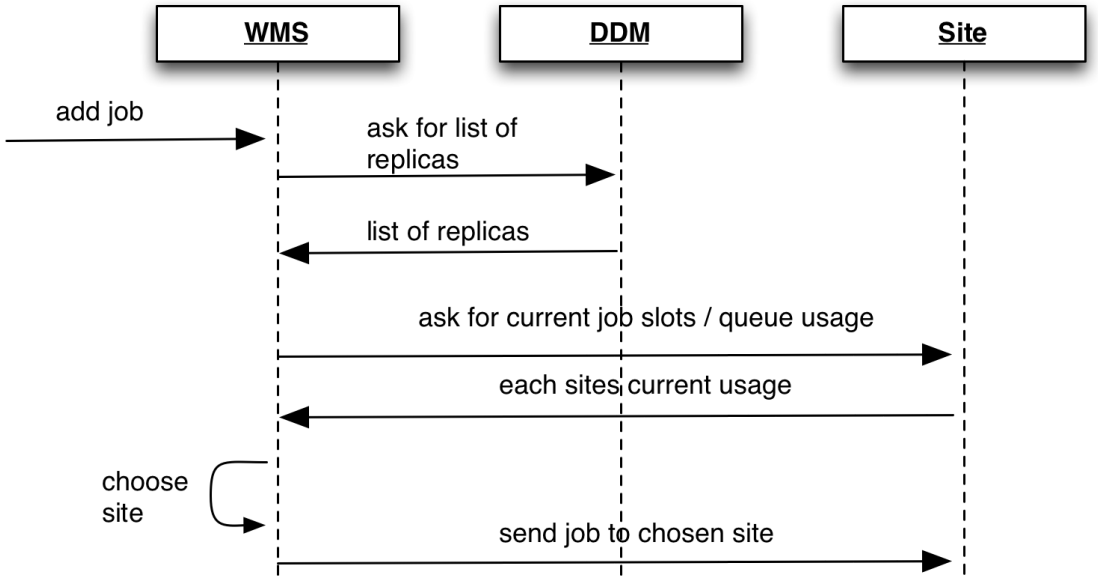
- **Site Utilisation:** Saves for every second of the simulation the number of totally used job slots, the size of the waiting queue, and the available job slots.

7.3.6 Workflow

This section describes the workflow and the interactions between the different system components.

The workflow starts by reading in the workload. The workload is a list of tuples where each tuple represents a job that has to be run by the system. A job is defined by the time it will be injected into the system, the dataset it will be using and the time it will be blocking the requested job slot. The simulation is a discrete-event simulation, i.e., it models all operation in the system as a discrete sequence of events in time. Each event occurs at a particular time and will change the state of the system. Between the events no changes to the system occur. In this case it means that the simulator only performs an action when a job is injected into the system or when it is starting or ending. The simulator will work through the list of jobs. Each job is sent to the WMS where it is scheduled for running. The WMS requests the list of sites that host a replica of the dataset requested by the job. Next, the WMS asks every site in the list about its current usage, i.e., job slots occupied, total number of job slots and number of waiting jobs. Based on this information it brokers the jobs to a site. To find a suitable site first it checks if any site has available job slots. If that is the case the job is brokered there. If no such site can be found the site with the shortest waiting queue will be taken. As sites have different resources the waiting queues are normalised by the total number of job slots, so that the workload will be evenly distributed over sites. In the simulation the jobs do not do any real work but block a virtual resource for a given amount of time.

Figure 7.2: The simulation workflow.



During this time the resource cannot be used by any other job. After the job is finished it releases the resource and the next waiting job, if there is any, can take it.

7.4 Implementation

This section describes how the simulator actually is implemented, i.e., which libraries where used, what is the input/output format, what is the mechanism to break and restart the simulation, which tweaks have been used to speed up the simulation. Code samples for the modules are included in the appendix in chapter 10.3.

7.4.1 Simulation Framework

The simulator is written in Python and built around the simple, yet powerful, SimPy[43] framework. SimPy is a process-based discrete-event simulation framework, which takes care of the management of so called Processes and Resources. Processes are active components in the simulation, like customers, vehicles or agents. The processes can access shared resources, like checkout counters, tunnels or servers, to model limited capacity points. The resources only have a limited capacity, i.e., if the number of processes accessing the resource exceeds the capacity, the process will block until one of the already holding processes give up the resource. Furthermore, SimPy has the functionality to either run a full simulation, until no more processes are available, or, simulate only until a certain point in time, which makes it perfect for debugging and, even more important,

for monitoring the status of resources and processes.

The simulation reuses and extends parts of the DDM and Site modules already implemented for the redistribution part.

Components

For this simulator only two components of SimPy are used:

- **Environment**
  The execution environment keeps track of all processes, resources and the simulation time. There are three main functions: `process(function)`, which is used to add new processes in the form of functions to the simulation, `timeout(delay)`, which suspends a process for a given delay and `run(until)`, which runs the simulation until a the given simulation time is reached.
- **Resource**
  The resource is initialised with its maximum capacity. Processes can call `request()` to ask for a usage slot and `release()` to eventually unblock.

7.4.2 Handling of Jobs and Sites

With these tools at hand it is now possible to model the sites and the jobs. A Site class already exists from the redistribution but until now only had the capability to handle how much data is stored at the different storage endpoints. For the simulation this part is not used anymore. The redistribution of data is only done before a simulation therefore the actual free or used space of a site is not relevant anymore. During the simulation the site only has to take care of the job slots and the queuing jobs. Therefore a SimPy resource will be added. The jobs can occupy this resource and it automatically takes care about queuing jobs. The capacity of the resource equals the maximum number of job slots per site.

A job is also modelled as a Python class. The object is initialised with the simulation environment, the site where the job should run and the execution time. The environment is needed to get the current timings in the simulation and to define a timeout with the given execution time for which the jobs is blocking the resource. The site is used to get access to the actual queue to request, block and release resources.

The job then has a `run` method, which is added to the simulation environment by using the `process`. During the runtime of the job it first retrieves the current simulation time and saves it as the time it was scheduled at. Next, it requests a slot in the queue. From

here the simulation framework takes over. If there is a free job slot available the job can continue and will again save the current time as the starting time. If nothing is available the environment will block this job until another jobs releases its slot. If multiple jobs are waiting then the longest waiting job will get the slot. After the job has blocked the resource for the given execution time it releases the slot again and for a last time requests the current simulation time and saves it as the finish time.

These three timings together with the used replica and site is written to a shared output file, which then can be used to calculate the average waiting time.

7.4.3 Data Catalogue

Before the jobs can be distributed over the sites a catalogue is needed that stores the locations of all replicas in the system. The data itself comes from the same database that is already used for the redistribution in Chapter 6.4.2. For the simulation the data has to be loaded into a special cache before the simulation starts. In the original database the records are stored for each replica with the files, the size and the storage endpoint. This works fine for the redistribution where the database has to be modified and single replicas have to be added and removed and overall the rate of queries is less, but in the simulation this will become a bottleneck. First, for the simulation the size and number of files are not relevant so they can be ignored. Furthermore, the scheduling algorithms has to be able to quickly lookup all available replica locations for one specific dataset and the rate of the queries are much higher. With the previously used database model it would first be necessary to query all replicas resulting in a couple of rows per dataset. From those results it would have to extract only storage endpoints. For the simulation this lookup is now only done once for each dataset before the simulation runs and stored in a separate cache dictionary. The dictionary has as keys the name of the dataset and as values only stores a list of sites where a replica is stored. So the scheduling algorithm can now simply use a quick key lookup in the dictionary and directly gets the list of possible sites.

7.4.4 Transfers and Deletions during a simulation

The simulator supports a primitive way to do transfers and deletions by simply adding and removing replicas at a given time during the simulation. The transfers are given to the simulator as input defined by a list of tuples. Each tuple has the following elements:

- **Transfer Time:** The time in seconds from the start of the simulation when the transfer/deletion will be applied.

- **Dataset:** The name of the added/deleted datasets.
- **Site:** The name of the site the dataset will be added to / deleted from.
- **Type:** Either transfer or deletion.

Those transfers and deletions will not permanently be added to the SQLite data catalogue during the simulation, which otherwise would result in a re-initialisation of the data cache. Instead they are added or removed directly from the cache. In case of a transfer the name of the site will be appended to the list of the given dataset in the cache. In case of a deletion it will be removed from the list.

7.4.5 Adding Workload to the System

The last missing piece now is the workload, that is defining the jobs, and a way to distribute them over the job slots at the sites. Similar to the transfers the workload is just a list of tuples where each tuple has the following elements:

- **Starting Time:** The time in seconds from the start of the simulation when this job will be sent into the system.
- **Dataset:** The name of the dataset the job will use.
- **User:** The name of the user that is running the job.
- **Execution Times:** A list of execution times in seconds.

One job defined in the workload can actually have multiple sub-jobs that use the same data but can run in parallel. Instead of having separate entries for each of those job the workload can have a list of execution times. This also helps the performance of the simulator as the replica lookup has to be done only once for this set of jobs instead separately for each one. In order to distribute the workload the simulator will go step by step through the simulation checking each time if an item in the workload has the current simulation time as starting time. The brokering is done as described in section 7.3.6. It searches for each sub-job for a suitable site and sends it there for execution.

7.4.6 Site Utilisation

As described in the previous section the simulator has two different outputs. The way the job output is created and stored is shown before. The second important output is the site utilisation. This is an overview of the total available job slots, the running jobs and the

queued jobs for all sites in the grid at several times during the simulation. To get those number a callback is registered with the simulation environment that is triggered every minute during the simulation. Then the simulation breaks and the function registered with the callback iterates over all available sites and collects the state of the job slots and waiting queues. There are two different ways how this data can be saved. First a total of all sites will be calculated and for each time the callback ran four values will be written to the output: the current simulation time, the maximum number of job slots, the used job slots and the queued jobs. The second option writes the same output but now separately for each site. This however creates a huge output and is only used for special cases when more detail is necessary for the evaluation.

7.4.7 Break and Restart Simulation

The simulation can run for a long time and sometimes only the effects on a special week are interesting. Because the queues fill up over time and therefore also affect the outcome of the simulation, it is important to run over the full simulation period. For this case in the simulator a mode to pause the simulation, change the data distribution and then restart the simulation was implemented. In this mode it is possible to let the simulation run until a certain number of steps. When the point is reached the status of the current simulation will be persisted on disk. To do this the simulator has to save the state, i.e., queued or running, of all jobs in all the queues of the sites and the current simulation time. Before the simulation is then started again, the simulator will put all saved jobs with the saved state in same queues again and also restore the simulation time.

7.4.8 Example

This last section shows a simple example to illustrate how the simulator works, how the inputs are used and which outputs are produced. In this example there are only two sites as defined in Table 7.1. They are only small sites were SiteA has only 4 and SiteB 3 job slots. There are three different datasets with overall four replicas as shown in Table 7.2. The first is exclusive to SiteA, the second to SiteB and the third dataset has a replica on both sites. Finally, Table 7.3 shows the workload of this simulated period. The simulation covers a period of six time units from 0 to 5. There are in total three jobs by two users with three sub-jobs each with varying execution times. Figure 7.3 shows for each step of the simulation how the jobs slots are occupied by the jobs and the queuing jobs, if any. The sub-jobs of the first job starting at 0 are directly started SiteA as the dataset is only available there and enough resources are available. At time 1 the next job is submitted which uses Dataset2. This one is only available at SiteB so the jobs are sent there. At the next step a job is submitted using Dataset3, which is available at both sites but only SiteA has one job slot available. So the first sub-job is directly started at SiteA and the other two jobs are split over both sites and the queue each site gets one job. At time 4 the first two sub-jobs of the first job are finished as well as two sub-jobs of the second job. So now the two waiting jobs can start running at SiteA and SiteB. At time 4 only those two are still running and at time 5 all jobs have finished. In Table 7.4 the output of each jobs is listed in the order it is written by the jobs when they finish. The first six rows in this table are the outputs of each sub-job of the first and the second job. The scheduled time is the same as the starting time as the jobs did not have to wait. Row 7 and 9 belong to the two sub-jobs of the third job, that had to wait in the queue. Here, the scheduled time and the starting are not the same and the difference between starting and scheduled gives the waiting time, which is 1 for both jobs. Finally, Table 7.5 shows the combined output of the site utilisation.

Table 7.1: Example Sites.

Site	Number of job slot
SiteA	4
SiteB	3

Table 7.2: Example Datasets.

Dataset	Replicas
Dataset1	SiteA
Dataset2	SiteB
Dataset3	SiteA, SiteB

Table 7.3: Example Workload.

Starting Time	Dataset	User	Execution Times
0	Dataset1	User1	3,4,3
1	Dataset2	User2	2,2,3
2	Dataset3	User1	3,1,2

Figure 7.3: Job slots and queues with example workload.

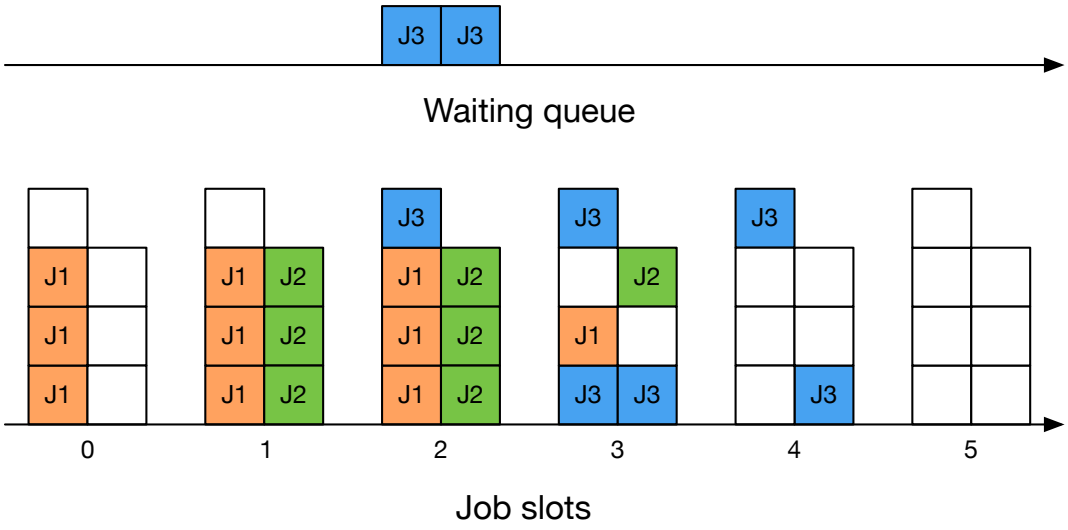


Table 7.4: Example Job Output.

Scheduled Time	Starting Time	Finished Time	Dataset	Site	User
0	0	3	Dataset1	SiteA	User1
0	0	3	Dataset1	SiteA	User1
1	1	3	Dataset2	SiteB	User2
1	1	3	Dataset2	SiteB	User2
0	0	4	Dataset1	SiteA	User1
1	1	4	Dataset2	SiteB	User2
2	3	4	Dataset3	SiteA	User1
2	2	5	Dataset3	SiteA	User1
2	3	5	Dataset3	SiteB	User2

Table 7.5: Example Site Output.

	0	1	2	3	4	5
total	7	7	7	7	7	7
running	3	6	7	5	2	0
waiting	0	0	2	0	0	0

7.4.9 Summary

With all the components introduced and explained in the last four chapters it is now possible to evaluate the effects of redistributing data. The popularity predictions can be used as input for the redistribution which then can be used as input for the simulation. The job and site utilisation output can then be used to calculate waiting times and examine the development of computing resources over time.

Chapter 8

Redistribution Evaluation

8.1 Introduction

This chapter now combines the previously introduced popularity prediction, the redistribution algorithm and the simulator and evaluates the effects of the redistribution on the user waiting times and the site utilisation. First, the setup of the simulation is detailed. Then, the redistribution is evaluated separately for three different weeks with different characteristics.

8.2 Simulation Setup

This first section describes the setup of the simulation that is used to evaluate the performance of the prediction based redistribution. To configure the simulation different sources have to be used. To setup the sites, information from PanDA, AGIS[1] and DQ2 needs to be combined to get the number of job slots, the storage space and the location of replicas. For the workload, the PanDA job archive and the DQ2 traces and data catalogue are used. The simulation used in this emulation is in many ways simplified compared to the real system, but those simplifications were necessary because of the complexity of the real system and should not compromise the results. Wherever those simplifications were applied they will be explained.

8.2.1 Sites Setup

The first parameters for the simulation are the configurations of the sites. For each site there are two different kinds of parameters: The total number of job slots and the total number of bytes per storage endpoint. It is hard to find the site configurations

[1] ATLAS Grid Information System `http://atlas-agis.cern.ch/agis/`

for a specific point in time. The total number of CPU cores per site cannot simply be taken as the resources can be dynamically allocated to production and analysis queues. It would be complicated and time consuming to collect the computing resources for every site for the whole period of the simulation. On top of that this is very difficult to do retrospectively for jobs running on the specific datasets used in this evaluation. So instead, for this evaluation, a simplified method has been used, where the maximum number of running jobs per site together with the workload have been used to estimate the number of job slot.

This has been done in several steps:

- First the number of running jobs per site has been extracted from the workload management system at multiple points per week. With that, the average number of running jobs per site can be computed which is then taken as the baseline number of slots for the simulation.
- Then, together with the workload the job slots per site can be scaled. If a week has a lower workload the number of available job slots per site will be scaled down by the same factor and vice versa.
- This is then tuned with the original workload until the simulated average waiting time for a specific week without any redistribution is within a small error margin compared to the actual recorded average waiting times for the job archive.

It has to be noted that the numbers of available job slots used in the simulation do not necessarily match the exact number in reality, but the order of magnitude will be preserved and the heterogeneous nature of the different computing sites will be correctly simulated.

8.2.2 Workload Setup

The workload has been extracted from the WMS archive and enriched with DDM information on a weekly basis. As with the sites this also was done is several steps:

- First, the job information has been extracted from the PanDA job archive. The archive includes the job id, the submission time, the job starting time and the execution time.
- The missing part is the input dataset and the user information. The PanDA server sends traces to the DQ2 system when it accesses a file in a dataset. The access

traces includes the job id, so both sources have to be joined to get the job timings as well as the used dataset information.

- The traces are on a file level so in the last step the DQ2 catalogue needs be used to resolve the information to dataset level.

The end result then consists of four different fields: the relative starting time in the simulation, the accessed dataset, the user and the execution time. This process is repeated for all weeks in the simulation interval.

8.2.3 Replica Catalogue

The replica catalogue used in this simulation is a simplified version of the catalogue used by DQ2. The database used is implemented in SQLite as described before and only contains the name, the creation date, the size and the storage element of the replicas. Those fields are extracted from the DQ2 database but only for the data* and mc* / AOD and NTUP* datasets, as those are the types used for user analysis and hence are the only ones used in this evaluation. This information is extracted at a given point in time, but as replicas are created and deleted over time, the catalogue changes constantly. To get the correct database condition at the simulated point in time the catalogue had to be adjusted. Therefore the transfer and deletion history is also extracted from DQ2 and is used to replay the changes in the catalogue.

8.2.4 Simulation Mode

After all the needed inputs and parameters have been collected the simulation runs as described in the simulation chapter. First, the site configuration is read and applied, then the replicas are read into the replica cache and finally the workload is read into memory. The simulation then steps through the workload and blocks and releases job slots according to the workload and the dataset locations. After a job has finished a log entry is written with the chosen site, waiting time and run time, which is then used for evaluation.

Overall a total period of 13 weeks of workload has been extracted from the database. Those 13 weeks include different weekly workloads. From those weeks 3 different weeks with different characteristics have been chosen, which will be described later and evaluated separately. For the evaluation only jobs that are submitted at the evaluated week are considered. This includes those which finished in a later week and it excludes jobs started in a previous week but finished in the evaluated week. This also means that the

simulation always runs over the complete 13 weeks. This is necessary because jobs from previous weeks can influence the waiting time for jobs in the evaluated week, as jobs submitted or started in a previous week can block resources for jobs submitted in the evaluated week.

8.2.5 Redistribution

The predicted accesses used for the redistribution are provided by the trained networks as explained in the prediction evaluation chapter. They are trained beforehand and then the trained networks are fed with all the available datasets in the data catalogue to get the predicted accesses for all datasets. The redistribution algorithm, as explained previously, is then applied with the predicted accesses. For each of the three evaluated weeks the algorithm is run for different redistributed data volumes to simulate how the system reacts with higher number of redistributed datasets and to find an optimal data volume. The volume is the amount of data that is newly added by the algorithm. The examined volumes are 100, 200, 500, 1000, 2000, 3000, 5000, 7500 and 10000TB. The volumes have been chosen like this as 100TB would only introduce a small load on the transfer system and is a good point to start whereas 10PB would already have a big impact and would mean redistribution of more than a third of all the data in this simulation. So, a big range of possible volumes will be tested.

The necessary deletion and transfers are always applied to the data catalogue before the simulation starts. Especially with higher data volumes this would put a lot of stress on the transfer system. Therefore, in the real system it would not be doable like this but should be spread over several smaller redistributions over the course of the week, but that would also require more fine-grained predictions as explained later.

8.3 Simulation Parameters

8.3.1 Workload

The workload was extracted from the PanDA job archive. It includes all user analysis jobs that are using datasets in the data* or mc* project and the AOD or NTUP* datatypes. The workload includes also jobs submitted by Hammercloud[2], which account for 10% of the jobs. They are included in the simulation as they run on the same data as user jobs

[2]Hammercloud is a distributed testing system. It submits small jobs to sites and it is used to perform basic site validation, help commission new sites, evaluate SW changes, compare site performances, etc. `https://twiki.cern.ch/twiki/bin/view/Main/HammerCloud`

Table 8.1: Workload Parameters (without Hammercloud jobs).

	Date	# of jobs	Mean runtime	Median runtime
Week 1	17.02. - 23.02.2014	2428732	3688	625
Week 2	31.03. - 05.04.2014	2879215	2309	379
Week 3	24.03. - 30.03.2014	2143319	3096	682

Table 8.2: Site parameters.

	Sites	Storage elements	DATADISK storage elements	Job slot
Week 1	69	248	68	24205
Week 2	69	248	68	18357
Week 3	69	248	68	18177

and using the same queues and therefore can effect them, but in general they only have a short run-time of around 7 minutes. In the evaluation of the waiting times they are not included.

Table 8.1 shows the parameters of each week without Hammercloud. Before going into the deeper analysis later it is immediately clear that there are gross differences between the three weeks. The first week has less jobs than the second one but the runtimes are smaller, which also results in noticably less waiting time for the second week. The third week on the other hand has the least jobs but the highest median runtimes, which results in slightly less, but similar, waiting times compared to the first week.

8.3.2 Sites

The maximum size of each storage endpoint has been set for this evaluation as the volume of data*/mc* AOD/NTUP* that are stored on that SE at the beginning of the week. i.e., the algorithm always first has to delete replicas before it can add new ones. This is a reasonable assumption as in the real system the storage elements are always kept close to maximum capacity.

The parameters for the sites are shown in Table 8.2.

Each site can have more than one storage element, which leads to the much higher number of SEs compared to sites. The workload management system can use any replica on any type of storage element as input for a job. On the other hand the redistribution algorithm only considers DATADISK, as those are the only centrally managed SEs, i.e., the algorithm has 68 SEs available for redistribution.

Table 8.3: Replica Parameters.

	# replicas	Size	# replicas (DATADISK)	Size (DATADISK)
Week 1	526511	42.3PB	210784	27.4PB
Week 2	535584	43PB	211798	27.5PB
Week 3	534838	43PB	211798	27.5PB

8.3.3 Replicas

The parameters shown in Table 8.3 represent the state of the data catalogue at the beginning of each week. They have been created by extracting all dataset replicas in the data* and mc* project and AOD and NTUP* datatype from DQ2 at a point in time after those weeks. Then, all transfers and deletions until the start of the week have been applied in reverse to get the starting distribution.

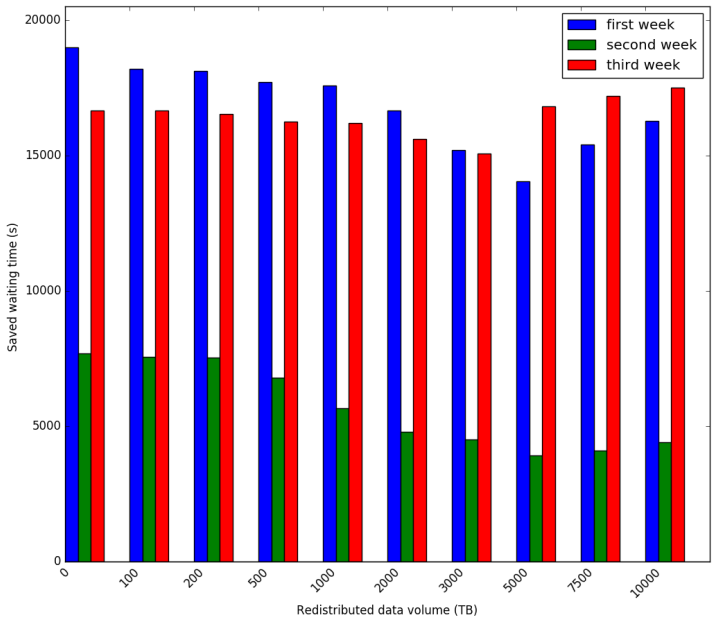
8.4 Results

In this section the results are discussed. The section first starts with an overview of the evaluation of the average waiting time for the three different weeks and a first discussion of this results. After that there are separate sections for each week discussing in detail the results by showing the site utilisation over time and the distribution of waiting times.

Figure 8.1 shows the average waiting time for all weeks and data volumes. The bins at zero show the average waiting time for the original distribution. The first thing to notice here is that the average waiting time between the first and the second week differ a lot, the 2 hours of the second week are less than half of the 5:15 hours of the first week. Taking into account the lower number of available job slots of the second week this is a first sign that the overall job load of this week is significantly lower than the first week. This will be even clearer when looking into the evolution of the jobs slot, which are discussed in detail later. Those two weeks were chosen because they show how the algorithm can positively affect the simulated system under different loads. The third week, on the other hand, was chosen to show a case when the algorithm could not enhance the original distribution and therefore did not improve the waiting time significantly for the smaller volumes. For the bigger volumes it even increases the average waiting time.

Figure 8.2 shows the relative time saved compared to the original distribution in percent. This highlights better how the algorithm effects the different weeks. For the second week the algorithm manages to achieve a maximum saving of close to 50% over the original distribution, whereas for the first week it is down to 25%. On the other hand, for the

Figure 8.1: Average waiting time for all three weeks and all simulated redistributed data volumes. The zero bin represents the original distribution.



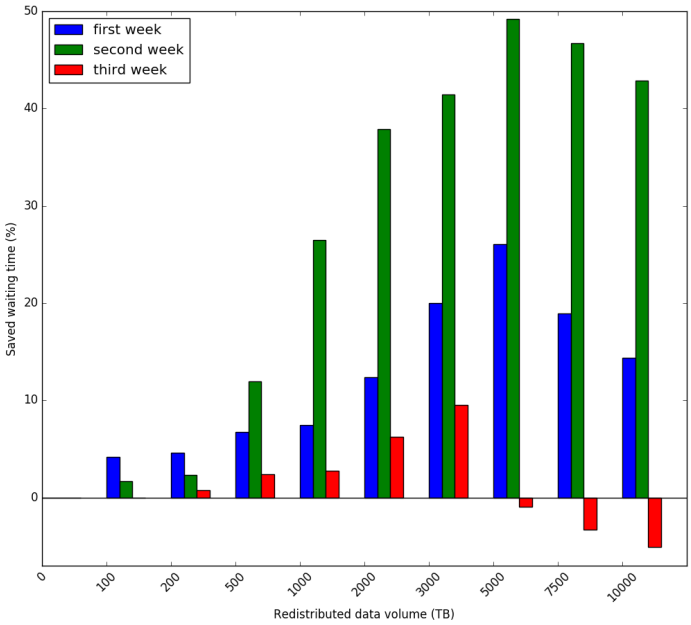
third week it only has a maximum of 8% and even goes down to -7% for the biggest data volume.

When first looking at the first week it can be seen that overall the waiting time constantly goes down with a higher data volume. The biggest gain can be observed at 5PB where the difference is 25%, after that the average waiting time even goes up again. This can be explained as at a certain volume the algorithm cannot delete unused data anymore to make space for the new replicas and therefore starts deleting replicas that would actually be useful for the brokering of the jobs. But 10PB is already more than a third of total amount of the datasets on DATADISK and it is so much that it would not make sense in a real system as it would put a lot of pressure on the transfer system and the storage elements without providing any benefit.

This is less the case when looking at the second week. The available computing resources are much less utilised by the system during that week. So on the one hand adding new replicas seems to have a bigger effect on the average waiting time and on the other hand the deletion of big volumes does not affect the system as much as in the first week.

For the third week the situation is different. The reason is that the system is at a high load, similar to the first week, but with the original distribution the system can already use almost all of the available resources and therefore changing the distribution does not change the average waiting time a lot. In general the waiting time also goes down

Figure 8.2: Percentage of waiting time saved for all three weeks and all simulated redistributed data volumes. The zero bin represents the original distribution.



in this week for the smaller volumes but only in a small margin and starting at 5PB the waiting time even goes up again above the original waiting time. This again can be explained with the deletions. Overall the positive effect of the new replicas is smaller in this week and the deletion of potentially popular replicas in the higher volumes cannot be compensated anymore by adding more replicas.

8.4.1 First Week

In this section the first week will be discussed in detail showing the waiting and running jobs over time, the distribution of waiting times, the distribution of new replicas of the sites and the job evolution for the top 3 sites picked by the redistribution algorithm. As the overview in Figure 8.2 shows, the biggest savings are achieved at a data volume of 5PB, so the following plots and table all detail on the 5PB case.

Figure 8.3 shows the job evolution of running and waiting jobs for the original distribution. The red line sets the maximum number of job slots for that week. During the first three days of the simulation the overall number of jobs is low, less than half of the job slots are used, but there are still some jobs waiting, if only a few. On the third day the number of jobs ramps up a lot, resulting in a number of running jobs close to the total number of job slots and a big amount of waiting jobs. It can then be observed that

after some time running jobs drop a bit and also the waiting jobs directly fall as well. After that there is a small bump in the running jobs resulting again in a bigger amount of waiting jobs until it slowly phases out at end of the week when there are again less jobs. The ratio of running to waiting jobs for this distribution is 1.2, but there is a big difference between the first 3 days and the rest of the week. During the first days the ratio is much higher with 2.6 compared to the rest with only 0.18.

Figure 8.4 now shows the same time interval but with a redistribution of 5PB. The beginning of the week is already better. Overall the number of running jobs is a little higher and therefore the waiting jobs are fewer. Then on the third day, when the number of jobs ramp up, there is no real difference as the number of running jobs in the original distribution are close to maximum. But unlike before, now the running jobs stay much closer to the maximum and the drop is a bit smaller, as the scheduling algorithm manages to use those extra replicas. When looking at the waiting jobs there are also two spikes at the beginning, but then when the running jobs drop the backlog of waiting jobs decreases much quicker and is nearly down to zero at the end of the week, whereas in the original distribution it is still around 30000 jobs. This is because more jobs could run already before, opening up job slots for the waiting jobs. Also, because of the extra replicas, the jobs could already be scheduled more widely across the sites beforehand, which then helps when working off the backlog.

The job ratio increases for this distribution to 2.97. Here again, the biggest difference can be observed in the first 3 days, where the ratio more than doubles with 6.3. But also for the rest of the week the ratio goes up to 0.3.

When looking at the distribution of waiting times as shown in the two plots in Figure 8.5 for the original distribution and for the redistribution of 5PB, it becomes apparent that the long waiting jobs profit most in this week. A logarithmic scale was chosen to properly encompass the wide range of job waiting time, from a few minutes to several days. The average waiting time goes down from 5:16 to 3:54 hours but it is dominated by the very long waiting time of the tail of the distribution. When looking at the mean time already half of the jobs are waiting 19 minutes or less in the original distribution which goes down to 13 minutes with the redistribution. So during this week the long waiting jobs are profiting most from the redistribution.

Table 8.4 shows the top 10 sites that have been chosen by the redistribution and Figure 8.6 shows how the job evolution has changed for three of those sites. The algorithm chose those sites as they have a lot of available job slots but not as many popular datasets as other sites. Particularly when looking at the first two sites it can be seen that at the beginning of the high load on day 3 those sites run at full capacity, but the grid is running

Figure 8.3: Job evolution for the first week with original distribution, showing the evolution of waiting jobs and running jobs for the time period of one week. The red line is static and represents the total number of job slots for this week. If the green line for the number of running is not visible anymore it overlaps with the red line and it means that all resources are fully exhausted.

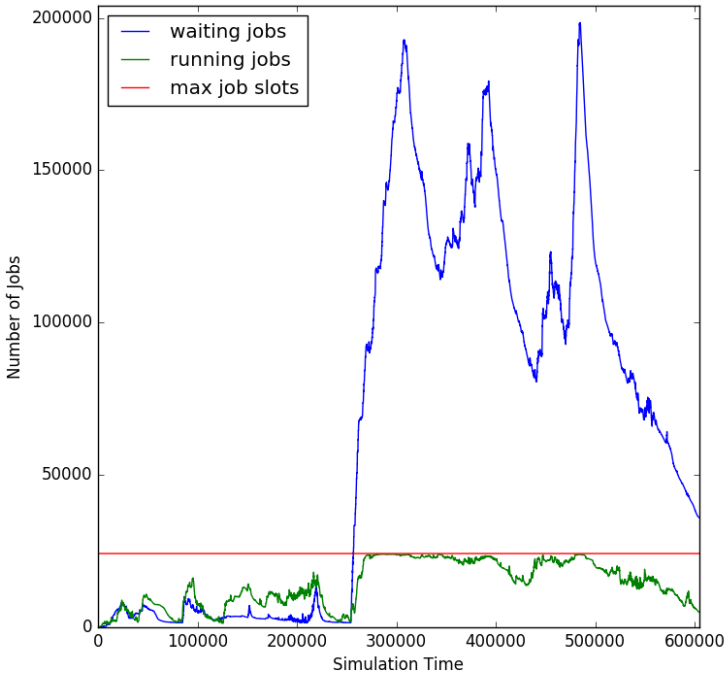
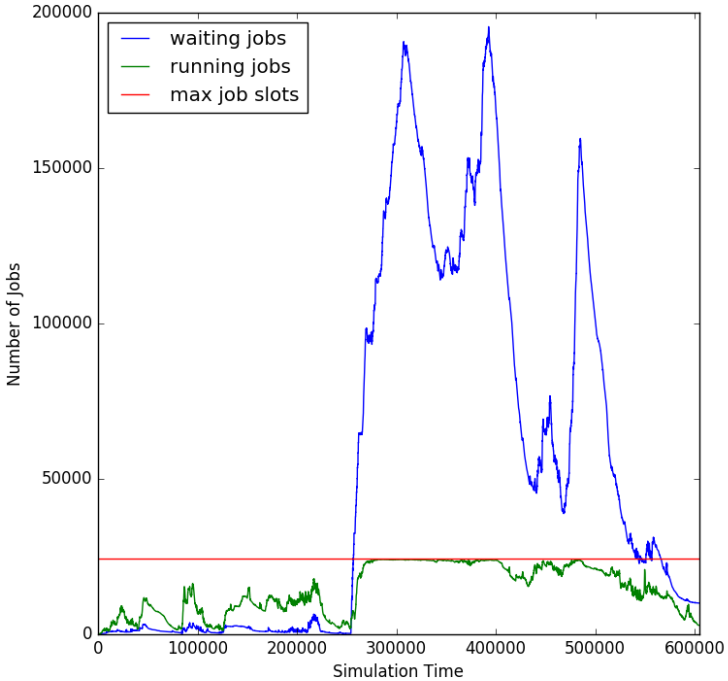


Figure 8.4: Job evolution for the first week with 5PB redistribution.



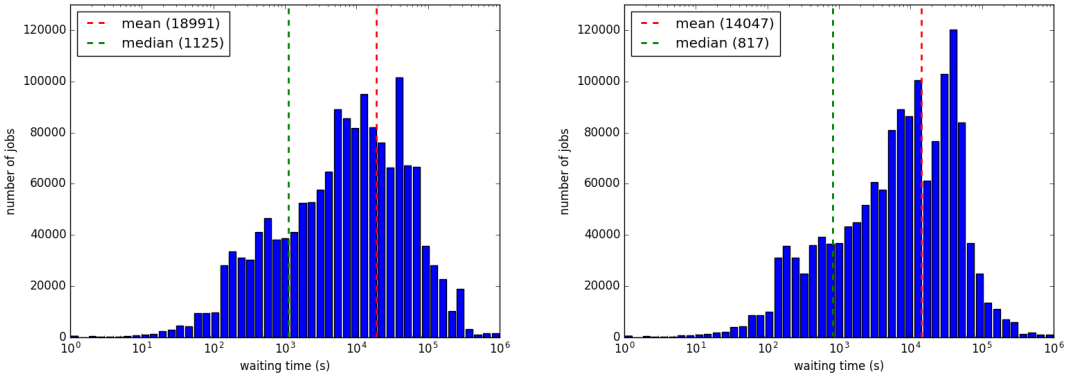


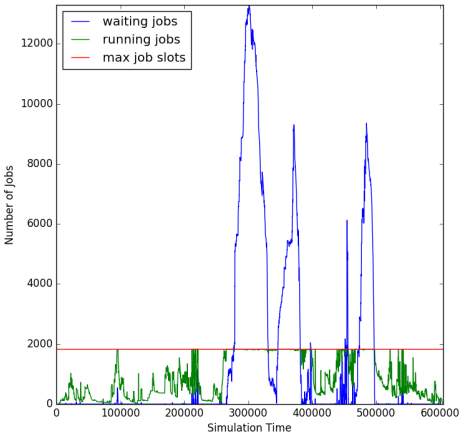
Figure 8.5: Distribution of waiting times for the first week. The left plot show the original data distribution, the right one a data redistribution of 5PB. A logarithmic scale was chosen for the x-axis to properly emcompass the wide range of job waiting times. This was necessary to be able to actually see the change for the long waiting jobs in the tail of the distribution. But this also means that the bins are skewed and everything left of of the median line seems much less but in fact it represent the same amount of jobs.

completely full at that time. Then, later on, those two sites start to become empty or close to empty already after half a day. On the other hand with the redistribution those sites stay full for a longer period of time as they now have more relevant datasets for the workload.

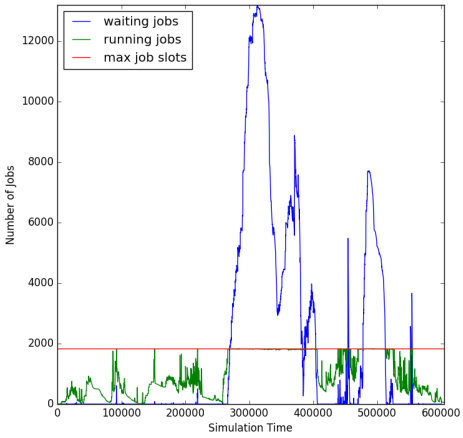
As a summary for this week it can be said the most difference in the average waiting time originates from the beginning and the end of the week, where the number of running jobs could be increased by the redistribution. During mid-week the impact was small, but the jobs could be scheduled differently compared to the original distribution. During this time, not much gain could be achieved, but it helps later when working off the backlog. But at the end of the week the extra replicas helped the scheduling algorithm, as it has more possible sites and can already send the jobs to a wider number of sites, so then the system later can more rapidly process the backlog.

8.4.2 Second Week

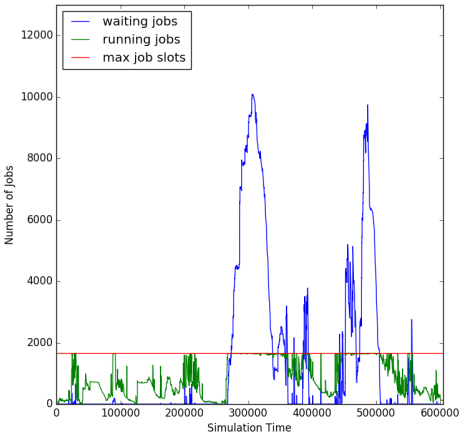
The situation overall is much different for the second week. When looking at the initial job evolution of the original distribution in Figure 8.7 it can be seen that the workload for this week is different to the first week. In this week the number jobs at the beginning is low and the concurrently running jobs are taking only half of the available job slots. The difference here is now that there is not a big ramp up of jobs as in the first week.



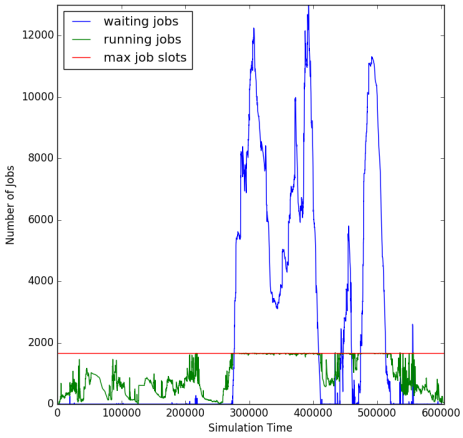
SWT2_CPB Original



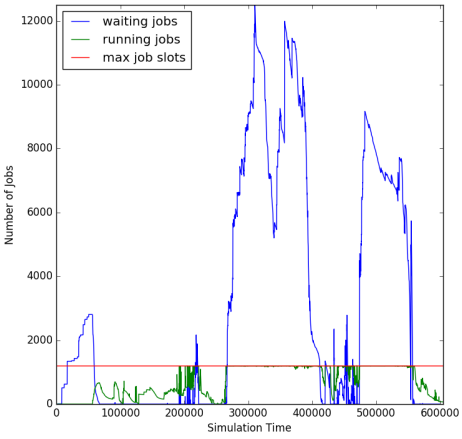
SWT2_CPB 5PB



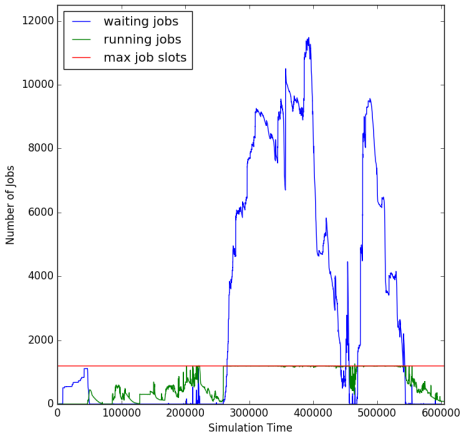
IN2P3-CC Original



IN2P3-CC 5PB



RAL-LCG2 Original



RAL-LCG2 5PB

Figure 8.6: Job evolution for the top 3 redistribution sites for the first week.

Table 8.4: Top 10 sites for the first week.

Site	**New Replicas**	**New Replica Size**
SWT2_CPB	2343	612.0TB
IN2P3-CC	1855	433.2TB
RAL-LCG2	1483	403.9TB
MWT2	1286	394.5TB
AGLT2	933	391.5TB
RU-PROTVINO-IHEP	1098	292.0TB
DESY-HH	703	259.9TB
UKI-NORTHGRID-MAN-HEP	769	215.9TB
FZK-LCG2	704	190.4TB
TOKYO-LCG2	1212	178.3TB
total	19613	5PB

Overall the number of running jobs rises a bit on day two and the number of waiting jobs spikes, but with the original distribution the system never manages to utilise all of the available resources. The jobs continue like this with daily spikes and it is apparent that the average waiting time for this week will be much smaller than in the first week. The job ratio for this week is just 0.47 and compared to the first week there is not much difference between the beginning and the end of the week.

Figure 8.8 shows the same time period but with a redistribution of 5PB. The first thing that can be noticed is that overall waiting jobs are much less. There are still the spikes as before but they are smaller now and decrease faster. Whereas before the number of waiting jobs were always above the number of running jobs, now in several periods during the week they are below the running jobs. This is also underlined by much higher job ratio, which nearly triples up to 1.45. At the beginning of the week the number of running jobs is already slightly higher and the redistribution prevents the system from building bigger queues already here. Most of the saving is achieved during mid-week. The number of running jobs in general is higher and at some points very close the maximum capacity and the spikes are a lot lower. Interestingly, during some period where the systems works off the backlog of waiting jobs also the running jobs are lower than in the original distribution, as more jobs could run in parallel before and have finished earlier.

The overall lower number of waiting jobs can be explained by that the scheduling algorithm manages with the extra replicas to spread the jobs more widely over the sites, which results in overall more running jobs. Those will then again not be in the waiting queues anymore, so that the queues are not filling up much. Then, when the next spike

comes the new jobs do not add up to the already existing waiting jobs and so the waiting queues overall remain less used during this week.

In Figure 8.9 on the left the distribution of waiting time is shown for the original distribution. As seen before, the average waiting time is already much lower compared to the first week at around 2 hours, but the median is higher at 25 minutes. So overall there are not so many very long waiting jobs but the short waiting jobs have to wait longer. This can be explained as in the first week it takes longer until the jobs ramp up but then there is a big continuous queue, so in the beginning there are a lot of very short waiting jobs but then in the end the waiting jobs have to wait longer. In this week on the other hand with the daily spikes there at not so many very long waiting jobs anymore as the system can work them off already at the end of each spike, but the spikes start earlier in the week and more jobs are affected and therefore there is a bigger amount of jobs that have to wait longer.

With the redistribution as shown in Figure 8.9 on the right the average nearly halves to about an hour but the median even reduces to a third of the original distribution. So half of the jobs have to wait less then 9 minutes. From the distribution it is apparent that most of the savings comes from jobs with medium to long waiting times. Overall for the short waiting times below 2 minutes there is no difference. There are only very few jobs with a very long waiting above a day, those are reduced. So the jobs originally waiting 2 hours and more benefit the most and reduce their waiting time to under an hour.

Table 8.5 shows again the top 10 sites that have been chosen by the redistribution algorithm. Compared to the first week the algorithm picked the sites differently based on the predicted popularity and the original distribution. When looking at the first two sites in Figure 8.10 it can be seen that the sites are affected quite differently during this week. Those two sites were already utilised a lot with the original distribution. Redistributing popular datasets to those sites actually did not increase the load of the sites but overall it went down. When looking at the third site it can be seen that this site has a slightly higher amount of overall running jobs especially in the beginning and middle of the week. This is similar for most of the rest of the redistributed sites. So there is where most of the savings have been achieved. When looking at the job evolution of the whole grid it shows that the available jobs slots, even with the redistribution, are never fully used. The full grid might have been possible if the algorithm had chosen to move the replicas it replicated to the first two sites to other less used sites. But this could have only be done with more fine-granular predictions in a week like this with several spikes and low load periods.

Figure 8.7: Job evolution for the second week with original distribution.

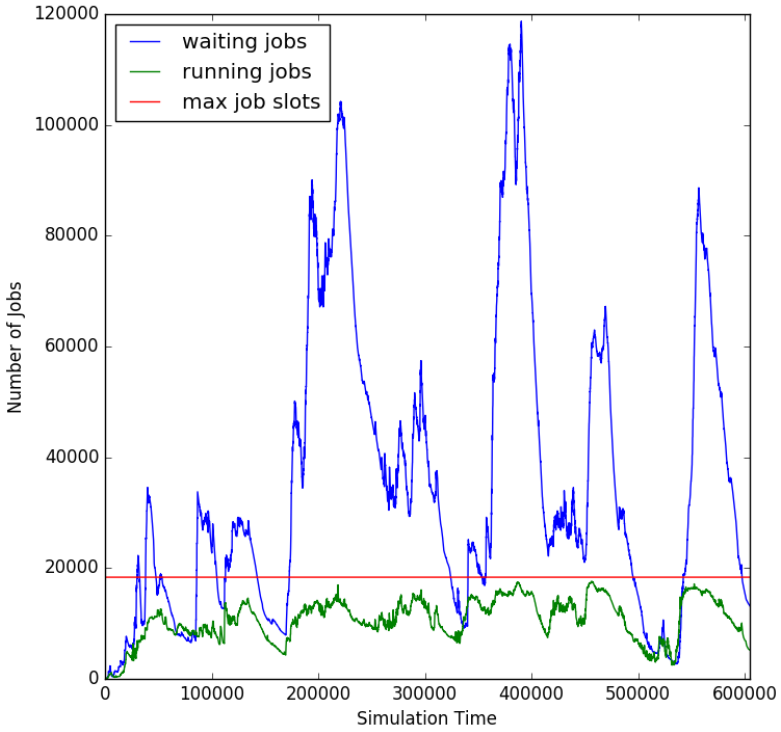
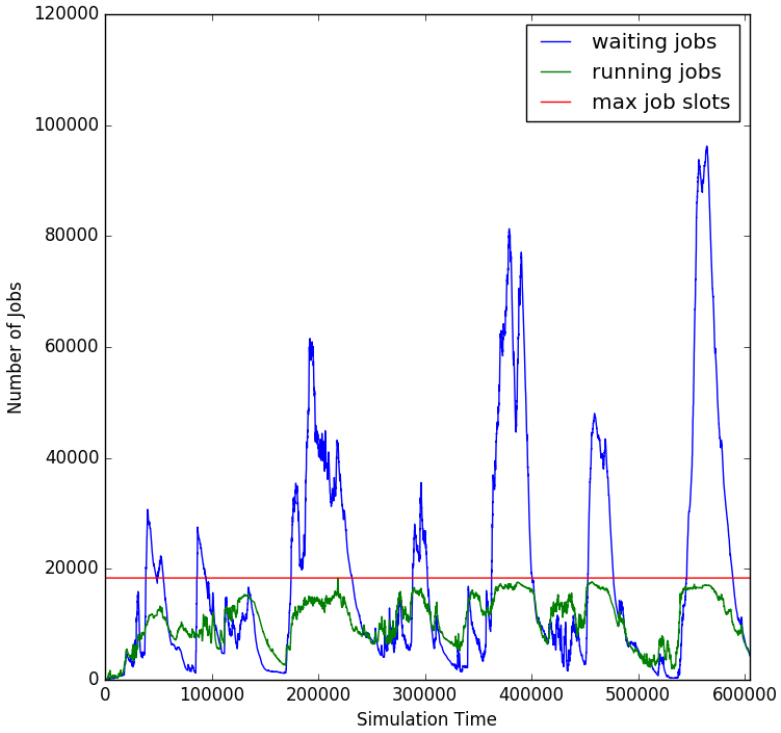


Figure 8.8: Job evolution for the second week with 5PB redistribution.



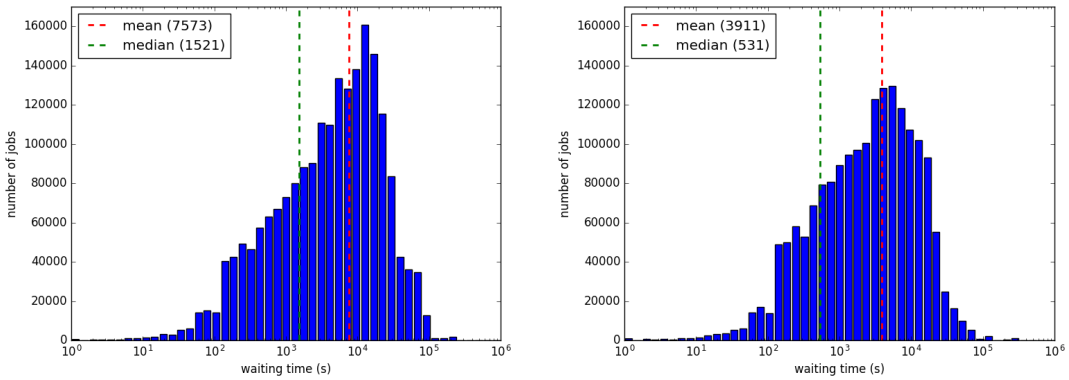
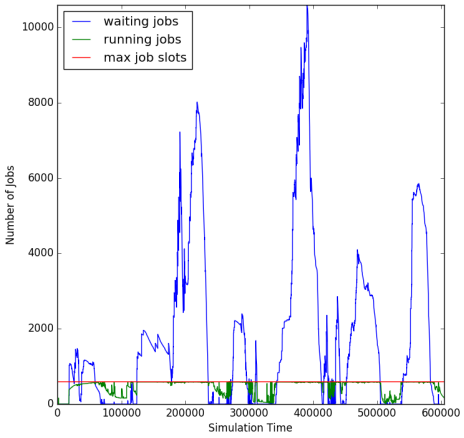


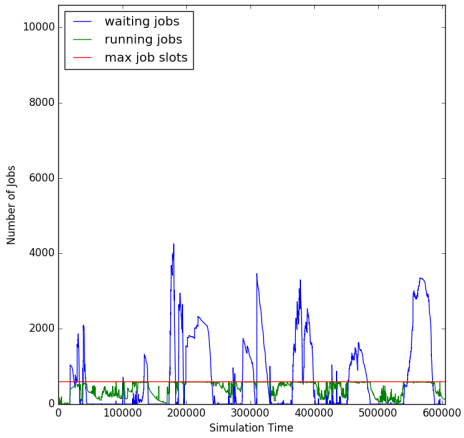
Figure 8.9: Distribution of waiting times for the second week. The left plot show the original data distribution, the right one a data redistribution of 5PB. As before, a logarithmic scale was chosen for the x-axis.

Table 8.5: Top 10 sites for the second week.

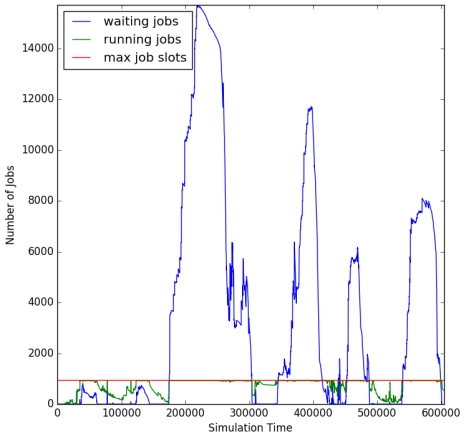
Site	**New Replicas**	**New Replica Size**
RAL-LCG2	1973	526.9TB
TRIUMF-LCG2	3235	501.9TB
INFN-T1	1123	398.5TB
FZK-LCG2	1633	389.3TB
NDGF-T1	653	243.3TB
PIC	586	216.2TB
MWT2	1104	194.7TB
UKI-SCOTGRID-GLASGOW	640	186.8TB
NIKHEF-ELPROD	571	183.4TB
SFU-LCG2	443	171.6TB
total	21458	5PB



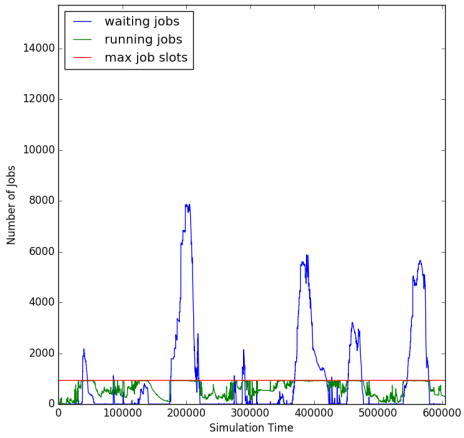
TRIUMF-LCG2 Original



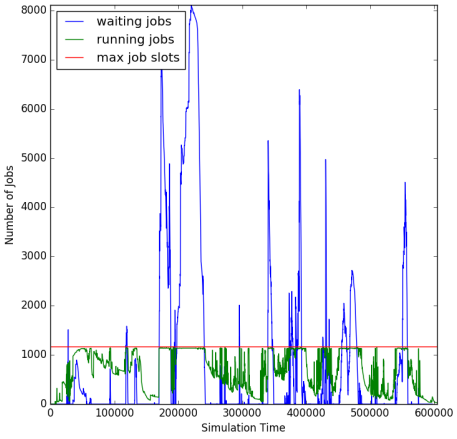
TRIUMF-LCG2 5PB



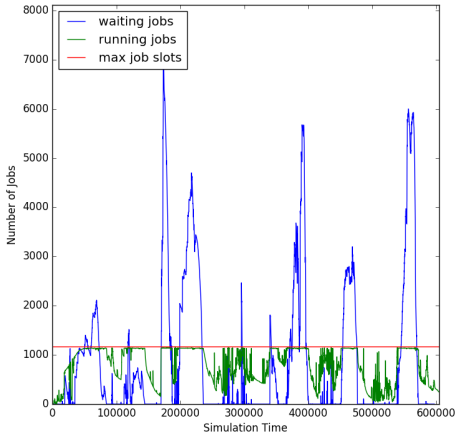
RAL-LCG2 Original



RAL-LCG2 5PB



INFN-T1 Original



INFN-T1 5PB

Figure 8.10: Job evolution for the top 3 redistribution sites for the second week.

In summary for this week it can be said that a week like this with medium load which is spread evenly across the week can benefit a lot from the redistribution. The additional replicas help the scheduling a lot so that for every spike of incoming jobs the backlog of waiting jobs does not build up as much and overall the waiting queues are much smaller, which then again results in smaller waiting times. As can be seen when looking at the job evolution of the separate sites, this could have been also achieved with a smaller data volume, but for this a more fine-grained prediction and redistribution would be necessary.

8.4.3 Third Week

The third and last week shows an example when the redistribution algorithm cannot effectively reduce the waiting times. As shown in Figure 8.2 the average waiting times only improve slightly and even become worse at very high data volumes. When looking at the job evolution of the original distribution in Figure 8.11 it can be seen that overall it is similar to the first week. Even in the original distribution the system can utilise almost all available resources during peak times. After a redistribution of 3PB it can still improve the job slot utilisation a bit in some cases as shown in Figure 8.12 but overall the improvements are little and the system is already running at its limit. The small difference can also be observed in the job ratio, which only slightly increases from 1.7 for the original distribution to 1.9 with the redistribution.

This can also be seen when looking at the change of the waiting time distribution in Figure 8.13 for the original distribution and for the 3PB redistribution. Overall the change in the average waiting time from 4:40 hours to 4:10 hours is marginal. Also the big difference to the other weeks is the median of 1:12 hours. There are in total far fewer short waiting jobs as the peak workload already starts earlier in the weak compared to the first week and does not reduce a lot until the end of the week. Even with a 3PB redistribution the median only goes down to 55 minutes. From the distribution only a few long running jobs have a slight reduction in waiting times but for the short running jobs there is barely any change.

The only way in this week to improve the waiting time significantly would be if the workload management system would reschedule jobs that are already submitted into a the waiting queue at a site. When looking at the job evolution it becomes apparent that when the running jobs go down also the waiting jobs decline, but when those waiting jobs could have been directly re-submitted to the just freed job slots the grid could run longer at full capacity. The caveat here is that with the scheduling algorithm as simulated here the jobs get scheduled when they are submitted. During the submission time the queues

Figure 8.11: Job evolution for the third week with original distribution.

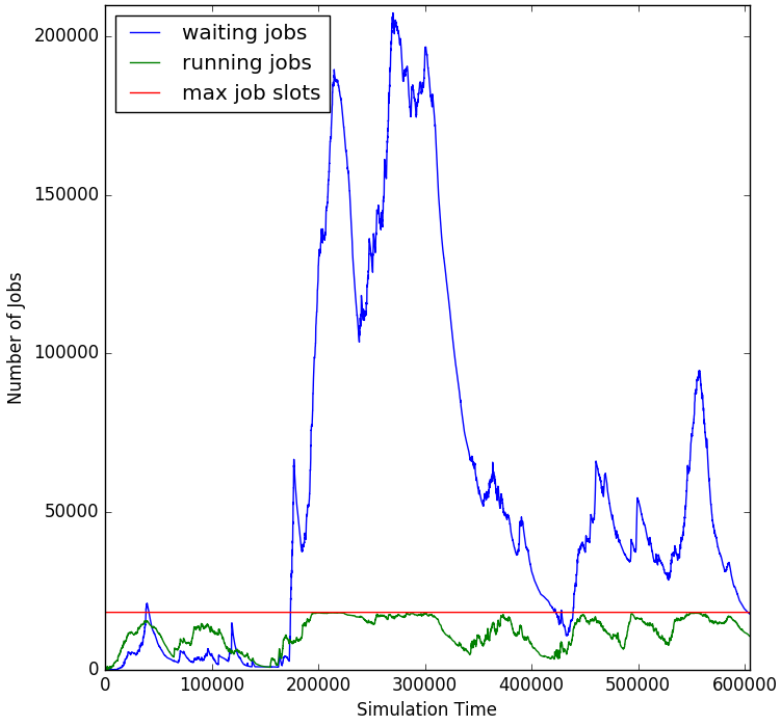
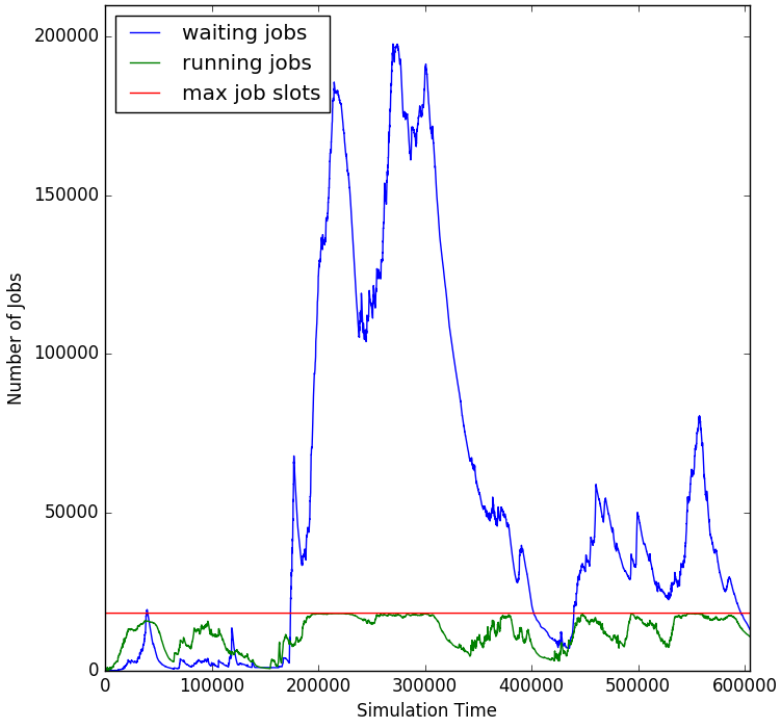


Figure 8.12: Job evolution for the third week with 3PB redistribution.



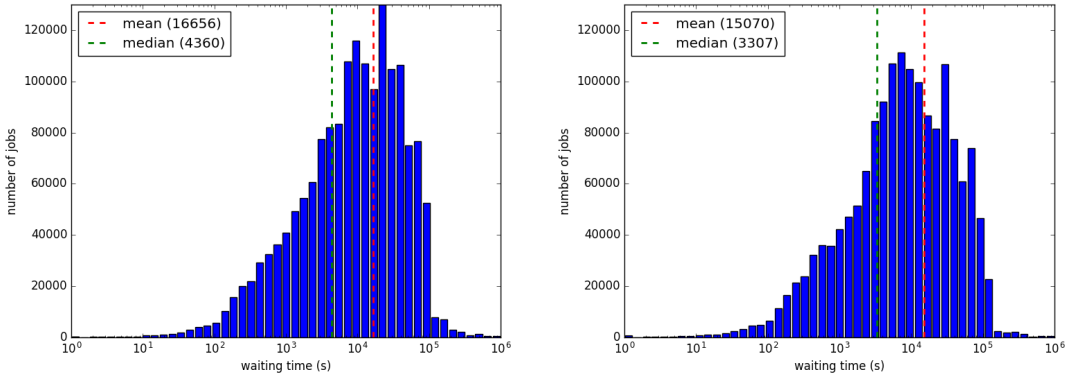


Figure 8.13: Distribution of waiting times for the third week. The left plot show the original data distribution, the right one a data redistribution of 3PB. As before, a logarithmic scale was chosen for the x-axis.

at a specific site might have been small but the running jobs or the ones in the queue might be long running jobs. The scheduling algorithm cannot know this at that moment and would send the job to this site. The simulation goes on and at some point job slots at another side with the needed replica become free but the job is already scheduled for the first site, so it will have to wait. If the scheduling algorithm would now re-examine already queued jobs on a regular basis it could remove the job from a queue on one site and send it directly to a free job slot at another side.

8.5 Summary

The results of this simulation show that is is possible in general to reduce the user waiting times if the conditions are right, i.e., the original distribution does not allow the scheduling algorithm to fully use all of the available computing resources. The ratio of running to waiting jobs can improve a lot and also the waiting times can benefit. On the other, if the jobs can already be scheduled optimally with the original distribution, then the improvements are marginal or at some point the waiting time can even increase. With improvements to the prediction and redistribution algorithms this can potentially be spotted beforehand, which will be briefly discussed in the outlook in the next chapter.

Part IV

Summary

Chapter 9

Conclusion and Outlook

9.1 Summary

This study has shown that it is possible to predict future dataset accesses, that can be used in an effective way to redistribute data, so that user analysis jobs can benefit from shorter waiting times.

The prediction evaluation proved, that a split of the input data by project and datatype already provides acceptable results. Together with a careful pre-filtering of the data the combination of neural network and static prediction provided a good input for the redistribution.

The redistribution part introduced an effective algorithm to dynamically redistribute data evenly over the available computing and storage resources by removing unpopular data and filling the free space with popular data and taking into account available job slots.

The simulator has proven to be a simple, but effective, tool to quickly run the same workload for different distributions, producing helpful output to analyse the underlying effects.

And finally, the evaluation of the prediction and redistribution put together showed that they can give very good results when the conditions are right. This leads to possible future improvements, that will be discussed in the remainder of this work.

9.2 Outlook

Overall the experiences collected during this study provide a solid grounding for a redistribution system that is currently in development, which will be briefly introduced at

the end of this chapter. But first, open questions and possible working points will be addressed in this section.

9.2.1 Open Questions

- **Evaluation of given distribution:**
  The evaluation has shown that the effectiveness of the redistribution depends a lot on the current distribution on the grid and the workload for the week. The overhead of the redistribution only pays off when the conditions are right, so an evaluation is needed that tries to assess the possible benefits before starting a redistribution. This could be addressed by predicting the accesses for the given week and computing the accumulated accesses for each site as introduced in 6.3.1. If a big imbalance for several sites is anticipated a redistribution could be started.

- **Continuous redistribution:**
  The presented redistribution model only does a redistribution once a week. In the production system this could lead to several problems. The redistribution would saturate the links between sites and interrupt other parallel transfers. The large number of deletions and creations of files would stress the storage systems a lot. So instead, a smaller period between redistribution runs would be better. For this a few changes are needed:

  - **Prediction for smaller periods:**
    If the redistribution would be done on smaller periods, for example daily, a weekly prediction would not be granular enough anymore. Therefore, neural networks need to be trained using those smaller periods. They could then be used to spot datasets that will become popular on the next day. On the other hand the long-term neural networks can still be used to supplement the decision making by spotting datasets that will be popular for a longer period. The distribution would benefit in this case by additional replicas that could be used directly on the following day and also potentially by jobs in the long-term.

  - **Memory of already distributed datasets:**
    Depending on the period, the algorithm needs to track the already created replicas. The algorithm always starts with the most popular dataset. If a particular dataset stays popular over several periods, the algorithm would continue to create a new replica each time, if the threshold of replicas is not passed. As the turnover would also be smaller when using smaller periods,

it could happen that datasets that are further down the popularity list will never get a new replica. For this case the algorithm could have a memory where it stores all replicas it created and when it created them. The datasets in the memory will be ignored and they could be deleted from the cache after some periods.

  - **Re-training of neural networks:**
    For this evaluation the neural networks are trained once for each project / dataset combination and then directly evaluated. With a continuous redistribution also the neural networks would have to be re-trained periodically to incorporate new patterns into the model. The networks could be saved to be available for querying at any time and then periodically re-trained.

- **Evaluation of other machine learning techniques:**
  For this evaluation the neural networks have been chosen as they have shown good to very good results in predicting the future accesses, but there are also other techniques available to model time-series, like the autoregressive integrated moving average (ARIMA), that could be evaluated in the future and compared to the neural network predictions.

- **Improvements for the simulator:**
  For this study the simulator was developed to measure directly the effects of a changed distribution while putting some characteristics of the real system aside. To improve the simulator the following points could be added:

  - **Include network / storage systems in simulation:**
    A big part that has been ignored in this study is the impact of the network and the storage systems. In the currently simulated system the transfers and deletions always happened instantaneously. Adding those systems to the simulation is not trivial as it would require modelling the different types of storage system that are used in the grid and the transfer links. Work on this topic has already been done [44] and is currently worked on and it could be incorporated into the simulation.

  - **Site availability:**
    In the real system the sites can be temporarily blacklisted for reading and writing for numerous reasons, e.g., hardware maintenance / upgrade, power loss or problems with the storage system or batch queues. In those cases the sites will not be available for the redistribution. Unless it is a scheduled operation, those events occur randomly, which makes it harder actually sim-

ulate them. But it could be done by adding a configurable random factor the simulation that removes a certain set of sites for some given time.

– **Include Production system:**
Another missing part is the simulation of the production system jobs. Overall this workload is less chaotic, as it is not run by users directly, unlike the analysis jobs, but by production managers with well defined tasks. As already mentioned in the evaluation chapter, the resources between Production and Analysis jobs are shared and they are dynamically allocated between the two systems. The jobs itself are similar to the analysis jobs and could easily be added to the simulation.

9.2.2 Further developments

In preparation for the increased needs of ATLAS during Run 2, a completely new DDM system, called Rucio [16], has been developed. As one part of this system a redistribution daemon called C3PO is being developed. This daemon is still under active development and will be extended in many ways, but already now, many of the experiences collected during this study have been used in its development.

The daemon has three main types of components that can be added and removed easily from the system, like plug-ins. Those types are:

- **Collectors:**
The collectors gather information from different sources and make those available to the redistribution algorithms. Currently the system supports collectors to get newly defined jobs and their input datasets from PanDA, free space for all storage endpoints, topology information, network metrics and job slots.

- **Algorithms:**
The redistribution algorithms can use the data gathered by the collectors to decide when and where to make new replicas.

- **Utilities:**
The utilities are tools that provide needed helper functions for the algorithms, e.g., an expiring dataset cache to store which datasets have been newly created, so that they will not be considered again for some pre-defined time.

The current implementation of the daemon is not using a popularity prediction yet, but instead continuously polls for incoming definitions of user analysis jobs. If a new job

has been defined, the redistribution algorithm is triggered with the name of the dataset used as input for the job. It then checks several things: past popularity of the dataset in the previous week, already available replicas of the datasets, and if a replica has already been created in the last 24 hours. If those test pass, the algorithm decides where to place a new replica. The current implementation bases the decision mainly on two parameters: the free space of a site and the network connections, but the algorithms are easily exchangeable and for the future computing resources will also be considered. The current algorithm ranks the sites first on their relative free space and then compares each possible destination site with all sites already having a replica to find an optimal link. If the decision has been made the daemon places a transfer request and the DDM system takes care of the replication.

The deletion part in this model is handled by another daemon call Reaper. The Reaper periodically runs and checks the free space on each storage endpoint. If the free space goes below a certain value the deletion is triggered. For the deletion the Reaper sorts the replicas at a site by their last access date, which is set based on the traces, and then starts deleting unused data until their enough free space again.

The daemon is currently running in a monitored, pre-production mode, already creating replicas on the grid. There is currently work undergoing to extend the daemon incorporating time-to-complete predictions for file transfers between two links and also dataset popularity predictions.

Part V

Appendix

List of Figures

List of Tables

Chapter 10

Code Samples

10.1 Prediction

Listing 10.1: 'Neural Network Initialisation'

```
class Neural:
    def __init__(accs, num_weeks):
        self._dsns = []
        self._accs = []
        m = 0
        n = 0

        for dsn, prev in prev_accs.items():
            self._dsns.append(dsn)
            self._accs.append(prev)
            m += 1
            n = len(prev)

        self._accs = array(accs).reshape(m, n)

        self._minmax = (self._accs.min(), self._accs.max())

        self._scale_range = (-1.0, 1.0)
        self._scaled_accs = self.scale_down(self._accs)

        self._train_inp = accs[:, -(num_weeks):-1]
        self._train_tar = accs[:, -1].reshape(m, 1)
```

```
        self._pred_inp = accs[:, -(num_weeks - 1):]

        self._scaled_inp = self.scale_down(self._train_inp)
        self._scaled_tar = self.scale_down(self._train_tar)
        self._scaled_pred_inp = self.scale_down(self._pred_inp)

        in_neurons_scale = []
        for i in xrange(num_weeks):
            in_neurons_scale.append(self._scale_range)

        self._net = newelm(in_neurons_scale, (num_weeks, num_weeks, 1))

        if (not self.train_nn(goal, epochs, runs)):
            print("Could not train network")
```

Listing 10.2: 'Scale Down/Up'

```
def scale_down(self, data):
    a, b = self._scale_range[0], self._scale_range[1]
    min, max= self._minmax[0], self._minmax[1]

    return ((b - a) * (data - min) / (max - min)) + a

def scale_up(self, data):
    a, b = self._scale_range[0], self._scale_range[1]
    min, max = self._minmax[0], self._minmax[1]

    return (data - a) * ((max - min) / (b - a)) + min
```

Listing 10.3: 'Training'

```
    def train_nn(self, goal, epochs, runs):
        for i in range(runs):
            error = self._net.train(self._scaled_inp, self._scaled_tar,
                                    epochs=epochs, goal=goal)
            if error <= goal:
                return true
        return false
```

Listing 10.4: 'Prediction'

```
    def predict(self):
        res = self._net.sim(self._pred_inp)
        scaled_res = self.scale_up(res)
        predictions = {}
        for i in xrange(len(res)):
            predictions[self._dsns[i]] = scaled_res[i]

        return predictions
```

10.2 Redistribution

Listing 10.5: 'The Redistribution Main Method'

```
def redistribute(self, max_volume, minimum_age, storage_type):
    total_moved = 0
    redistributed_replicas = []
    for dataset, accesses in predicted_accesses:
        size = get_dataset_size(dataset)
        if (total_moved + size) > max_volume:
            break
        sorted_sites = generate_sorted_sites(storage_type)
        for site, se in sorted_sites:
            delete_replicas, free_space =
                clean_storage_endpoint(site, se, size, minimum_age):
            if free_space < size:
                continue

            for replica in delete_replicas:
                delete_replicas(replica.name, site, se)
                redistribute_replicas.append(
                    (replica.name, site, se, 'remove'))

            add_replica(dataset, size, site, se, now)
            redistribute_replicas.append((dataset, site, se, 'add'))

            total_moved += size
```

```
            break

    return redistributed_replicas
```

Listing 10.6: 'Generate Sorted Site List'

```
def generate_sorted_sites(storage_type):
    accumulated_accesses_per_site = {}
    for site in sites:
        if not site.has_storage_type(storage_type):
            continue
        accumulated_accesses = 0
        replicas = get_replicas_for_site(site.name)
        for replica in replicas:
            if replica in predicted_accesses:
                accumulated_accesses += predicted_accesses['replica']
        accumulated_accesses_per_site[site.name] =
            accumulated_accessses / site.get_job_queue_size()

    sorted_sites = sorted(x.items, key=itemgetter(1), reverse=True)

    return sorted_sites
```

Listing 10.7: 'Clean Storage Endpoint'

```
def clean_storage_endpoints(site, se, needed_space, minimum_age);
    deleted_bytes = 0
    eligible_replicas = []
    for replica in list_replicas_per_se(site, se):
        creationdate = get_replica_creationdate(replica.name, site, se)
        if creationdate > minimum_age:
            continue
        replicas = list_replicas_for_dataset(replica.name)
        if len(replicas) <= min_replicas:
            continue
        if replica.name in historic_accesses:
            continue

        eligible_replicas.append((replica, replica.size))
```

```
    delete_replicas = []
    sorted_replicas =
        sorted(eligible_replicas, key=itemgetter(1), reverse=True)
    for replica in sorted_replicas:
        if deleted_bytes >= needed_space:
            break

        deleted_bytes += replicas.size
        delete_replicas.append(replica)

    return replicas
```

10.3 Simulator

Listing 10.8: 'The Site module'

```
class Site:
    def __init__(self, capacity):
        self.queue = simpy.Resource(capacity=capacity)
```

Listing 10.9: 'The Job module'

```
class Job():
    def __init__(self, env, site, execution_time):
        self.env = env
        self.site = site
        self.execution_time = execution_time

    def run(self):
        queue = self.site.queue
        scheduled = self.env.now
        job_slot = queue.request()
        yield job_slot
        started = self.env.now
        yield self.env.timeout(self.execution_time)
        job_slot.release()
        finished = self.env.now
```

Listing 10.10: "The DDM cache module"

```
class DDMCache:
    def add_replica(self, dataset, site, se):
        if name not in self.catalogue:
            self.catalogue[dataset] = []
        self.catalogue[dataset].append(se)
        return True

    def get_replicas(self, dataset):
        return self.catalogue[dataset]
```

Reference

[1] O. S. Brüning, P. Collier, P. Lebrun, S. Myers, R. Ostojic, J. Poole, and P. Proudlock, *LHC Design Report*. Geneva: CERN, 2004. [Online]. Available: https://cds.cern.ch/record/782076

[2] C. Service graphique, "Overall view of the LHC. Vue d'ensemble du LHC," jun 2014. [Online]. Available: https://cds.cern.ch/record/1708849

[3] ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation*, vol. 3, no. 08, p. S08003, aug 2008. [Online]. Available: http://stacks.iop.org/1748-0221/3/i=08/a=S08003

[4] J. Fleckner, "Commissioning of the ATLAS inner detector with cosmic rays," *J. Phys.: Conf. Ser.*, vol. 219, p. 32038, 2010. [Online]. Available: https://cds.cern.ch/record/1270199

[5] ATLAS Collaboration, "Readiness of the ATLAS Liquid Argon Calorimeter for LHC Collisions," *Eur. Phys. J. C*, vol. 70, no. arXiv:0912.2642. CERN-PH-EP-2010-041, pp. 723–753. 31 p, may 2010. [Online]. Available: https://cds.cern.ch/record/1228823

[6] J. Maneira, "Commissioning of the ATLAS Tile Calorimeter," nov 2009. [Online]. Available: https://cds.cern.ch/record/1221300

[7] Y. Takahashi, "ATLAS Muon Spectrometer," oct 2009. [Online]. Available: https://cds.cern.ch/record/1211567

[8] J. Pequenao, "Computer generated image of the whole ATLAS detector," mar 2008. [Online]. Available: https://cds.cern.ch/record/1095924

[9] ATLAS Collaboration, "ATLAS high-level trigger, data-acquisition and controls: Technical Design Report," *Atlas-Tdr-16; Cern-Lhcc-2003-022*, no. July, 2003. [Online]. Available: https://cds.cern.ch/record/616089

[10] C. Eck, J. Knobloch, L. Robertson, I. Bird, K. Bos, N. Brook, D. Düllmann, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, J. Harvey, A. Heiss, F. Hemmer, S. Jarp, R. Jones, D. Kelsey, M. Lamanna, H. Marten, P. Mato-Vila, F. Ould-Saada, B. Panzer-Steindel, L. Perini, Y. Schutz, U. Schwickerath, J. Shiers, and T. Wenaus, *LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005)*, ser. Technical Design Report LCG. Geneva: CERN, 2005. [Online]. Available: https://cds.cern.ch/record/840543

[11] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, I. Fisk, M. Girone, J. Harvey, B. Kersevan, P. Mato, R. Mount, and B. Panzer-Steindel, "Update of the Computing Models of the WLCG and the LHC Experiments," Tech. Rep. CERN-LHCC-2014-014. LCG-TDR-002, apr 2014. [Online]. Available: https://cds.cern.ch/record/1695401

[12] "The Worldwide LHC Computing Grid website," 2014. [Online]. Available: http://wlcg.web.cern.ch

[13] R. Jones and D. Barberis, "The ATLAS computing model," *Journal of Physics: Conference Series*, vol. 119, no. 7, p. 72020, jul 2008. [Online]. Available: http://stacks.iop.org/1742-6596/119/i=7/a=072020

[14] "Persistency Web," 2016. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/Persistency

[15] V. Garonne, G. A. Stewart, M. Lassnig, A. Molfetas, M. Barisits, T. Beermann, A. Nairz, L. Goossens, F. Barreiro Megino, C. Serfon, D. Oleynik, and A. Petrosyan, "The ATLAS Distributed Data Management project: Past and Future," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 32045, dec 2012. [Online]. Available: http://stacks.iop.org/1742-6596/396/i=3/a=032045

[16] V. Garonne, R. Vigne, G. Stewart, M. Barisits, T. Beermann, M. Lassnig, C. Serfon, L. Goossens, and A. Nairz, "Rucio - The next generation of large scale distributed system for ATLAS Data Management," *Journal of Physics: Conference Series*, vol. 513, no. 4, p. 042021, jun 2014. [Online]. Available: http://stacks.iop.org/1742-6596/513/i=4/a=042021?key=crossref.2f2f6aed0f8b9cc1e22f22d60edd6c61

[17] "SantaClaus," 2014. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/SantaClaus

[18] "Data Transfer Request Interface," 2014. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/DataTransferRequestInterface

[19] "ATLAS Computing Resource Management." [Online]. Available: https://twiki.cern.ch/twiki/bin/viewauth/Atlas/ComputingResources

[20] T. Maeno, K. De, T. Wenaus, P. Nilsson, G. A. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, and D. Smith, "Overview of ATLAS PanDA Workload Management," *Journal of Physics: Conference Series*, vol. 331, no. 7, p. 72024, dec 2011. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=7/a=072024

[21] P. Nilsson, J. Caballero, K. De, T. Maeno, A. Stradling, and T. Wenaus, "The ATLAS PanDA Pilot in Operation," *Journal of Physics: Conference Series*, vol. 331, no. 6, p. 062040, dec 2011. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=6/a=062040?key=crossref.2e095972d7ca249b0f4e57905da1d143

[22] D. Zang, V. Garonne, M. Barisits, M. Lassnig, G. A. Stewart, A. Molfetas, and T. Beermann, "The ATLAS DDM Tracer monitoring framework," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 32119, 2012. [Online]. Available: http://stacks.iop.org/1742-6596/396/i=3/a=032119

[23] A. Molfetas, F. B. Megino, A. Tykhonov, M. Lassnig, V. Garonne, M. Barisits, S. Campana, G. Dimitrov, S. Jezequel, I. Ueda, and F. T. A. Viegas, "Popularity framework to process dataset traces and its application on dynamic replica reduction in the ATLAS experiment," *Journal of Physics: Conference Series*, vol. 331, no. 6, p. 062018, dec 2011. [Online]. Available: http://iopscience.iop.org/1742-6596/331/6/062018

[24] "Victor Documentation," 2011. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/DDMVictorHowTo

[25] T. Maeno, K. De, and S. Panitkin, "PD2P: PanDA Dynamic Data Placement for ATLAS," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 32070, dec 2012. [Online]. Available: http://stacks.iop.org/1742-6596/396/i=3/a=032070?key=crossref.51564fef0e85210677dbbda0230ca158

[26] G. Dorffner, "Neural networks for time series processing," *Neural Network World*, vol. 6, no. 4, pp. 447–468, 1996.

[27] J.-N. Hwang and E. Little, "Recurrent neural networks and time series prediction," *Proceedings of International Conference on Neural Networks ICNN96*, vol. 4, p. 220, 1991. [Online]. Available: http://www.csa.com/partners/viewrecord.php?requester=gs{&}collection=TRD{&}recid=0107583CI

[28] I. Kaastra and M. Boyd, "Designing a neural network for forecasting financial and economic time series," *Neurocomputing*, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0925231295000399

[29] K. A. de Oliveira, Á. Vannucci, and E. C. da Silva, "Using artificial neural networks to forecast chaotic time series," *Physica A: Statistical Mechanics and its Applications*, vol. 284, no. 1-4, pp. 393–404, 2000. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2:353048

[30] V. Pacelli, "An Artificial Neural Network Model to Forecast Exchange Rates," *Journal of Intelligent Learning Systems and Applications*, vol. 03, no. 02, pp. 57–69, 2011. [Online]. Available: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jilsa.2011.32008

[31] S. Albrand, J. Chapman, D. Cote, L. Fiorini, E. J. Gallas, V. Garonne, C. Gwenlan, P. Laycock, A. Klimentov, D. Malon, E. Torrence, G. Unal, T. Wenaus, J. Catmore, D. Charlton, L. Gossens, A. Nairz, D. Barberis, F. Gianotti, C. Guyot, R. Hawkings, I. Hinchliffe, B. Heinemann, A. Höcker, G. Lehmann, P. Nevski, and H. von der Schmitt, "ATLAS Dataset Nomenclature," CERN, Geneva, Tech. Rep. ATL-GEN-INT-2007-001. ATL-COM-GEN-2007-003, nov 2007. [Online]. Available: https://cds.cern.ch/record/1070318

[32] "Python NeuroLab documentation." [Online]. Available: https://pythonhosted.org/neurolab/

[33] "NTUP_SMWZ datatype description," 2012. [Online]. Available: https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/SMWZD3PD

[34] "NTUP_COMMON datatype description," 2014. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/CommonD3PD

[35] P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, no. 1, pp. 5–40, 1999.

[36] C. Dobre and C. Stratan, "MONARC Simulation Framework," *Proc. of the RoEduNet International Conference*, jun 2011. [Online]. Available: http://arxiv.org/abs/1106.5158

[37] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing*, vol. 2002. IEEE

Comput. Soc, 2002, pp. 352–358. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1029935

[38] a. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the SimGrid simulation framework," in *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.* IEEE, 2003, pp. 138–145. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1199362

[39] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies," *International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 403–416, nov 2003. [Online]. Available: http://hpc.sagepub.com/cgi/doi/10.1177/10943420030174005

[40] H. Song and X. Liu, "The microgrid: a scientific tool for modeling computational grids," *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, p. 53, 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1592766

[41] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, nov 2002. [Online]. Available: http://doi.wiley.com/10.1002/cpe.710

[42] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating data Grids: an extension to GridSim," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 13, pp. 1591–1609, sep 2008. [Online]. Available: http://doi.wiley.com/10.1002/cpe.1307

[43] "Simpy Documentation," 2016. [Online]. Available: https://simpy.readthedocs.io

[44] M. Barisits, E. Kühn, and M. Lassnig, "A hybrid simulation model for data grids," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on.* IEEE, 2016, pp. 255–260.