FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Hybrid simulation models for data-intensive systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Martin-Stefan Barisits
Registration Number 0326059

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ.-Prof. Dipl.-Ing. Dr. eva Kühn

The dissertation has been reviewed by:

(Ao. Univ.-Prof. DI Dr. eva Kühn)	(Univ.-Prof. DI Dr. Erich Schikuta)	(Priv.-Doz. Dr. Sandford Bessler)

Wien, 01.03.2017

(Martin-Stefan Barisits)

Erklärung zur Verfassung der Arbeit

Martin-Stefan Barisits
Färbermühlgasse 13/4/11, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

This work would not have been possible without the support and dedication of several people.

First, I would like to thank *eva Kühn* for her enduring support and confidence in me. The physical distance between Vienna and Geneva was sometimes challenging. Thus I am extremely grateful that eva never hesitated to take time out of her busy schedule, to meet with me or talk to me on a Skype call. At the same time she always gave me the time I needed, when my other responsibilities at CERN did not allow my full dedication to my thesis. Her thoughtful guidance and feedback made this thesis possible.

Second, I would like to thank my supervisor at CERN, *Mario Lassnig*. Mario's motivation and precise feedback helped me to not only stay focused but also to push for higher goals. Our innumerable discussions lead to many fruitful ideas and were crucial to produce my best possible work.

Furthermore I would like to thank my colleagues at CERN. Lots of interesting discussions, many of which lead to successful publications, were made with *Vincent Garonne, Angelos Molfetas, Cedric Serfon, Thomas Beermann*, and *Ralph Vigne*. Particular gratitude goes to *Armin Nairz*, my section leader at CERN, who always supported me and gave me the time needed to finish this thesis. I also want to thank *Stefan Craß*, my colleague at TU Wien, who often helped in bureaucratic endeavors which were difficult due to my distance to Vienna.

I also want to thank my fiancée Carolina whose love and support helped me immensely to stay motivated and focus on my research.

Finally I would like to thank my parents for the faith they always had in me as well as for their support which made my thesis possible.

Abstract

Data-intensive systems are used to access and store massive amounts of data by combining the storage resources of multiple data-centers, usually deployed all over the world, in one system. This enables users to utilize these massive storage capabilities in a simple and efficient way. However, with the growth of these systems it becomes a hard problem to estimate the effects of modifications to the system, such as data placement algorithms or hardware upgrades, and to validate these changes for potential side effects.

This thesis addresses the modeling of operational data-intensive systems and presents a novel simulation model which estimates the performance of system operations. The running example used throughout this thesis is the data-intensive system Rucio, which is used as the data management system of the ATLAS experiment at CERN's Large Hadron Collider.

Existing system models in literature are not applicable to data-intensive workflows, as they only consider computational workflows or make assumptions which do not hold for operational systems. A hybrid modeling approach is proposed which addresses the limits of these models. It partitions the system into discrete components, creates models for these components, and combines them into one concise system model. However, each component model is only built on observed data metrics, such as system traces. The identification of which system components to model and which ones to omit is based on a quantitative system analysis of the Rucio data-intensive system. The storage, network, data integrity validation, and services components were identified. An existing model from literature was utilized for the network component. For the other components models based on machine learning techniques are created and evaluated against historic workloads from the running example. The component models are unified in an

event simulator and evaluated agains historic workloads from the Rucio data-intensive system. The median relative evaluation error of the hybrid system model is demonstrated with 22%.

Kurzfassung

Datenintensive System werden benutzt um massive Mengen an Daten zu speichern und um auf diese zuzugreifen. Diese Systeme vereinen die Speicher Ressourcen multipler Datenzentren welche üblicherweise über die gesamte Welt verteilt sind. Dies befähigt die Benutzer des Systems die Ressourcen in einer einfachen und effizienten Art und Weise zu benützen. Der Wachstum und die Größe dieser Systeme macht es allerdings äußerst schwierig die Auswirkungen von Änderungen am System, wie zum Beispiel Daten Platzierungs Algorithmen oder Hardwareanpassungen, vorherzusagen.

Diese Dissertation behandelt die Modellierung von operativen datenintensiven Systemen und präsentiert ein neuartiges Simulationsmodell welches die Leistung von Systemoperationen vorhersagt. Als Anwendungsbeispiel dient das datenintensive System Rucio, das als Datenverwaltungssystem für das ATLAS Experiment an CERNs Large Hadron Collider verwendet wird.

Existierende Systemmodelle in der Literatur sind nicht anwendbar auf Abläufe von datenintensiven Systemen, da sie hauptsächlich rechenbetonte Abläufe modellieren oder Annahmen treffen, die auf operative Systeme nicht zutreffen. Diese Dissertation präsentiert einen hybriden Modellierungsprozess der die Limitierungen existierender Modelle löst. Der Prozess partitioniert das datenintensive System in verschiedene Komponenten und erstellt Modelle für jede einzelne Komponente. Diese Modelle werden dann zu einem Systemmodell verbunden. Die Komponenentenmodelle basieren allerdings nur auf global zugänglichen Informationen, wie Ereignis Protokolle von Systemen, da dies die einzigen Informationen sind die gemein für datenintensive Systeme verfügbar sind. Die Identifikation der zu inkludierenden Komponenten basiert auf einer quantitativen Analyse des Rucio Systems. Die Speicher-, Netzwerk-, Datenvalidierungs-

und Servicekomponente werden dabei identifiziert. Für die Netzwerkkomponente wird ein existierendes Modell, aus der Literatur, verwertet. Für die anderen Komponenten werden Modelle, basierend auf maschinellem Lernen, erstellt. Alle Modelle werden mit historischen Workloads des Rucio Systems evaluiert. Die Komponentenmodelle werden in einem Simulator implementiert und anschließend gegen historische Workloads des Rucio Systems evaluiert. Der Median des relativen Evaluierungsfehlers des hybriden System Modells ist 22%.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Defining the hybrid simulation model

CHAPTER 1

Introduction

1.1 Motivation

In 2012 at the world economic forum *Big Data, Big Impact* in Davos, Switzerland a report was presented that declared data as a new class of economic asset, similar to crude materials or currency [Vital Wave consulting, 2012]. Data is becoming more available and more integrated in modern society but the challenge of storing, organizing, and accessing vast amounts of data amplifies as well. The term *big data* is commonly used to describe datasets of a size so large that conventional data processing systems are unable to handle them. Systems specifically designed to handle these extreme amounts of data are called *data-intensive* systems and are usually highly distributed.

The definition of data-intensive systems changed rapidly over the years. According to Hilbert and Lopez [2011] the combined capacity of storage in the world in 1986 was about 2.6 exabytes, optimally compressed, which grew to 295 exabytes, optimally compressed, in 2007. In 1986 a system would be considered data-intensive if it manages more than a gigabyte of data, nowadays in 2015 a big data-intensive system would have about the same storage capacity as the total storage capacity of the world in the year 1986. With the rapid growth of data the concentration of big amounts of data in single data-intensive systems also grew significantly. Most people are familiar with products like Twitter [Twitter Inc., 2015], Youtube [Google Inc., 2015],

Facebook [Facebook Inc., 2015] or Dropbox [Dropbox Inc., 2015]. Without these data-intensive systems life would be very different. But there are also numerous data-intensive systems in domains which are less in the focus of the public eye, like economics and finance, health, as well as science.

With the rise of these massive systems technological challenges arise as well. It becomes more difficult to estimate the characteristics of a system. In 1986 it was moderately simple to estimate the time to process a certain sized dataset on a hard drive. However, modern data-intensive systems are distributed over multiple continents and involve thousands of computers, hence it becomes a very complex undertaking to estimate how these systems behave. This creates hard problems such as optimizing scheduling, data placement algorithms, or planning hardware upgrades. It is unclear if any changes result in the expected performance and if there are any unwanted side effects. There are several approaches for software systems to estimate the effects of changes like software test-cases, code deployment in pre-production testbeds, as well as simulation. But these approaches are very often limited, non constructive, or even impossible in the domain of data-intensive systems.

This thesis presents a novel approach to model data-intensive systems under these restrictions. The model is able to capture the performance-relevant characteristics of a data-intensive system and can be used to estimate the performance of system operations.

1.2 Problem statement

With increasing size and complexity it becomes very difficult to estimate the outcome of even tiny changes to the system. Nonetheless, developers and operators have to adapt their system continuously to guarantee a smooth and efficient experience for their users. This user experience stands at the center of this thesis: How long does it take for a user or application to read or write data in the system. Of course there are also other factors, like internal peculiarities such as resource consumption, costs or code complexity, which are interesting for the developers and operators, but the external operational performance of the system is the most important characteristic for the user as well as the developer and operator. Therefore this thesis focuses on

system modeling of operational data-intensive systems, with a case study from a grid computing system. The research question posed by this thesis is:

Can a model be constructed that estimates the response time of user operations in a data-intensive system?

Consequently, the research objective of this thesis is to create a model for operational data-intensive systems which is able to predict the response time of user operations. The model must be able to take an operational workload as an input and estimate the response time of individual operations in the workload. With this model researchers can propose changes to the data-intensive system, such as data placement or data selection algorithms, and investigate the effect of the proposed changes to the response time of write and read operations. Models in literature are focused on studying resource consumption like disk and bandwidth usage but do not, or only inaccurately, give an estimation about the performance changes for the end user of the system.

The field of modeling data processing systems is essentially as old as computer science itself hence there exists a wide area of related work in process modeling, disk drive modeling, and network modeling. However, when it comes to large-scale distributed systems the literature is more limited and in the area of data-intensive systems there are gaps which are filled by this thesis. Certain assumptions are made in literature on distributed systems which raise concerns about their applicability to operational data-intensive systems. These assumptions are also discussed in this document. Specifically this thesis addresses the following problems which are not yet sufficiently answered by existing work:

Data-intensive workflows

One of the key-requirements of a data-intensive system model is to be able to estimate the response time of an individual *read/write* operation. However the vast majority of literature focuses on computational use cases and thus makes certain assumptions and simplifications, such as instead of estimating response times the model is classifying operations as fast or slow. Most often the access times are not estimated at all, as they are considered negligible when investi-

gating computation workflows. Are these simplifications valid when modeling the workflows of data-intensive systems? If not, what can be done to actually estimate the operation response time? Which components of a data-intensive system have to be modeled at all and which ones can be considered negligible?

High number of events

The operational workloads of data-intensive systems consist, even in a short time-frame, of a very high volume of events. This is different to computational systems, as a single computational job can involve thousands or even hundreds of thousands of events which are usually abstracted as one computational event. Thus, a data-intensive workload is orders of magnitude larger than a workload of a computational distributed system. How does this influence the methodology when modeling a data-intensive system? Is it even possible to construct models on such large input datasets and how does sampling influence the accuracy of the final model? Are there stochastic attributes of the workload which have to be represented in the sample to generate an accurate model? Also, once the model is created, is it able to process input workloads of that size and predict the response times?

Requirements for modeling operational data-intensive systems

How is it different to create a model for a data-intensive system compared to a conventional data processing system. Does the fact that the system is operational change the modeling approach? Are the input requirements to data-intensive system models distinctively different compared to other distributed systems? Do the same assumptions about accessible information and system knowledge hold for data-intensive systems? Can system characteristics be extracted from an operational system and which ones are needed for a model? Are operational measurements representative enough compared to specifically injected measurement workloads?

Modeling of data-intensive system components

For the performance critical components of data-intensive workflows independent models have to be investigated and created. There are a number of modeling techniques from related areas

in network modeling, disk drive modeling, or the modeling of relational databases. Are these approaches from related domains applicable to data-intensive component models? If not, what are more appropriate modeling techniques and how do they perform?

Unifying models in a hybrid system model and integration in an event simulator

The independent models have to be unified into a hybrid data-intensive system model. What provisions have to be made to be able to switch specific component models without invalidating the hybrid model? How can the component models be plugged into the hybrid model, while still keeping a consistent global state? How can the model be integrated into an event simulator and used to predict full input workloads of a system?

1.3 Outline

The remainder of this thesis is organized as follows: Chapter 2 introduces the operational data-intensive system Rucio, which acts as running example of this thesis. To ease the content of the thesis, this chapter also gives a brief overview of the data grid reference architecture presented by Foster et al. [2001] and builds the connections between the reference architecture and the running system example. The section also discusses how data grids, a special type of data-intensive system, compare to other types of data-intensive systems.

Chapter 3 describes the related work in modeling and simulating data-intensive systems. This chapter only treats full-system models, individual component models are discussed in their respective chapters.

Chapter 4 presents the novel hybrid simulation model proposed by this thesis. This chapter gives a general introduction on modeling and simulation and then establishes the input requirements for modeling operational data-intensive systems. The chapter then explains the principal idea and concepts of the hybrid simulation model. Based on a statistical analysis of the workflows and historical workloads of the operational data-intensive system Rucio, a selection of components is done whose component models are part of the hybrid system model. The specifics of these component models are presented in chapters 5 to 8.

Chapter 5 first details related work on how storage systems are modeled. The chapter then continues by investigating different machine learning techniques to create black-box models for storage systems in data-intensive systems. Each technique is carefully evaluated based on historic workloads.

Chapter 6 investigates existing techniques for modeling network systems. Specifically the question of how to model large-scale networking systems, transferring very large volumes of data, is addressed here.

Chapter 7 details how typical data integrity validation workflows, based on checksum algorithms, can be accurately modeled.

Chapter 8 continues with the different service components, identified in Chapter 4. Techniques to create accurate performance models for these service workflows are investigated and evaluated.

Chapter 9 details how the component models are combined in a unified hybrid system model. The chapter continues with a comprehensive evaluation of the hybrid data-intensive system model by replaying historic workloads in the event simulator. This chapter gives a clear picture of the performance of the model and also details its limitations.

The thesis finishes with a summary and a conclusion in chapter 10. This includes relevant work that arose during the research carried out for this thesis.

CHAPTER 2

Overview of the ATLAS data-intensive system

2.1 Introduction

As the emphasis of this thesis is on the modeling of an *operational* data-intensive system, a running example is needed to conduct and apply the research. This running example is *Rucio*, the distributed data management system of the high-energy physics experiment ATLAS at the Large Hadron Collider [ATLAS Collaboration, 2008]. In Section 2.2 a general overview of Rucio is given. As the architecture and workflows of Rucio are quite specialized, a clear mapping of the system to the widely used data grid reference architecture is given in Section 2.3. This approach prepares the content so that the findings and analysis are applicable and adaptable to other systems. As data grids are a specific type of data-intensive systems, section 2.4 details the similarity and differences between data grids and other data-intensive systems. The section continues in presenting overviews of other data-intensive systems.

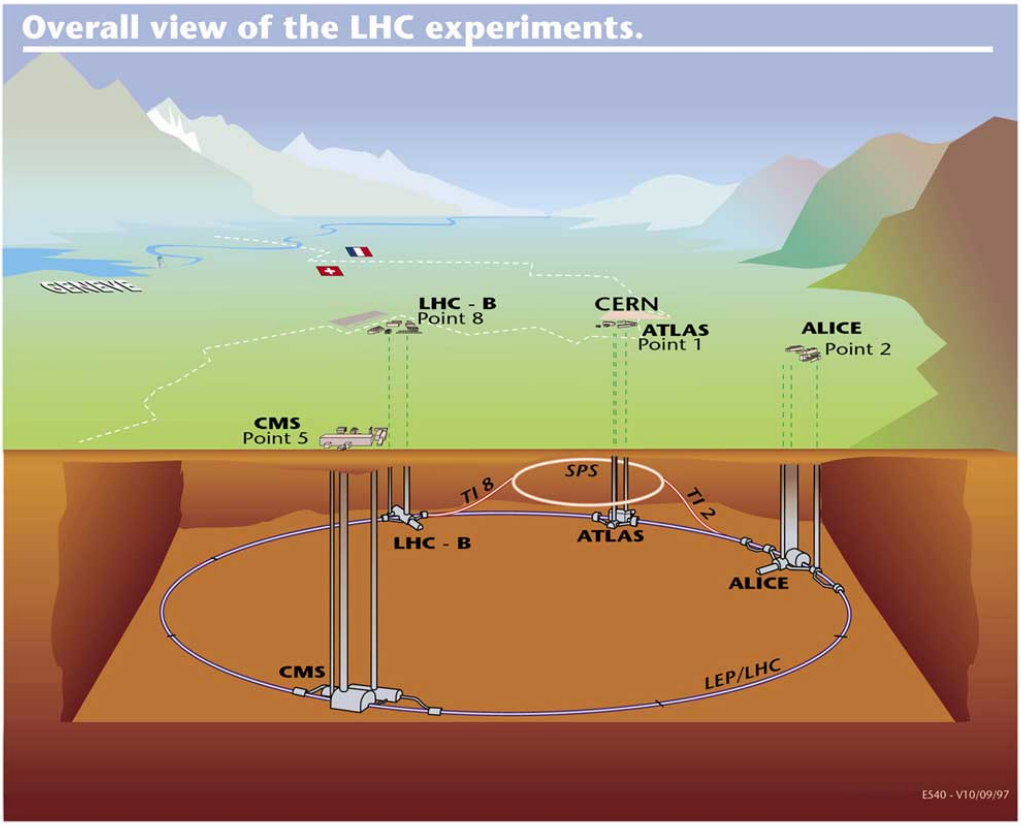

Figure 2.1: Overview of the Large Hadron Collider (LHC) experiments and facilities (Copyright 2011 CERN).

2.2 Rucio

Rucio [Garonne et al., 2012, 2014] is the distributed data management system of the high-energy physics experiment ATLAS, at the *Large Hadron Collider* (LHC) located at the *European Organization for Nuclear Research* (CERN) in Geneva, Switzerland. It is ring-shaped, measures 27km in circumference, and is buried 100m underground, as depicted in Figure 2.1. ATLAS (*A Toroidal LHC ApparatuS*) is one of two general purpose detectors used to investigate particle collisions at the LHC. The detector consist of several sub-detectors shown in Figure 2.2 which observe head-on proton-proton collisions at extremely high energies. The experiment studies these collisions in hope of making discoveries about the basic forces that shape our universe. Possible areas of investigations include extra dimensions of space, unification of fundamental

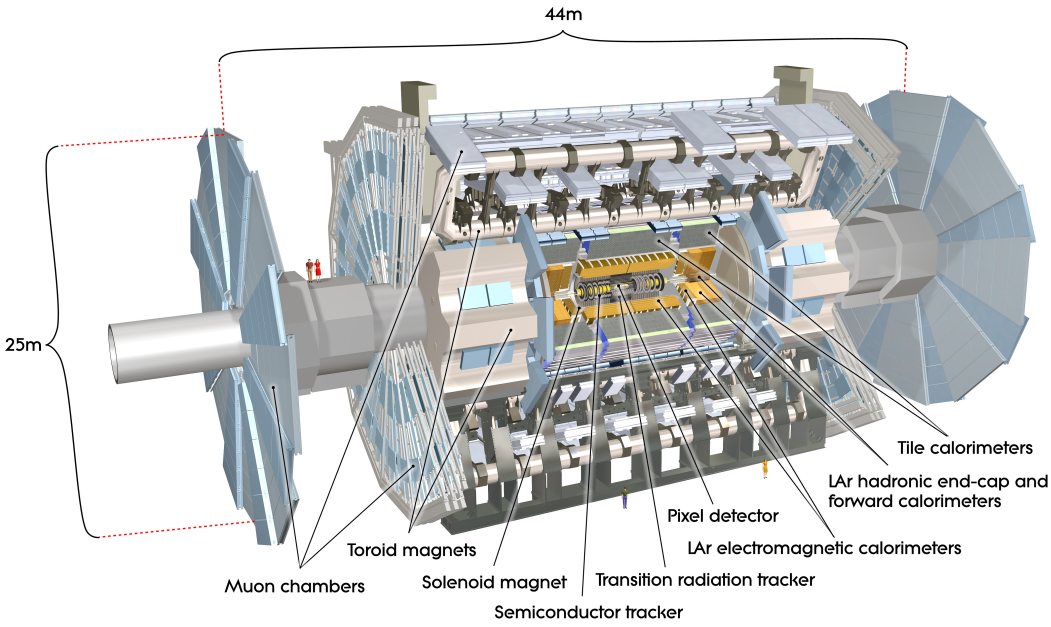

Figure 2.2: Overview of the ATLAS detector and its sub-detectors (Copyright 2008 CERN).

forces, or evidence of dark matter in the universe. ATLAS is one of the two experiments that was involved in the discovery of a particle consistent with the Higgs boson in 2012 [Aad et al., 2012]. This discovery ultimately led to the physics nobel prize in 2013 for Peter Higgs and François Englert, who proposed the mechanism in 1964. The author of this thesis is one of the core software engineers included in the development of the Rucio system.

Rucio is also used as the distributed data management system of the *Alpha Magnetic Spectrometer* (AMS). AMS is an experiment designed to measure antimatter in cosmic rays and to search for the evidence of dark matter [Aguilar et al., 2002]. The AMS Module is attached to the International Space Station to which it was carried by the space shuttle Endeavour on flight STS-134 on May 16, 2011 and was installed 3 days later.

The ATLAS detector records huge amounts of data which have to be stored and processed by the collaboration in search for physics discoveries. Due to the amount of data produced by ATLAS a data-intensive system is needed. *Rucio* is responsible for organizing the entirety of the collaboration's data. At the time of writing this thesis, this is about 250 Petabytes of physics

data. The data is stored on over 750 storage endpoints in more than 150 data centers worldwide which are connected by the Worldwide LHC Computing Grid [Bird et al., 2008] (WLCG). Each site provides storage resources, like tape libraries, disk pools, distributed hierarchical file systems, or other large-scale storage systems as well as computing resources like CPU or GPU farms. The reason for this highly distributed infrastructure mostly lies in cost issues as well as to increase the availability of resources. A single large concentration of resources is usually not possible with the type of funding of large multinational consortiums. The ATLAS collaboration's participating institutions pledge these resources, under certain service level agreements, to the management of the distributed system. The distributed computing paradigm of dispersing resources geographically and facilitating them using middle-ware software is called a data grid [Allcock et al., 2005]. The job of the middleware is to organize the data and to coherently use the storage resources while enabling the user to interact with the system in a simple and efficient way.

Data grids are organized in different ways which are dominated by the data access workflows or by physical limitations of the hardware. In ATLAS the sites are categorized in four different tiers:

Tier-0 is the CERN computing centre. Data read directly from the detector is processed by a chain of triggering systems and the derived, as well as raw, data products are stored in large-scale tape and disk storage systems directly at CERN.

Tier-1 sites are typically large national laboratories pledging a significant amount of storage and computing resources. Currently there are 10 Tier-1 sites, seven in Europe, two in the United States and one in Asia, which are all connected to each other and CERN by dedicated high-capacity networks. Tier-1 sites are primarily used to store a fraction of the raw data for availability reasons and to perform large-scale reprocessing of the raw data to derive data products.

Tier-2 sites are generally medium sized universities or laboratories usually associated to some Tier-1 site. Typically the Tier-2 sites are connected to a single Tier-1 by national scientific networks. Tier-2 sites provide the storage and processing capacities for specific research groups within the collaboration. Also Tier-2 sites are heavily used to generate simulated data using

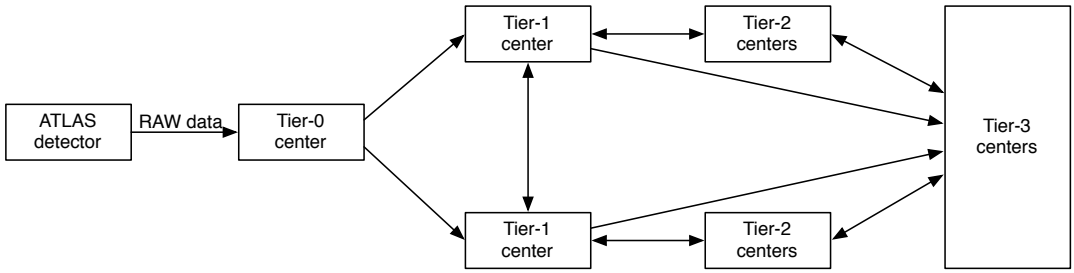

Figure 2.3: Overview of the ATLAS topology and data workflow.

Monte Carlo methods, and for storing and processing user analysis data.

Tier-3 sites are smaller universities providing a small amount of resources without any guarantees of availability or fault-tolerance. These sites are usually not part of official data replication policies, but pull data to their site for specific user analysis purposes.

Figure 2.3 denotes the workflow of ATLAS data. This shows that resources in a data grid are not only different due to their heterogeneity in technology, but also due to different usage caused by the experiment's workflows. These workflows are controlled by policies integrated in the middleware system. The policies guarantee the availability of data as well as the access performance of the data, as different data products are considered with different importance by the collaboration. For a data-intensive system model it is important being able to represent the policies defining these workflows and to be flexible in adapting to all types of topologies and data workflows.

Concept & Workflows

Every data-intensive system has, to some extent, unique workflows and concepts. This section gives a brief overview of the concepts and workflows used in Rucio. In section 2.3 the mapping of these concepts and workflows to the data grid reference architecture is made. This is important in order to describe a generic data-intensive system model which is capable to model systems universally without restricting itself to specific concepts.

In the center of Rucio are the concepts of accounts, files, datasets, and storage systems. *Accounts* represent individual users, a group of users, or a centrally organized production activity.

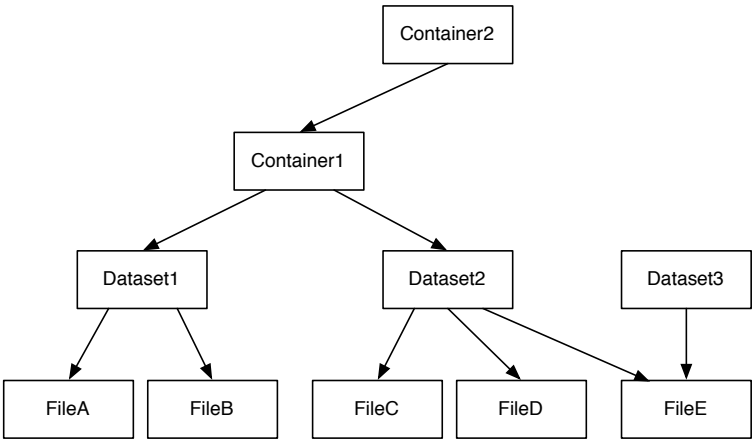

Figure 2.4: File, dataset and container aggregation hierarchy in Rucio.

Accounts are assigned privileges and access credentials such as X509 certificates or Kerberos tokens.

Files are the smallest concept of data in Rucio. Each file is addressable by a unique filename called a *data identifier*. However, physicists rarely interact with a single file but with a group of files, which in Rucio are called *datasets*. Datasets are also uniquely addressable by a data identifier and can be resolved to a specific set of files. Datasets can be grouped into *containers* which, further on, can also be grouped into other containers as well. Again, each container is addressable by a unique data identifier as well. The aggregation hierarchy of Rucio is shown in Figure 2.4.

For data discovery and lookup, each file, dataset and container can be assigned with metadata attributes. There are two groups of attributes, physics attributes for example the number of collision events and the physics run number, and system attributes such as size, creation time, or checksums.

Storage resources are abstracted in Rucio by a concept called a *Rucio Storage Element* (RSE). A storage element can be anything from a tape library system, a disk pool, a distributed file system, or even just a specific directory on a file system. Each RSE is associated with a set of access protocols, policies for deletion like thresholds and quality of service attributes like planned downtimes. Also metadata attributes, for example `country=us` or `tier=1` can be

assigned to storage elements. Rucio uses a formal language combining set operators and meta attributes to address a set of storage elements [Barisits et al., 2014]. This language can be used to express data placement to a specific set of storage elements.

A file physically located at a specific storage element is called a *replica*. Each replica is also associated with a additional attributes like creation time or the physical file path. In most cases a deterministic function is used to generate the physical file path at a storage element. This allows clients to access files at a storage element without the need of resolving the physical file path at the Rucio server. However, certain storage elements like tape libraries do require a non-deterministic file path to work efficiently.

Another concept in Rucio are *replication rules*, which are used for three different purposes. Replication rules are associated to a data identifier and define the number of replicas which should be created on a certain set of storage elements. Thus, replication rules are used to initiate data replication. If necessary, multiple replication rules can be defined for the same data identifier, thus they also express the concept of multiple ownership of data. Every rule is interpreted as an account expressing their interest in this data. The third purpose is data deletion: As long as there are replication rules affecting a file replica, the file replica will not be deleted by the system. Once the last replication rule is removed, the replica is marked as being available for deletion. The actual deletion is then depending on the storage policies of the storage element, such as if certain occupancy thresholds are reached.

The second concept used in Rucio to control data placement are *subscriptions*. Subscriptions are used to automatically create replication rules for data which does not exist yet. A subscription is a set of criteria for meta data which are checked against every newly created data identifier. If the data identifier matches the criteria of the subscription, a replication rule is set for the data identifier.

Architecture

Rucio is designed as a distributed architecture with four sections: Client, server, daemon and storage resources. See Figure 2.5 for an overview. The user can access Rucio by three different ways: by using a python API or command line client, by directly making REST calls to the

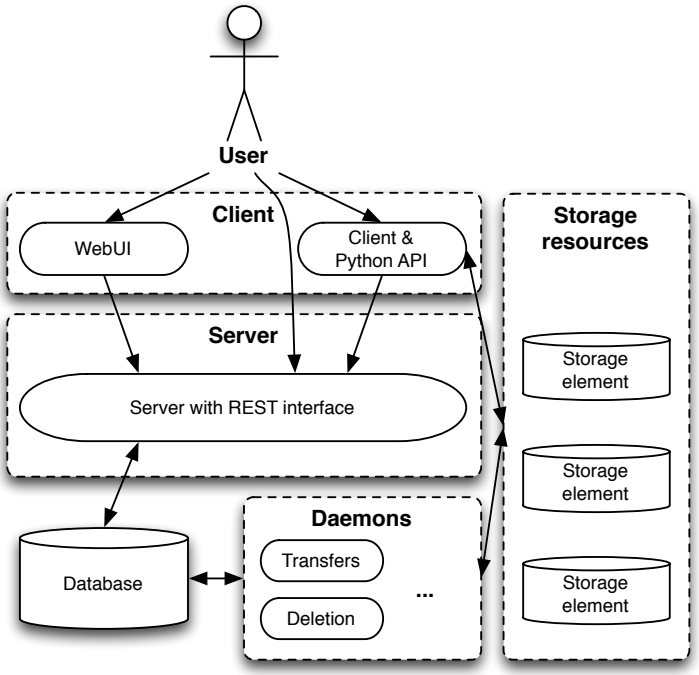

Figure 2.5: General overview of the Rucio architecture.

server or via the Rucio web user interface. All calls to the client are transformed into HTTP request which are sent to the servers REST interface [Richardson and Ruby, 2008]. The Rucio servers are listening for calls. Once a call is received the account is authenticated based on the provided client certificates. On successful authentication the client receives a token which is valid for one hour and can be used for subsequent requests.

The separation between daemons and servers is guided by one principle: Potential workload intensive requests are only acknowledged by the server but the actual execution is done asynchronously by the daemons in the background. This ensures a smooth user interaction and avoids the congestion of the servers. Also recurrent actions, for example consistency checks, are executed by the daemons. There is generally no direct interaction between storage resources and the server, as these interactions are also potentially time intensive.

The servers are written in Python [Python Software Foundation, 2015] and the architecture is designed in a horizontally scalable way. For each physical host an Apache webserver is

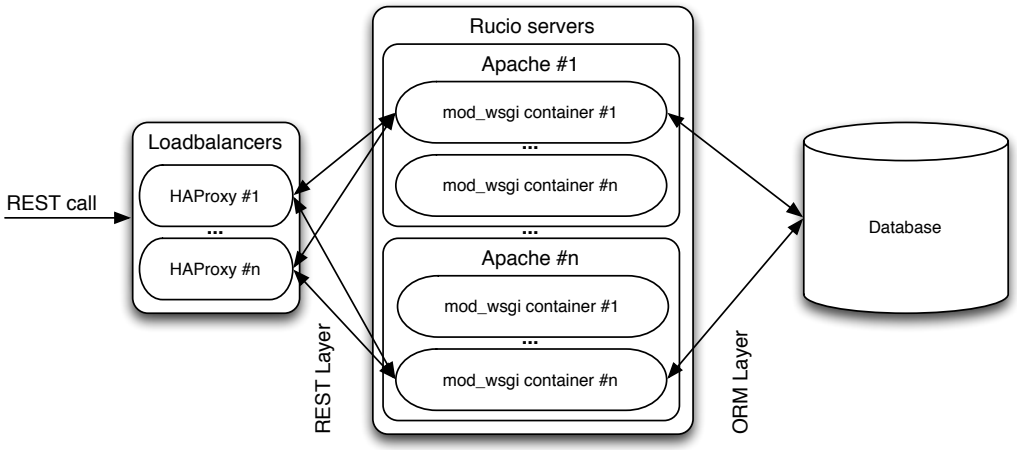

Figure 2.6: Distributed architecture of the Rucio servers.

started which initializes a number of mod_wsgi containers [Apache Software Foundation, 2015] each hosting a single Rucio server instance. The distributed architecture of the Rucio servers is detailed in Figure 2.6. Each client-generated HTTP REST request is sent to a DNS address which resolves to multiple loadbalancers. Each loadbalancer is based on the HAProxy software [Tarreau, 2015]. The loadbalancer does the SSL handshake and relays the request to a random Rucio server based on the least-connection scheduling algorithm. The Rucio servers then undertake the necessary communication with the database. The return-stream of the HTTP request is channeled back through the loadbalancer to the client. Thus the architecture is horizontally scalable both on the loadbalancer as well as on the Rucio server layer.

The database consists of two instances, one serving as standby instance and thus this layer is not fully horizontal scalable.

The Rucio backbone consists of a set of horizontally scalable daemons all assigned with a specific task.

The *Judge* is the daemon in charge of the evaluation of replication rules. Once a replication rule is requested the daemon checks the necessary account quotas and permissions, and then parses the replication rule expression, and creates a set of transfer requests as well as logical file replicas on the database. Once a replication rule is deleted or expired, the daemon updates the quotas as well and flags the according replicas for deletion.

The *Conveyor* is the file transfer daemon of Rucio. It continuously picks up file transfer requests from the database, selects the optimal source replica, and executes file transfers between the source and destination storage element. It also regularly checks the progress of ongoing transfers, and verifies successful transfers with checksum comparisons. Once a transfer was successful the daemon is also responsible for updating the respective replicas and replication rules on the database.

For the deletion of data there are two different types of daemons: The *Reaper* is responsible for physically deleting replicas on the storage elements. Based on deletion policies it selects storage elements which are in need of storage space and deletes previously flagged replicas by directly interacting with the storage elements. The second deletion daemon is the *Undertaker* which acts on the logical expiration of datasets and containers. When a data identifier expires, all its assigned replication rules are removed and the containing replicas are marked for deletion. This ensures that data with a very limited lifetime is removed from the system to free up storage resources.

There are also a set of secondary daemons in charge of consistency checks and data recovery, message exchange with other applications, quotas and usage statistics as well as a tracing daemon which records the system and data access performance for subsequent analytical use cases.

Scale

The core part of Rucio, not including storage systems and databases, runs on about 60 4/8 core machines with 8/16Gb of memory. There are 4 load balancers, 15 Apache frontend servers, 28 daemon nodes, as well as several nodes for administration and monitoring. The amount of managed resources is shown in Table 2.1 while the evolution of total managed data is shown in Figure 2.7. The system manages 730 logical storage elements which map to about 230 physical storage systems.

The evolution of the amount of globally managed data shows a linear increase in data until the technical stop of the LHC in the beginning of 2013. During the 2 year technical stop no new data was recorded from the detector, however, simulation data was generated as well as

Entity	Volume
Accounts	5308
Files	~0.6 billion
Datasets	~7.7 million
Replication rules	~7.5 million
Replicas	~0.65 billion

Table 2.1: Volumes of selected system resources of Rucio.

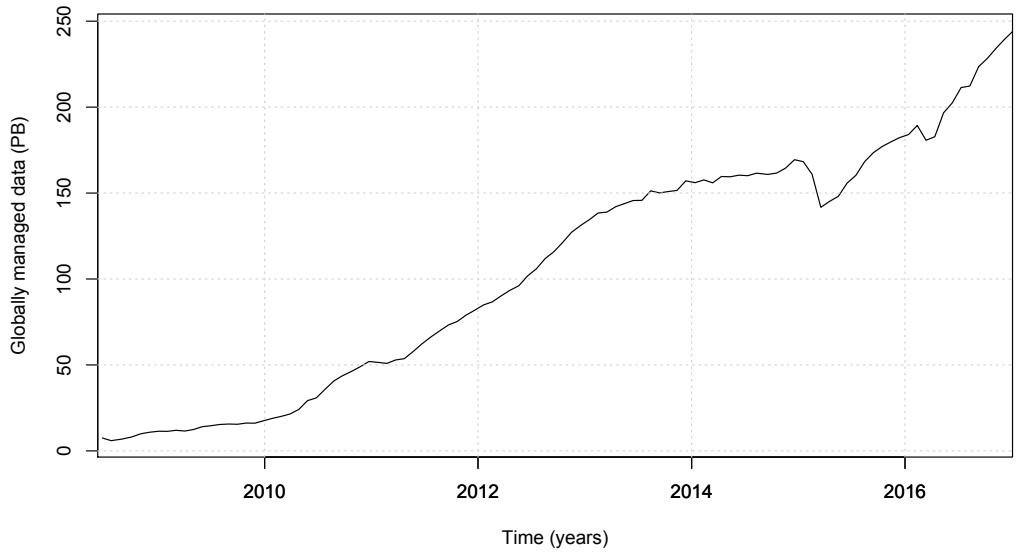

Figure 2.7: Evolution of total data volume managed by Rucio.

reprocessing campaigns of already collected data were conducted. At the end of the technical stop in the beginning of 2015, a major deletion campaign removed almost 30 PB of data. Since the start of the LHC for Run-2 a linear increase in data can be observed again. The frontend request rate and response time of Rucio, measured at the loadbalancers is shown in Figure 2.8. This plot shows a typical week of interactions. The typical workload of Rucio is rather spiky, averaging at 180 Hz frontend request rate and about 120 ms response time. The transfer and deletion rate, shown in Figure 2.9, is similarly spiky. This is both due to the characteristics of the workload and the bulk-handling of requests in the system. The plots show that Rucio nearly

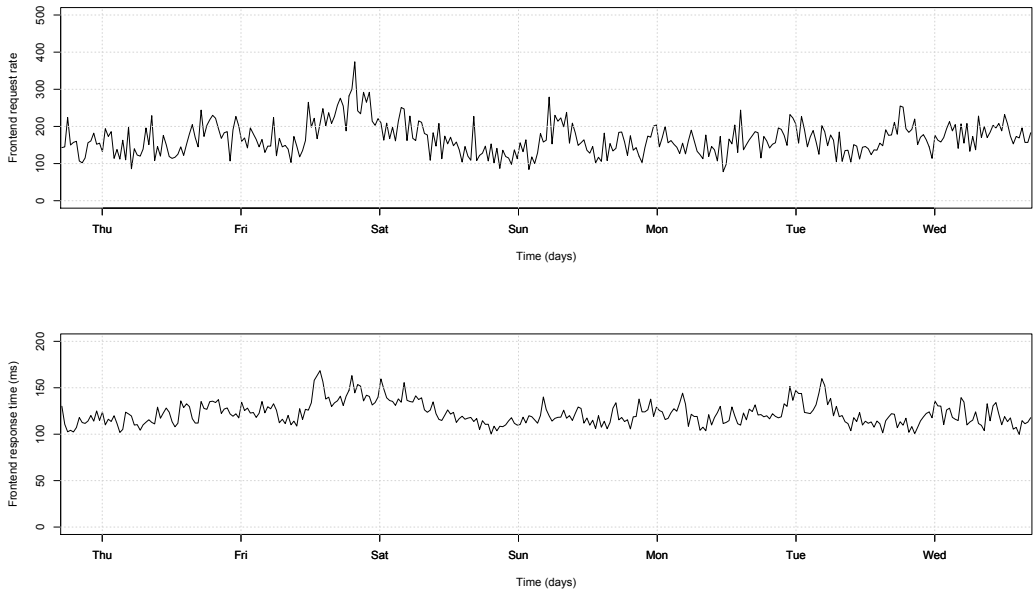

Figure 2.8: Frontend request rate and frontend response time of Rucio.

deletes twice as much files as it transfers in between storage systems. The reason for this is that intermediate data products are temporarily stored at a storage system but only read locally at the data center, thus a WAN transfer is never executed for these files.

2.3 The data grid reference architecture

Foster et al. [2001] describe a grid reference architecture by identifying the general components of a grid computing system. Based on this work Allcock et al. [2002] formulate the data grid specific reference architecture. The goal of their work was to describe the ”Grid problem” both from an architectural point of view, as well as the interactions between collections of individuals, institutions, and resources. Their work enabled researchers to describe their findings in a common framework, and to also make the bridge from data grids to other data-intensive systems. The layers and a partial list of components of the data grid reference architecture is shown in Figure 2.10.

The entire grid is decomposed into 5 layers which are further split into different components.

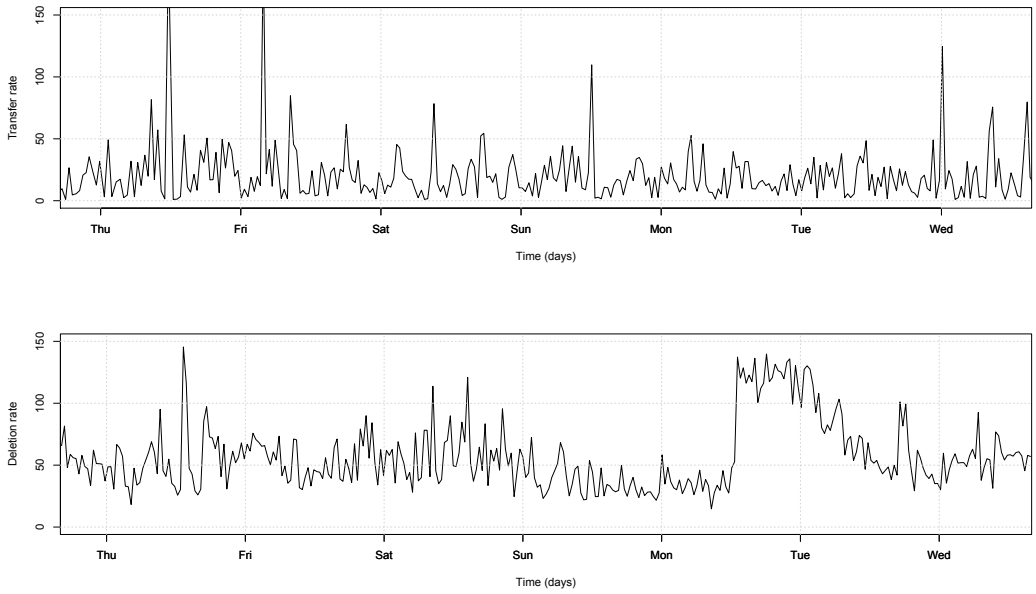

Figure 2.9: WAN transfer rate and deletion rate of Rucio (Files per second).

The bottom layer is the *fabric* layer consisting of all the basic hardware components of the grid. Storage systems, network systems, databases, both for *catalog* as well as *metadata* information, as well as computing resources for the management machinery of the distributed system. The computing resources described here are strictly to run the software of the data grid middleware and should not be confused with computing resources used by the user applications. The *connectivity* layer holds protocols responsible for the communication and authentication with elements from the fabric layer. At the *resource* layer are mostly protocols associated with managing individual resources. This can be common protocols like HTTP or proprietary protocols of certain vendors of storage resources. The *collective* layer of the architecture holds the actual high-level service architecture of the data gird middleware:

- The *Replica management* service is responsible for resolving file collections to files and for the bookkeeping of replica locations. Also the transfer management is part of this component. The service uses underlying data movement protocols from the resource layer which are independent from the replica management itself. In Rucio the core parts of the

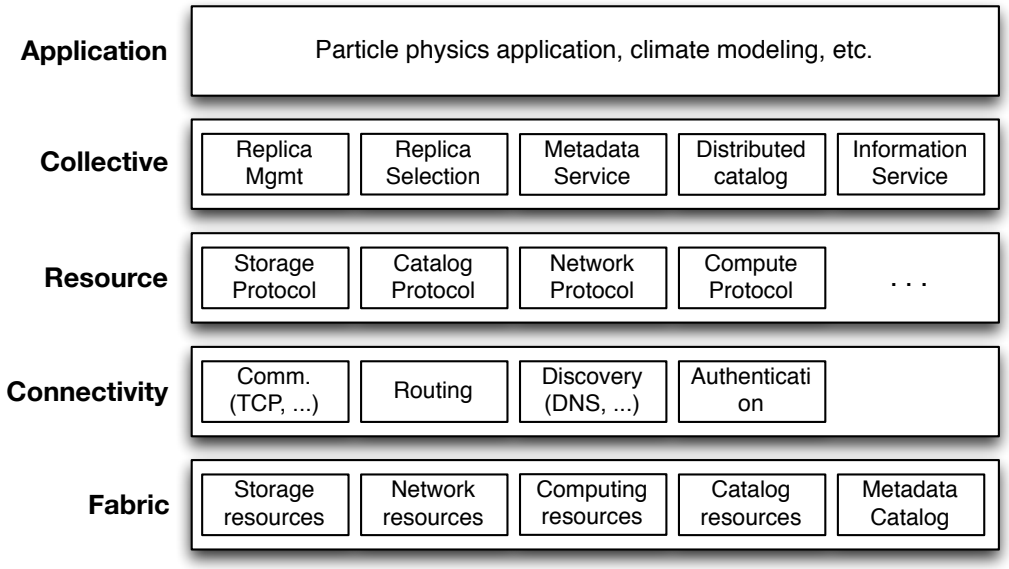

Figure 2.10: Data grid reference architecture by Allcock et al. [2002].

server, as well as the Judge and Conveyor daemons, are associated with this component.

- The *Replica selection* service is responsible for selecting the best source replica, when requesting data as well as selecting the best destination for data replication. In Rucio this is associated with the Judge daemon.

- The main reason for separating a *Metadata service* from the replica management is that metadata is often very application specific. Thus different workflows are used to create and query metadata. This is also the reason why there are different fabric resources for the metadata catalog, as the architecture could foresee different technologies, for example non-relational databases, to store metadata. In Rucio there are specific workflows in the core which are covered by the metadata service in the reference architecture but there is no separate metadata database, thus the same relational database is used for both replica information as well as metadata.

- The idea behind the *distributed catalog service* is to run a central replica catalog as well as local mirrors of the catalog for certain data partitions, for example one local catalog

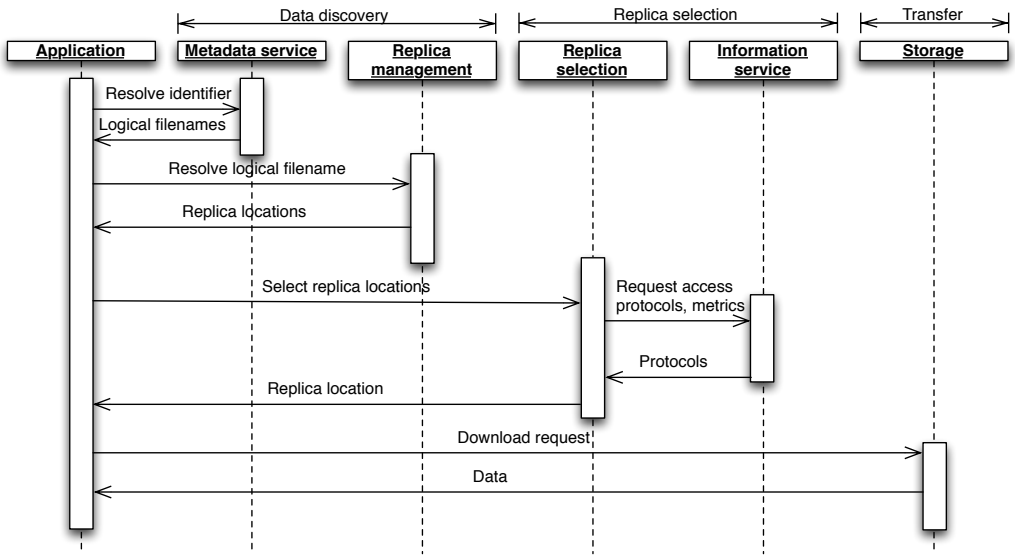

Figure 2.11: Data discovery and download workflow in the data grid reference architecture [Allcock et al., 2002].

per geographic continent. This concept has mostly historical reasons, as the file catalogs could not handle the amount of requests, thus they had to be partitioned for performance and reliability reasons. Rucio does not use distributed catalogs thus this component is not associated.

- The *information service* holds information about the performance of the system as well as the mapping of access protocols to storage resources. This component also exists in the server part of Rucio and is either stored in the same database as used by the catalog services or in external timeseries databases such as Graphite [Graphite Collaboration, 2016].

The mapping of Rucio components to their reference architecture counterpart is mostly quite direct but can slightly divert depending on the workflow. An example workflow of data discovery and download is shown in Figure 2.11. In this workflow the application or user wants to download a set of files associated to a collective data identifier. The Rucio equivalent for this is downloading a dataset. In the first step the data identifier or meta data query is resolved against the metadata service to receive a list of logical filenames. In the next step the replica manage-

ment service is queried to receive, for each logical file, a list of replica locations. In Rucio the last two requests are done in one atomic step, but internally still both interactions are executed in an optimized way. The replica selection service is queried next which internally queries the information service for current system state and metrics as well as storage protocols to select the optimal replicas. In Rucio these requests are received by the server and internally processed by the Judge component. The client application then receives the list of selected replicas and protocols and can then start the downloads directly from the storage systems.

Other data-intensive system, outside of the data grid domain, do not strictly stick to the data grid reference architecture. But in general the steps in any data-intensive system, like in cloud-, cluster- or partly super-computing, are very similar: data discovery, replica selection and transfer. In the remainder of this thesis the data grid reference architecture will be used to describe the findings of the investigated operational system. More detailed explanations of the internal workflows in Rucio are added when necessary.

2.4 Other data-intensive systems

There exists a vast amount of data-intensive systems, both in the research domain as well as coming from industry. However, as the core of the problem lies in processing the data efficiently, industry application providers are usually less open about their architecture and design choices as well as the scale of their systems, compared to other data-intensive systems from the open source and science domain. This section presents a concise introduction of some selected data-intensive systems next to Rucio.

Large Synoptic Survey Telescope

The Large Synoptic Survey Telescope (LSST) will conduct a 10-year survey of the nights sky starting in 2019 [Abell et al., 2009]. The telescope is a 8.4 meter large special three-mirror telescope located in north-central Chilè. It's mission is to produce images of the nights sky to look for evidence of dark matter, dark energy, and astronomical rapid movements.

The planned data production is 30 Terabytes of data per day. The expected volume over the

ten year lifetime of the telescope are 200 petabytes of image and derived data. The LSST will use a grid based solution for its data storage and analysis [Kantor et al., 2013] with multiple data centers across the American continent. The infrastructure is a close adaption of the historic design of the WLCG and the Rucio predecessor DQ2 [Branco et al., 2010].

National Centers for Environmental Information

The National Centers for Environmental Information (NCEI) is the world's largest active archive of environmental data [United States Government, 2015]. It is a merger of the National Climatic Data Center, the National Geophysical Data Center, and the National Oceanographic Data Center. The center stores an aggregated volume of 20 Petabytes of atmospheric, oceanic, costal, and geophysical digital data. The data comes from a wide variety of sources and is up to 150 years old. The data is stored in several mass storage systems across the United States but no direct links to the public are provided. Users have to explicitly request access to certain datasets which are either provided online or by shipping digital media to the requestor. This approach of controlling the access to the data as well as providing the data on offline sources significantly reduces the workload on the data-intensive system.

Google & YouTube

Google is very secretive about its infrastructure operation and actual system scale. In a recent investigation by Munroe [2015] where the author followed the hardware spending and power needs of Google, the storage capacity was estimated at about 10 Exabytes of data stored on hard disks in addition to another 2-5 Exabytes on tape storage. Most of Google's data is stored in their own proprietary data-intensive system, the Google File System (GFS), in at least 19 different data centers. Many of the companies software products or intermediate structured storage systems, such as BigTable [Chang et al., 2008] are built on top of GFS [Austin, 2009]. Due to the nature of Google products and the exposure to both the general public as well as commercial customers, the usage patterns are different than to other data-intensive systems. YouTube video views, for example, do not follow Zipf's distribution thus requiring different policies in terms of caching and replication [Cheng et al., 2008].

CHAPTER 3

Existing full system models

3.1 Introduction

This chapter covers the related work done in the area of modeling data-intensive systems. The models and simulation systems presented in this section only cover full system models. The related work for specific component models, or component models, as well as different modeling techniques, are discussed in the respective sections further on in this thesis.

Data-intensive systems can be categorized into a range of classes of systems and predictive models are usually made for a specific type of system. However, those models are most often also applicable to other classes of data-intensive systems, thus these should not only be seen in the context of their origin. As established in the introduction chapter, this thesis focuses on the modeling of response times of user operations of operational data-intensive systems, not on the execution time of the user applications instrumenting the data-intensive system. This difference is crucial as many system models focus on this part of the data-intensive system. For example, *High Performance Computing* (HPC) is one class of data intensive system, however, HPC systems are mostly dominated by the job execution of user applications as the direct interactions with the data-intensive system are local and therefore extremely fast. These models usually strongly simplify many parts of the interaction with the data-intensive system.

The focus of this thesis lies in distributed systems, where the interactions with the data-

intensive system take a non-trivial amount of time. In the spotlight of this chapter are models for computing- and data-grids, *peer-to-peer* computing, and *cloud* computing. Casanova et al. [2014] give an extensive overview of simulation systems and models of distributed applications and platforms which would exhaust the scope of this chapter.

This chapter continues by describing the most widely used simulation frameworks used to model grids, clouds and peer-to-peer systems.

3.2 Grid simulation frameworks

Over the last decade numerous models have appeared and disappeared on the scientific horizon. Usually these models serve a very specific purpose, for example measuring the relative difference between two scheduling algorithms, simulating the execution time of computing jobs, or predicting the storage utilization. There are strong differences in the modeling approaches and also especially in the effort of validating the respective models against an operational system. Only a few simulation systems passed the test of time and are still actively maintained or are considered mature. This section gives an overview of these systems.

GridSim

Buyya and Murshed [2002] first presented the GridSim simulation system in 2002. Over the years numerous extensions were published, with the latests extension for data grids being presented in 2008 [Sulistio et al., 2008]. The GridSim architecture is based on the discrete event simulation infrastructure of SimJava [Kreutzer et al., 1997]. SimJava is a process based simulation package where each component is represented as a process, and multiple processes act together to advance in simulation time and deliver events. The different components communicate with each other based on this events. An overview of the GridSim architecture is shown in Figure 3.1. The basic, event-based, simulation system of GridSim is built directly on top of SimJava and offers different toolkits, such as application modeling, resource entities, and job management. These toolkits are linked by resource brokers and schedulers to the simulation configuration. Each simulation configuration consists of a couple of entities, for example ap-

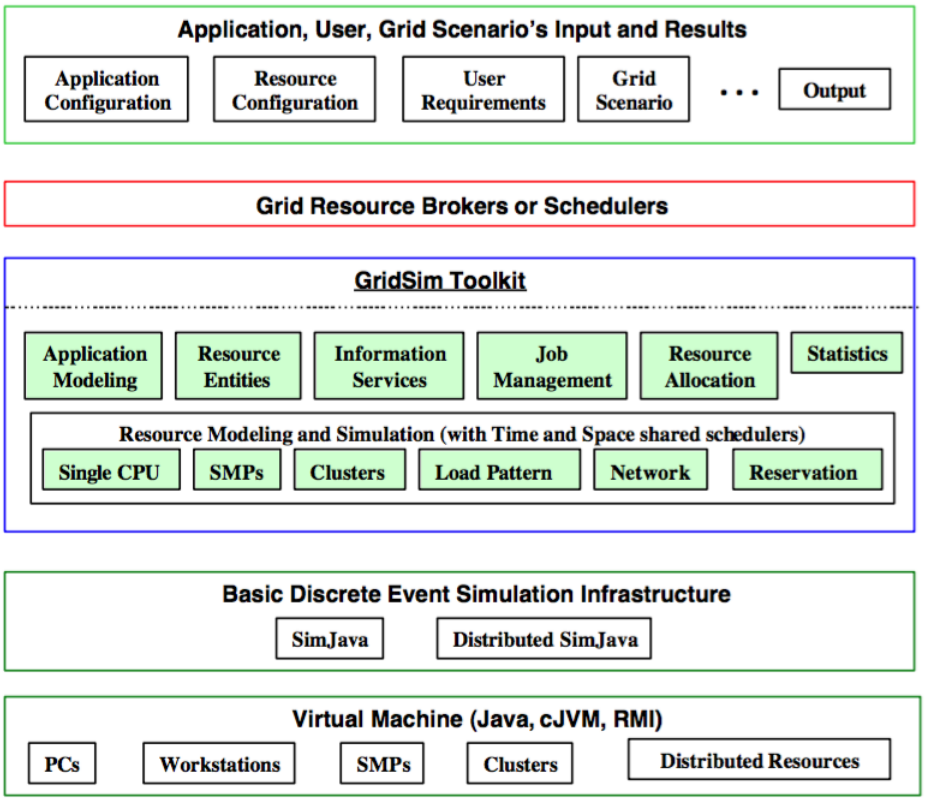

Figure 3.1: Overview of the GridSim architecture [Buyya and Murshed, 2002].

plication configuration, resource configuration, and user requirements. Once the simulation is started, the resource broker links the inputs of the configurations to the different toolkits in GridSim. These toolkits then use the high-level functions of SimJava, as well as the configuration parameters, to make predictions and report them back to the resource brokers. Despite the very wide usage of the simulator its network model was somewhat criticized as the validity limits have never been properly documented [Casanova et al., 2014]. Also the more recently published extension for data grids is a rather simplistic model, as it only allows a fixed seek time and fixed data transfer rate per storage element. Nevertheless GridSim seems to be the most widely downloaded computing grid simulator, with over 20.000 downloads since 2007.

OptorSim

OptorSim by Bell et al. [2003] is a rather short-lived simulator as it is no longer maintained. The architecture and design of OptorSim is built around the use-case of optimizing replica placement. Each computing element is represented by a single thread. The computing elements are controlled by a resource broker, run in another thread. Each thread reports the advancement of simulation time, which is modeled based on static configuration tables, to the broker. The broker assigns jobs to idling computing resources, thus, there is no real scheduling. For the use-case of replica placement the approach was deemed sufficient. Each storage resource is represented by a data structure noting the file availability in the storage system. If a computing job is accessing a local storage element, no storage access time is predicted, as the local availability of the replica is deemed sufficient. If a transfer from a remote storage is required, the transfer time is predicted based on a static bandwidth table and the job is given back to the broker until the time the transfer finishes. This means that the congestion of several network flows on the same network link is not taken into account. Despite the non-existence of a proper storage model and the simplifications of the network model OptorSim is still quite widely used and cited, mostly as the main objective of OptorSim was to simulate storage capacity and the frequency of replica accesses under different distribution patterns, which allowed for these simplifications.

SimGrid

SimGrid was presented by Casanova [2001] and is probably the most maintained grid simulators, with the newest extension published in 2015 [Lebre et al., 2015]. SimGrid performs event-driven simulation and its design offers three different APIs for user applications to interact with the simulation, as shown in Figure 3.2. The APIs enable the SIMIX part of the simulator, which is a kernel providing abstractions for concurrent process simulation. The actual core of the simulator is the SURF layer of the application, which enacts the simulation of execution of activities and resources. Each activity is defined as an amount of work by the upper layers, and when its remaining amount of work reaches zero the SURF layer signals the SIMIX layer of the success of an activity. SimGrid's most notable contribution is their network model

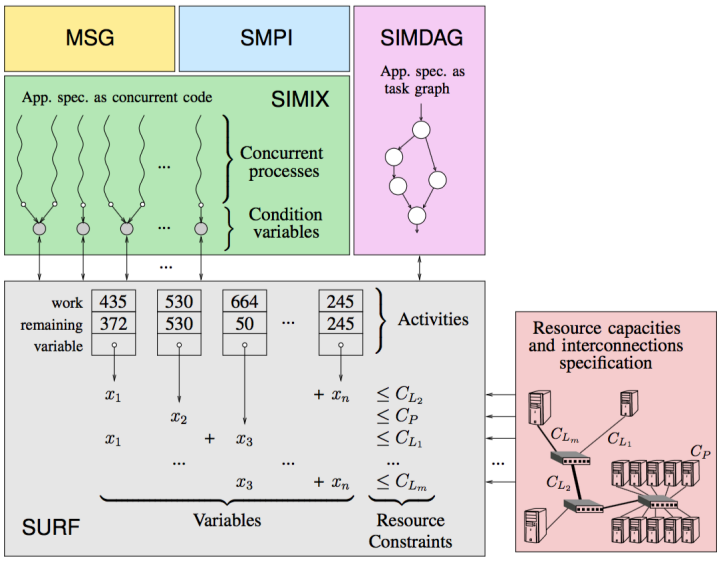

Figure 3.2: Overview of the SimGrid architecture [Lebre et al., 2015].

[Velho and Legrand, 2009], which is flow-based. This model offers, in contrast to many network simulators in literature which are based on packet-level simulation, high accuracy while it still scales to data-intensive systems. Also one of SimGrids big advantages is that its network model is strongly validated against real systems [Fujiwara and Casanova, 2007, Velho and Legrand, 2009]. Storage support for data grids was added very recently to SimGrid [Lebre et al., 2015] and is based on seek time and bandwidth configurations for each storage system, however with a much more advanced sharing model when compared to OptorSim.

GloBeM

Grid global behavior prediction [Montes et al., 2011] based on the GloBeM model [Montes et al., 2008, 2009] follows a different approach than the previously presented models. GloBeM abstracts the grid as one single entity and uses a machine learning approach, based on monitored traces, to model system behavior. An overview of the GloBeM workflow is shown in Figure 3.3. The advantage of this approach is that nearly no system knowledge is necessary to build accurate models. Thus model creation is quick and in general also quite accurate. However, the simulation accuracy is very much depending on the similarity of the simulated workload to

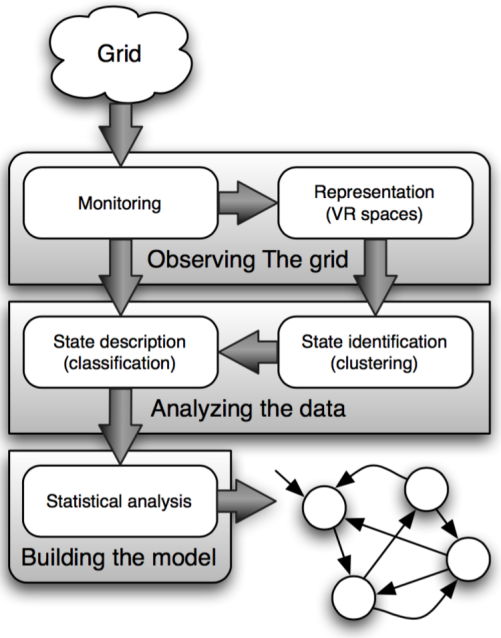

Figure 3.3: Overview of the GloBeM workflow [Montes et al., 2011].

the trained workload. In practice the simulation accuracy suffers when simulating non-similar workloads. The described use-cases of the model only include computing grids.

3.3 Cloud simulation frameworks

Cloud computing is a computing paradigm which gives the user convenient, on-demand access to large amounts of shared resources. The types of resources are manifold, but usually when referring to cloud computing, compute or storage resources are assumed. At the foundation of cloud computing is a large pool of resources, like computing nodes connected in a networked infrastructure. The cloud platform then assigns the resource, or part of the resource, to the requesting user.

Due to the similarity to grid models, cloud simulators mostly evolved from them. This section introduces the most commonly used cloud simulators.

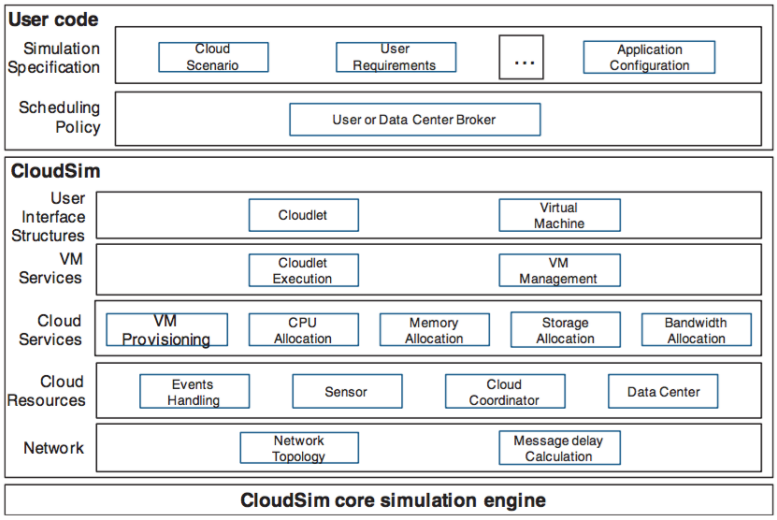

Figure 3.4: Overview of the CloudSim architecture [Calheiros et al., 2011].

CloudSim

CloudSim was introduced by Calheiros et al. [2011]. The simulator uses GridSim as its simulation core and builds the abstractions for the cloud activities and resources, such as virtual machines, around it. The architecture overview is shown in Figure 3.4. Each resource in the cloud is described by a set of characteristics like millions of instructions per second (MIPS), memory and storage and a provisioning policy for allocating processing cores to the resource. The actual virtual machine allocation model is based on a space-shared policy that estimates the finish time of a task managed by a virtual machine. For storage the model focuses on predicting capacity usage, as the response time for storage operations is, similar to GridSim, based on static transfer times. As already explained in the GridSim section, the model is not well validated against operational systems [Casanova et al., 2014].

GroudSim

GroudSim [Ostermann et al., 2010, 2011] is a simulation model specifically designed to simulate cloud platforms and applications. It also originates from an event-driven grid simulation model. The modeling approach focuses on job executions in clouds, and grids. The execution time estimation is based on a stochastic model for runtimes and different cost models for scheduling. The

model does not consider network congestion but bases its transfer estimations on a simple bandwidth sharing model, with different flows just getting an even amount of bandwidth. Storage access is not estimated and considered to be negligible.

3.4 Peer-to-peer simulation frameworks

The peer-to-peer computing paradigm splits workloads equally between privileged hosts. The peers of the network provide their resources, such as storage, bandwidth, or processing capacities to the network, without needing a central point of coordination. In this model peers act both as consumers and suppliers of resources, in contrast to the classical client-server paradigm.

Modeling peer-to-peer systems quickly gets into the dilemma of trading accuracy for scalability, as peer-to-peer systems often involve millions of peers. This section presents the most prominent peer-to-peer simulators.

OverSim

Baumgart et al. [2007] published OverSim which relies on the OMNeT++ [Varga, 2001] discrete event simulation core. The architecture overview of the system is shown in Figure 3.5. The architecture is split up in three different parts. The application layer offers basic interfaces to basic key-based routing implementations like Distributed Hash Tables [Stoica et al., 2001]. The Overlay layer then offers different structured peer-to-peer protocol implementations, which are directly used by the application layer. The actual simulation is then done in the Underlay layer, which is partly based on the OMNeT++ core as it offers many features for visualizing the simulation workflow. The Underlay layer also offers different network protocol models, with different accuracy and scalability characteristics. Their most-scalable simple protocol puts the peers into an Euclidean space. Each access path gets characterized with a bandwidth, access delay, and packet loss. The end-to-end packet delay of a packet between two nodes is approximated by a constant formula containing these attributes. Thus the underlying network can be simply simulated by one single event and is even able to model contention happening close to the peers. Despite this simple model the accuracy is quite high and due to the low complexity

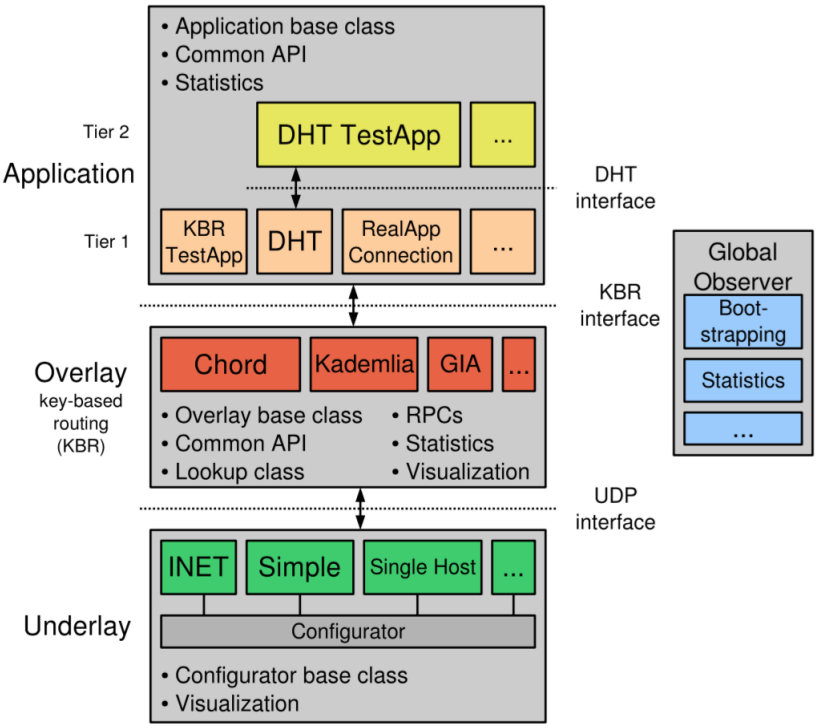

Figure 3.5: Overview of the OverSim architecture [Baumgart et al., 2007].

the model is very scalable. However, the model gets less accurate with large, wide-spanning networks. The model also doesn't predict any storage events as the authors focus on processing and bandwidth resources.

PeerSim

PeerSim [Montresor and Jelasity, 2009] is the most widely used simulator for peer-to-peer studies. The architecture of PeerSim is designed in a modular way for an efficient and simple configuration. To facilitate the simulation of large networks, the model makes certain simplifications. The main assumption is that mostly very small messages are exchanged between the peers, thus bandwidth effects are ignored, as the key characteristic is message count and not transfer duration estimates. To further improve the simulation time for big networks the message delay between two peers is assumed to be constant, thus the network model is initialized as a fixed-delay model. Due to the focus on small message passing, no storage model is included in the

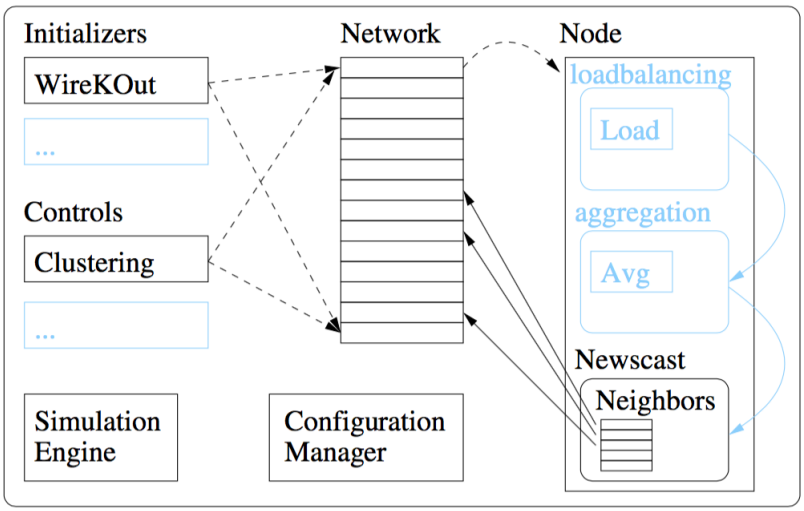

Figure 3.6: Overview of the PeerSim architecture [Montresor and Jelasity, 2009].

simulator as well. The architecture is shown in Figure 3.6 and is summarized as follows: The different initializers are executed before the simulation, based on the simulation configuration. Once initialized the control components manage the simulation by monitoring and manipulating the other components, such as network or nodes. The simulation engine works both in an event-based as well as cycle-based approach, where protocol actions are executed in a pre-defined order. The cycle-based approach makes further simplifying assumptions and scales better with large workloads, but is less realistic.

CHAPTER 4

Novel hybrid simulation model for data-intensive systems

4.1 Introduction

This section outlines the general idea behind system models and presents the different ways of achieving this in the context of data-intensive systems. Law and Kelton [1991] define a *system* as a collection of entities that act and interact together toward the accomplishment of some logical end. They define a *model* as a representation of the system to be studied as a surrogate for the actual system. In Figure 4.1 the general ways of studying a system, according to [Law and Kelton, 1991], are illustrated.

- *Experiment with the actual system.* Experiments conducted with the actual system are representative and relatively simple to conduct. However, with increasing complexity of a system, this method becomes infeasible. Even small experiments might be disruptive to the operation of the system and bigger experiments, such as testing partial system failures, would be simply catastrophical for most large scale systems. This is important for data-intensive systems as even small user inputs can create strong impacting effects on the system. For this reasons, data-intensive systems are usually studied by *experimenting*

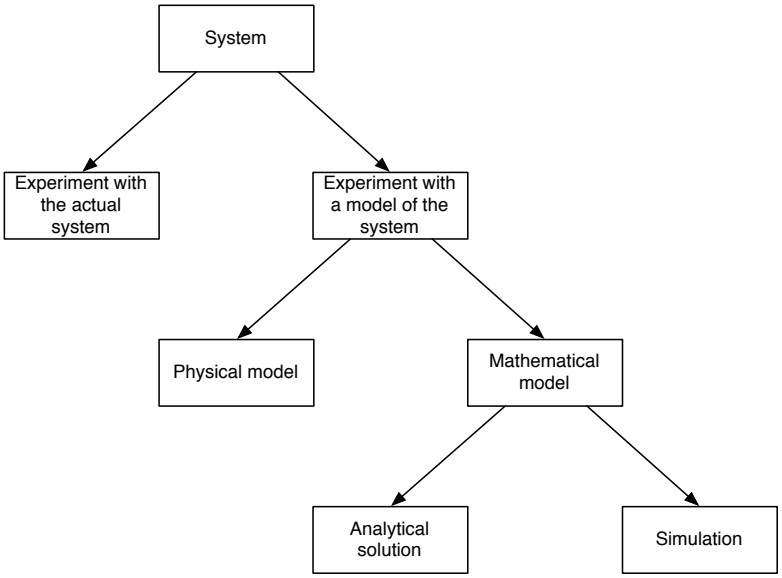

Figure 4.1: Strategies to study a system [Law and Kelton, 1991].

with a model of the system.

- *Physical models* are real-world representations of the system built at scale. In the context of software systems these are smaller-scale versions of the real system, such as experimental testbeds. However, for large scale distributed systems a representative testbed would require an infeasible amount of resources. Also, most often researchers want to investigate specific situations happening during operational workloads, however, it is non-trivial to create a representative operational workload at scale [Lassnig et al., 2011]. For these reasons physical models are most often ruled out for data-intensive systems.

- *Mathematical models* represent a system in terms of logical and quantitative relations which are manipulated in order to observe how the system reacts.

- *Analytical solutions.* Mathematical models can be distinguished into two different types. The first type being analytical solutions. With these models the researcher identifies a set of formulas that abstract the functionality of the system and thus always deliver an exact solution. With increasing system complexity such models are usually extremely convoluted and de-facto unobtainable.

- Banks et al. [2009] describe *Simulation* as the imitation of the operation of a real-world system over time. In contrast to analytical solutions simulation means to numerically approximate the model/system to see how different inputs affect the output measures. Due to the complexity of data-intensive systems, simulation is the only feasible method for modeling these systems.

Furthermore, mathematical models and specifically simulation models can be classified alongside three different dimensions [Law and Kelton, 1991]:

- *Static* and *Dynamic* simulation models. A static model is a representation of a system at a specific point in time, or a model of a system where time plays no role in the output of the system. A dynamic model represents a system which changes over time, which is the case for data-intensive systems.

- *Deterministic* and *Stochastic* simulation models. A deterministic simulation model does not contain any probabilistic functions. However, as a requirement for a model to be deterministic, all input quantities and functional relationships of the system must be known. This can be the case for certain processes in physics or chemistry, but in computing, it is infeasible to collect all input quantities of a system, thus a certain set of variables is always unknown. This leads to elements of the model always having some kind of random variate, making the model stochastic. For example, in a data-intensive system it is highly unlikely to have accurate and complete data about the traffic occurring on all network links, such as the internet.

- *Discrete* and *Continuous* simulation models. A discrete simulation model is a model, where the system values change at specific points in time. In a continuous model these values change continuously over time. A mixture between both types can be implemented.

Thus a model of a data-intensive system, such as the hybrid simulation model introduced later in this chapter, has to be discrete, dynamic and stochastic simulation model. The remainder of this chapter continues as follows:

In section 4.2 the requirements to an operational, data-intensive system model are discussed. What kind of estimators has the model to return? What kind of component knowledge is necessary in order to be able to build the model. What kind of data is necessary and can it be assumed that access to this data is, in general, available to researchers of data-intensive systems?

Section 4.3 introduces the idea and concept of the hybrid simulation model for data-intensive systems. The section covers the general structure and workflows of the system model as well as the introduction of system component models.

In Section 4.4 a quantitative system analysis is conducted on the data-intensive system Rucio. This analysis should shed light on the critical workflows of the system and which components have an impact on the performance of these workflows. The objective of this section is to identify a set of components which significantly influence the performance of these workflows, in respect to the thesis research question. These components then have to be modeled and integrated into the hybrid system model.

4.2 Requirements for an operational, data-intensive system model

The first requirement is already defined by the research question of this thesis:

Requirement 1 *The model predicts the response time of individual read and write operations.*

A model of a data-intensive system must process input operands, such as the current state of the system and a descriptor of a read/write operation, and return an estimator for the response time of the given operation. Existing grid models only estimate resource usage, such as bandwidth or storage utilization. Other models in literature give performance estimators, but at an abstract level such as classifications or quality of service distributions. These models have merit for certain studies, but for most researchers and operators of large scale data-intensive systems the performance of end-user operations is the focus. This enables researchers to investigate different data placement algorithms or even integrate the predictive model in the decision making workflow of the production system.

The second requirement for a data-intensive system model is defined as follows:

Requirement 2 *The data grid components are black boxes. Component models must be constructed based on global observations, rather than internal component data or knowledge.*

The reason for this requirement lies in the idiosyncrasies of large-scale distributed systems. These systems can be distributed across multiple continents and are managed by a large group of people. This leads to a highly heterogenous system, as the organizations [Foster et al., 2001] are free to choose different technologies, manufacturers, and configurations for their systems. Access to individual parts of the system is most often very limited, thus global observations, such as global traces, are the only type of available information.

Requirement 3 *Only passive observations must be used instead of observations from controlled system experiments.*

This requirement is specified due to the substantial differences between operational data-intensive systems and systems which only exist in test environments. A common approach in literature to construct a system model is to engage the system in controlled experiments, to capture more detailed performance characteristics. These experimental workloads are very often used in academic testbeds, such as PlanetLab by Chun et al. [2003]. This approach of *in-operando* built resource models most likely leads to more accurate results, however, in **operational** production systems it is simply not feasible to run large-scale controlled experiments, to observe the system under different load conditions. Such experiments are generally considered too risky as they could produce unforeseen side effects or even jeopardize operational data.

With the established requirements it becomes immediately apparent that the current state of research is in violation with most requirements. For example, the GridSim [Sulistio et al., 2008] and SimGrid [Lebre et al., 2015] models do estimate operation response times, thus they do comply with requirement 1, however both systems require extensive topology definitions for attached storage systems, such as the number of disks attached at an endpoint, as well as throughput, and seek rates for each disk. This breaks requirement 2. Not only is it mostly unknown how specific sub-systems are composed and specified, but even if that knowledge was available, these sub-systems are not necessarily compatible with the performance topology requested by these models. For example, tape robots for cold storage of data operate substantially different than

disk pool systems, thus it is not possible to define them with the constrains of these topologies, such as the number of disks.

The work of Montes et al. [2011] with their GlobeM model does not specifically consider response time estimators, as the authors focus on state transitions, but in principle, the model could also be used to predict response times, however for now there is no published work where the authors have demonstrated that. The GlobeM model complies to all requirements as it abstracts the data-intensive system as one single model and uses machine learning methods to train the model based on historic workloads. However, due to the abstraction of a very complex system into one single entity a lot of information gets lost leading to a potential loss in accuracy.

In summary, approaches in literature require either very detailed knowledge or data of the system, thus the level of abstraction is very small, or they over-abstract the system and therefore are subject to a potential loss in accuracy. The objective of this thesis is to create a model somewhere in between those extremes, which offers accurate predictions without requiring very detailed knowledge of the internals of the system.

4.3 Hybrid simulation model for data-intensive systems

The hybrid simulation model decomposes a data-intensive system into several components, builds component models for each, and unifies the models into a single system model. Specific general system knowledge is preserved, as it is not lost in a single giant black box model, while still complying to requirement 2. The approach of building the individual component models is driven by global observational data, instead of data or knowledge of internals of the component. The challenge of creating the hybrid simulation model lies in two steps: First, the selection of the right components to be represented by component models. This challenge is solved by analysis of an existing data-intensive system. A quantitative analysis investigates workflows and extracts the components which have a critical influence on the performance of the workflow. This analysis is shown in Section 4.4. The second step is modeling these components and achieving accurate estimators. These steps are shown in Chapters 5 to 8.

The overview of the hybrid simulation model is shown in Figure 4.2. The simulation ap-

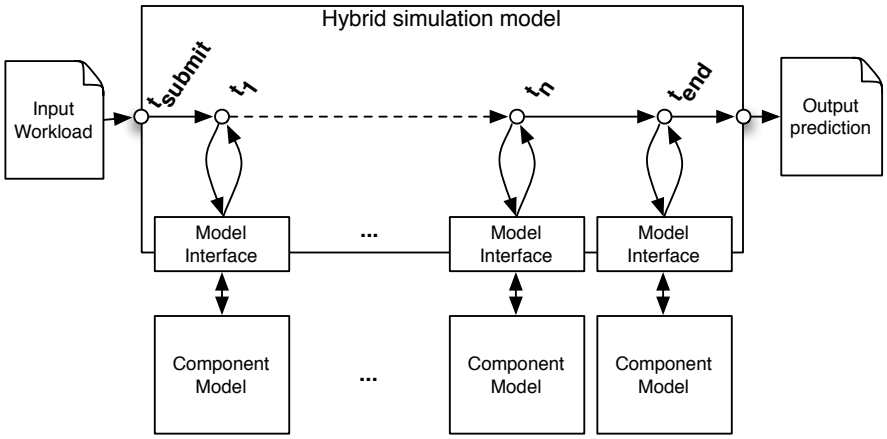

Figure 4.2: Overview of the hybrid simulation model concept.

proach is to execute the hybrid model in an event simulator, which accepts an input workload and estimates the individual operation response times. The hybrid model is designed in a modular approach. It consists of an interface for each component model. The component models themselves are connected to an interface which allows the attachment of different models while executing the same input workload. Each interface has access to the current system state of the data-intensive system. The component models return their operation estimator as well as their view of the current system state back to the interface, in order to be able to track the system state.

When the model gets integrated in an event simulator, the workflow is as follows: The input processor iterates all operations sequentially and queries the hybrid model for an estimator. The hybrid model then queries the component models for an estimator of the currently processed operation. The collected estimators for each operation are then aggregated by the hybrid model to generate the response time estimator t_{end}, which is reported back to the simulation processor.

4.4 Component model selection based on quantitative system analysis

Intuitively not all components will have the same impact on the performance of user operations and thus do not carry the same importance in respect to a system model. The objective of this section is to conduct a quantitative performance analysis of the workflows involved when writing and reading files in a data-intensive system. To that extent the analysis should identify the components used in this workflow and investigate how much time is spent in different parts of the workflow. Ultimately this should lead to the identification of the relevant and irrelevant components, in respect to the research question. This analysis is based on Rucio which collects detailed system traces since the beginning of 2009 [Lassnig and Fahringer, 2009]. These traces can be categorized into two groups: Internal traces, which are internal transfer requests coming from the Rucio daemons itself. This mostly includes transfer requests being made by the system for fully automatic replication of data. The second category of traces are external traces, which are requests being made by users or computing nodes to directly read or write data from or to storage systems.

The archive of internal traces in Rucio is collected since the launch of the system in late 2014. Each internal trace is a single file transfer event issued by one of the Rucio daemons. The dataset includes 670 million events. Figure 4.3 shows the amount of collected internal events since October 2014.

The archive of external traces includes more than 8 billion file operation level traces. Figure 4.4 shows the evolution of the captured traces starting from 2008. Thus the archive includes both traces from the DQ2 era as well as Rucio. The data shows a linear increase of traces since data taking with a drop in the year 2014, due to the upgrade phase of the LHC, during which no data was taken from the detector.

The conclusive outcome of the analysis is the identification of the performance critical components involved in user workflows of a data-intensive system. These identified components can then be modeled and used to predict the performance of user operations of data-intensive system.

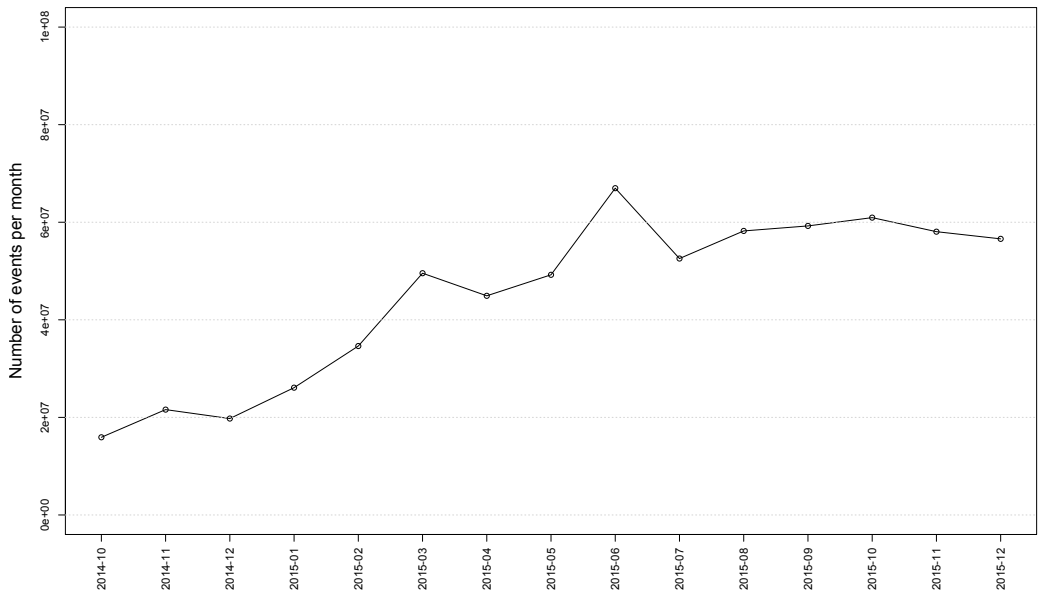

Figure 4.3: Amount of Rucio internal file transfer requests per month.

Instrumentation framework in concealed environments

The instrumentation framework used by Rucio was initially developed for its predecessor DQ2. It is described in detail by Lassnig and Fahringer [2009]. The initial objective was that no existing instrumentation framework had the capabilities to capture external operational traces at the required level of detail, while working under the operational and administrative requirements of the ATLAS data grid.

In contrast to other data-intensive systems, data grids have the unique characteristic that parts of the distributed system, such as storage and networking systems, are just utilized by the data-intensive system, and thus under very limited control of the distributed system. This leads to the idiosyncrasy that no instrumentation data is available about certain parts of the system. This data is contained within a *concealed environment*. This is partly a technological as well as an administrative challenge, as in data grids there is usually no administrative entity which is able to force specific monitoring software stacks to be deployed throughout the system. At the same time, however, operators and developers of the data grid require correct and timely runtime

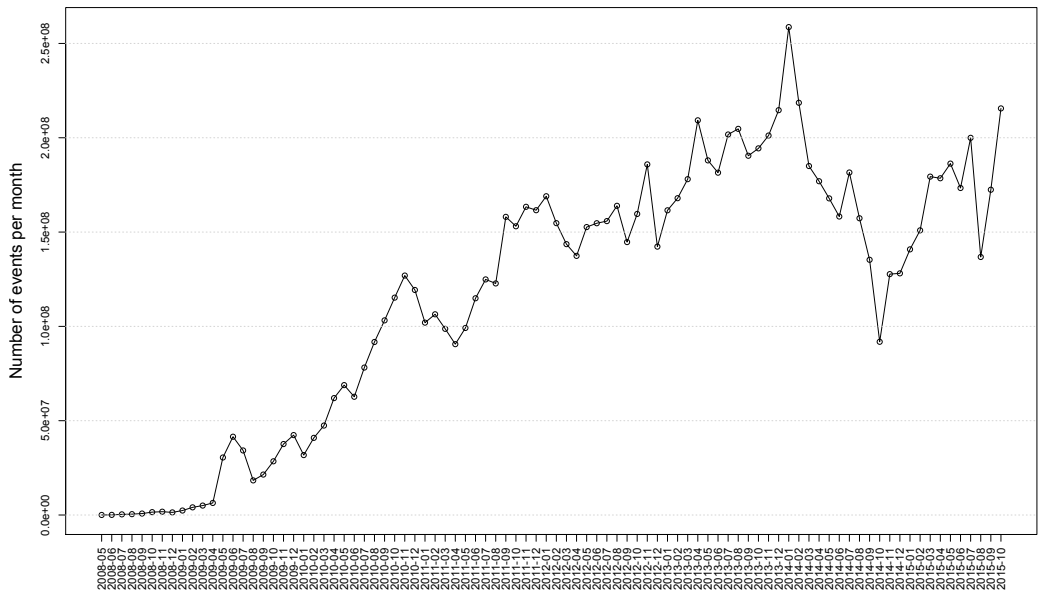

Figure 4.4: Captured amount of external trace events per month.

information at a specific level of detail. This challenge lead to the monitoring system of Rucio.

The characteristics of the instrumentation system are discussed in length in [Lassnig, 2014]. The key characteristics are *correctness*: the behavior seen by the participating clients must be correctly reported to the system. For example, if multiple client operations belong to the same transactional data grid operation, this transactional behavior must be part of the captured observation as well. *Non-invasiveness*: The observations must be captured without impacting the operational systems performance as well as reported within reasonable time. *Synchronization*: Events observed in multiple concealed environments must be reported in a synchronized way, without affecting the total order of events.

The implemented framework is partly based on the data stream reference architecture from Golab and Özsu [2003]. The architecture of the framework is based on data streams the distributed system has no control over. Thus the order in which data elements arrive as well as their volume is explicitly not controlled by the distributed system. In Rucio this is reflected by a client requesting data originating from a collective data identifier, such as a container or dataset. Multiple data streams are started by the client directly from the storage endpoints. As

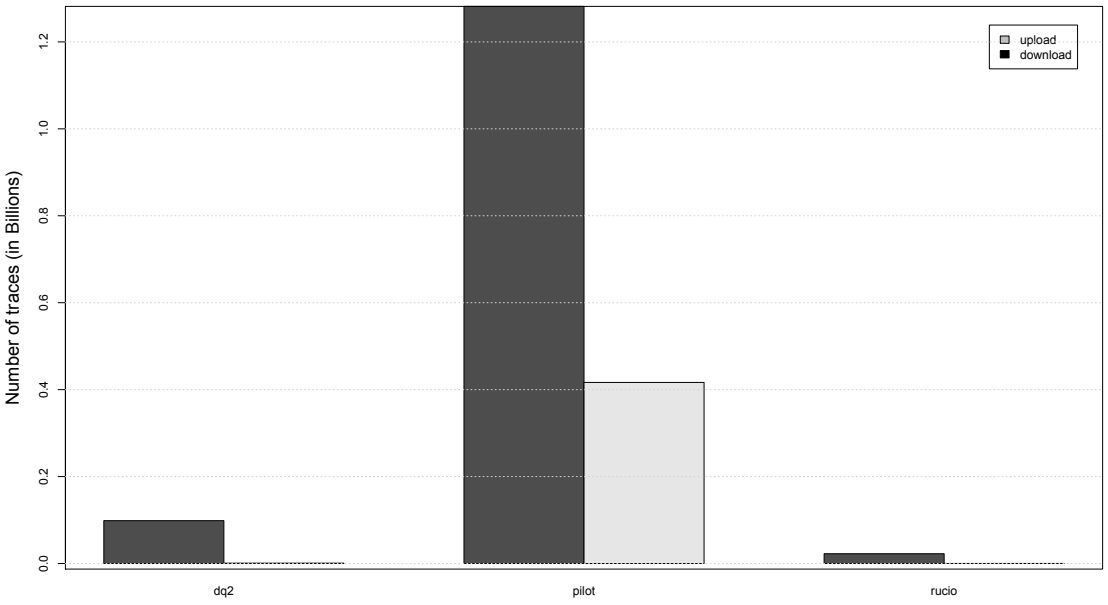

Figure 4.5: Number of traces per different client instrumenting Rucio in the year 2015.

no traces are directly captured, or reported, at the storage system or networking system, the only place to capture these performance characteristics is on the client which subsequently reports the observations back to the data grid. To achieve a full system view with this approach the majority of access-workflows used in the ATLAS collaboration had to be utilized to capture and report these traces. Specifically the Rucio clients as well as the workload management system PanDA [Maeno, 2008] was instrumented in that way. Figure 4.5 shows the distribution of different clients accessing files in Rucio in the year 2015. The majority of traces are generated by the workload management system PanDA through their pilot. The clients are usually only used by the user when finally reading the data, but all the intermediate analysis steps and recurrent processing runs are done directly by the workload management system, which pulls the data from storage directly to the computing element. Thus a much higher access rate of the data, originating from the workload management system, is expected. Only a very small amount of upload requests is done by the clients as they are mostly used by the users who are usually not creating data. The plot also shows that about 100 million traces come from the legacy DQ2

clients. These are the clients of the predecessor data management system (DQ2) which are still compatible with Rucio. Especially in the beginning of 2015, shortly after the decommissioning of DQ2, many users were still using the DQ2 clients and only switching to the Rucio clients later on.

A typical instrumentation workflow is described based on the `rucio download` behavior, when downloading a dataset. The workflow is depicted in Figure 4.7, at the end of the section. When executing the command the client starts to capture performance traces for different parts of the workflow. Between the timestamps t_{submit} and t_{start} the Rucio servers are queried for the contents of the dataset as well as the replica locations, including a preliminary replica selection, of all containing files. In the next step the client spawns a thread per file so that downloads are executed concurrently. For each file the client decides either to use the pre-selected replica or make a decision on its own. This means that the client is potentially using different storage systems to request the replicas. Once the replica is selected the correct storage protocol is chosen and the transfer call to the storage system is constructed. Before starting the transfer the $t_{transfer}$ timestamp is recorded. After the transfer finishes the data integrity validation phase of the workflow starts with recording the $t_{validation}$ timestamp. A checksum is calculated for each transferred file and compared with the checksum initially provided by the server. This concludes the validation phase and the t_{end} timestamp is recorded. The client then submits the complete trace event to the server, thus reporting the events which happened in the concealed environment.

While the t_{submit} and t_{start} timestamp can be collected directly at the server, the other timestamps are completely independent from the central infrastructure and thus only execute in the concealed environment. As the client spawns a thread for each file transfer one trace event is submitted for each transfer. However, as the first part of the workflow, until the t_{start} timestamp, is only executed once, these two timestamps will be similar in all traces. A detailed timeline of trace timestamps is shown in Figure 4.6. The trace events are sent to the server at the end of the workflow irrespective of the success of the transfer. Even if a transfer fails or the client is stopped gracefully, an informative trace event will be sent to the server. This is important as potentially incomplete trace events have to be accommodated for subsequent analysis. The trace events are sent to the REST interface of the loadbalancer in the JavaScript Object Notation

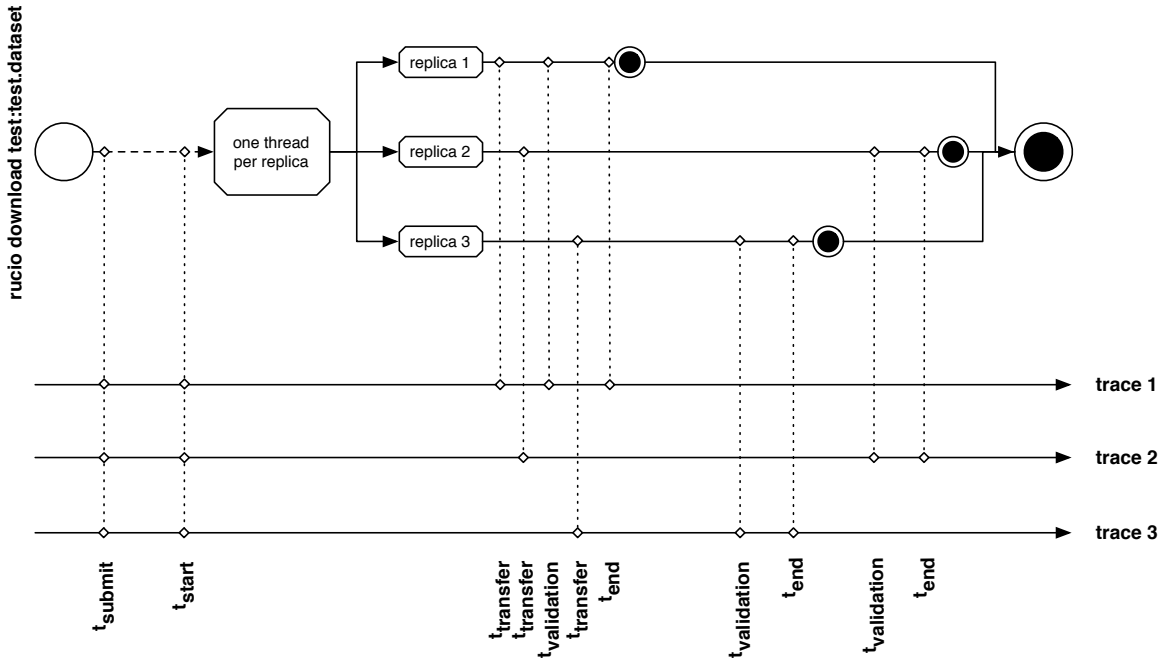

Figure 4.6: Rucio trace instrumentation timeline visualisation.

(JSON) format. The loadbalancer then relays the trace to a specific set of Rucio Apache servers, which are exclusively responsible for receiving and storing the traces in a timely fashion.

The `rucio upload` instrumentation behavior is quite similar to the download workflow and shown in Figure 4.8. At first the Rucio server is instructed to create the dataset and the files as well as replicas. All replicas are created in a `COPYING` state. Once the server call returns the t_{start} timestamp is recorded and a thread is created for each file transfer. The upload call is constructed based on the available protocols, and the transfer operation is executed between the $t_{transfer}$ and $t_{validation}$ timestamps. Once finished the server is notified about the success of the transfer operation and the replica state is changed to `AVAILABLE`. Similar to the download workflow, the t_{submit} and t_{start} timestamps will be equal in all file traces. The rest of the workflow is done concurrently as well, thus the other timestamps diverge.

A similar instrumentation was put on top of the pilot unit of the workload management system. Instances of the pilot run on all processing nodes of the computing grid, which interact directly with Rucio APIs for replica lookups but do construct and execute its own protocol

calls, without using the Rucio clients. The instrumentation of the PanDA pilot workflow when downloading data is shown in Figure 4.9. In this workflow the timestamps get recorded in two separated parts of the workflow. First, when the job gets dispatched from the workload management system in the sites computing queue the Rucio servers are contacted to resolve the data identifier into file and replica information. This is also the time when the t_{submit} and t_{start} timestamps are recorded. At a later point in time, when the pilot gets instantiated at the computing element, it already has all the replica information as part of the payload. The pilot then directly constructs the storage calls and transfers the data from the storage element to the computing element, without using the Rucio clients. After the transfer is done the checksums are also verified. The trace event is then sent to the Rucio server.

The pilot upload workload instrumentation, shown in Figure 4.8, is also similar. When the job gets created at the PanDA servers the respective output datasets are instantly created in Rucio. Once the pilot is instantiated and the job finished creating all the output data the t_{submit} timestamp is recorded. The pilot then uploads the data to the storage element. It does this without using the Rucio clients. However, the pilot uses the same deterministic functions to create the storage paths thus the files end up in the same location where the data management system would put them. After all transfers are done the t_{end} timestamp is recorded as well and the trace events are then sent to Rucio. Once the pilot sent a positive acknowledgment to the server of the workload management system. Rucio is contacted again to logically create the replicas in the catalogs, which are already physically existing in the storage system. The files are also added to the output datasets.

Next to the tracing timestamps, which are important for the performance analysis, each trace also records a set of meta attributes describing the operation. The attributes are shown in Table 4.1. These attributes not only give the timestamps for performance analysis, but also hold information about the protocols used and errors in case of transfers not being successful. This information is essential as there, most often, is no other information about why a certain storage end point is failing.

Attribute	Description
account	The account name of the issuer of the operation.
appid	Application id field which is set by the application recording the trace.
catStart	The start of the catalog lookups. In this document it is referred to as t_{submit}.
clientState	The final state of the client when finishing the transfer. This can be DONE, FAIL or partial failures.
dataset	The dataset name the transferred file belongs to.
datasetScope	The dataset scope the transferred file belongs to.
duid	The dataset universal id. This field was used in DQ2 and is meanwhile deprecated.
eventType	The event type of the operation. This can either be a get/put from a client, from a pilot or from a pilot for analysis data.
filename	The file name of the transferred file.
filesize	The file size, in bytes, of the transferred file.
guid	The global unique identifier id of the transferred file.
hostname	The hostname of the machine issuing the request.
ip	The ip address of the machine issuing the request.
localSite	In case of a download, the site identifier of the site the data is sent to. In case of an upload, the site identifier of the site the data is originating from.
protocol	The protocol used for the transfer.
relativeStart	The starting time of each transfer thread (t_{start}).
remoteSite	In case of a download, the site identifier of the site the data is originating from. In case of an upload, the site identifier of the site the data is sent to.
scope	The file scope of the transferred file.
stateReason	The reason for a failed transfer.
suspicious	The suspicious flag indicates that, for a (partially) failed transfer, data is potentially left at the storage system and has to be checked or removed.
timeEnd	The end time of the operation (t_{end}).
timeStart	The start time of the client (Not depicted in the other figures).
transferEnd	End time of each single transfer.
transferStart	Start time of each single transfer ($t_{transfer}$).
url	The fully qualified url of the source replica file.
usr	The hashed user name in case of the trace being given out externally to guarantee anonymity.
usrdn	The distinguished name of the certificate the user is using to authenticate himself.
uuid	The universal unique identifier of this operation. In case of multiple transfers being executed in one client operation, all the traces will have a similar uuid.
validateStart	Start time of the data integrity validation process ($t_{validation}$).
version	The version of the client being used.

Table 4.1: The recorded attributes for each trace in Rucio.

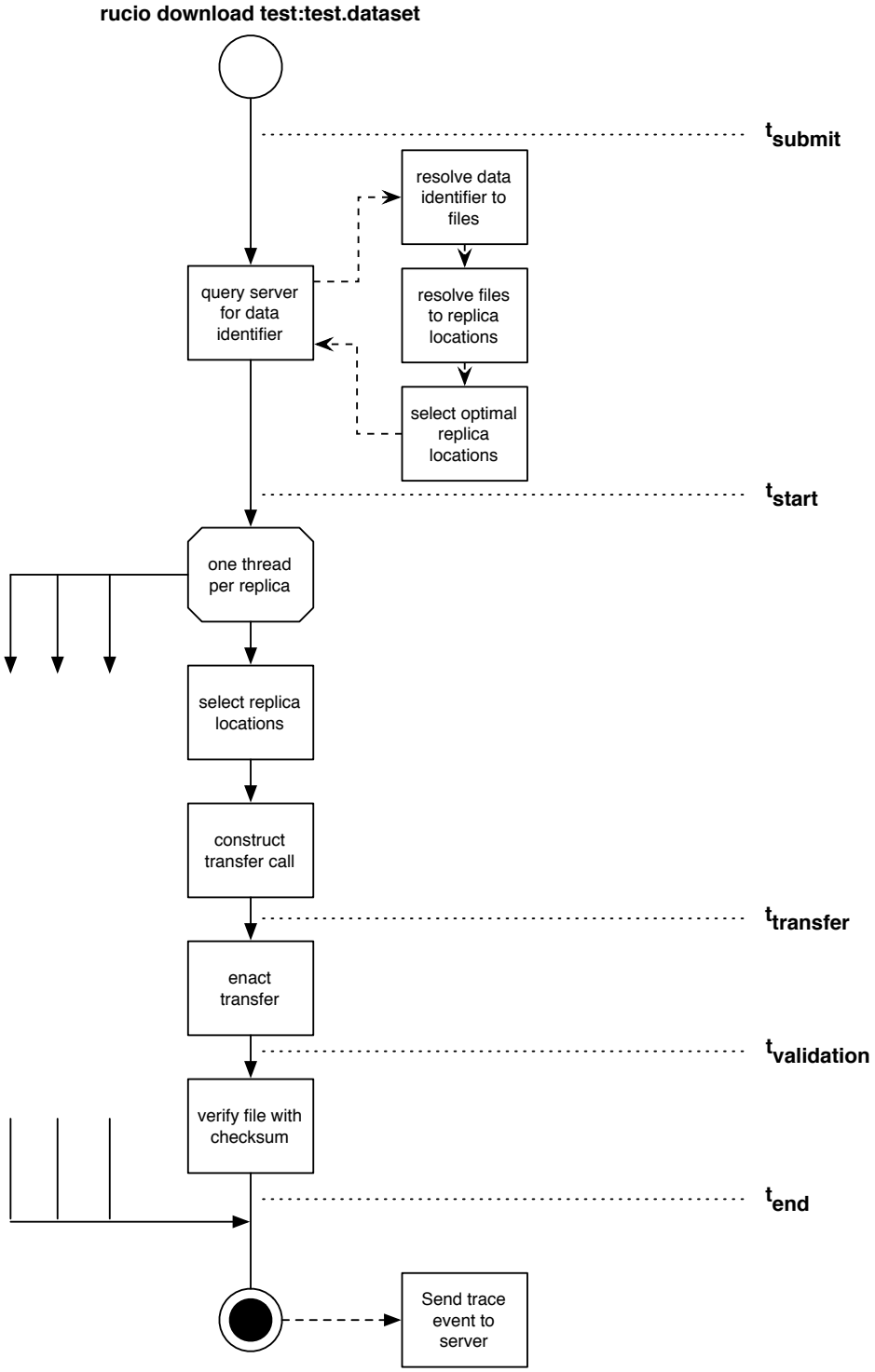

Figure 4.7: `rucio download` workflow and trace instrumentation.

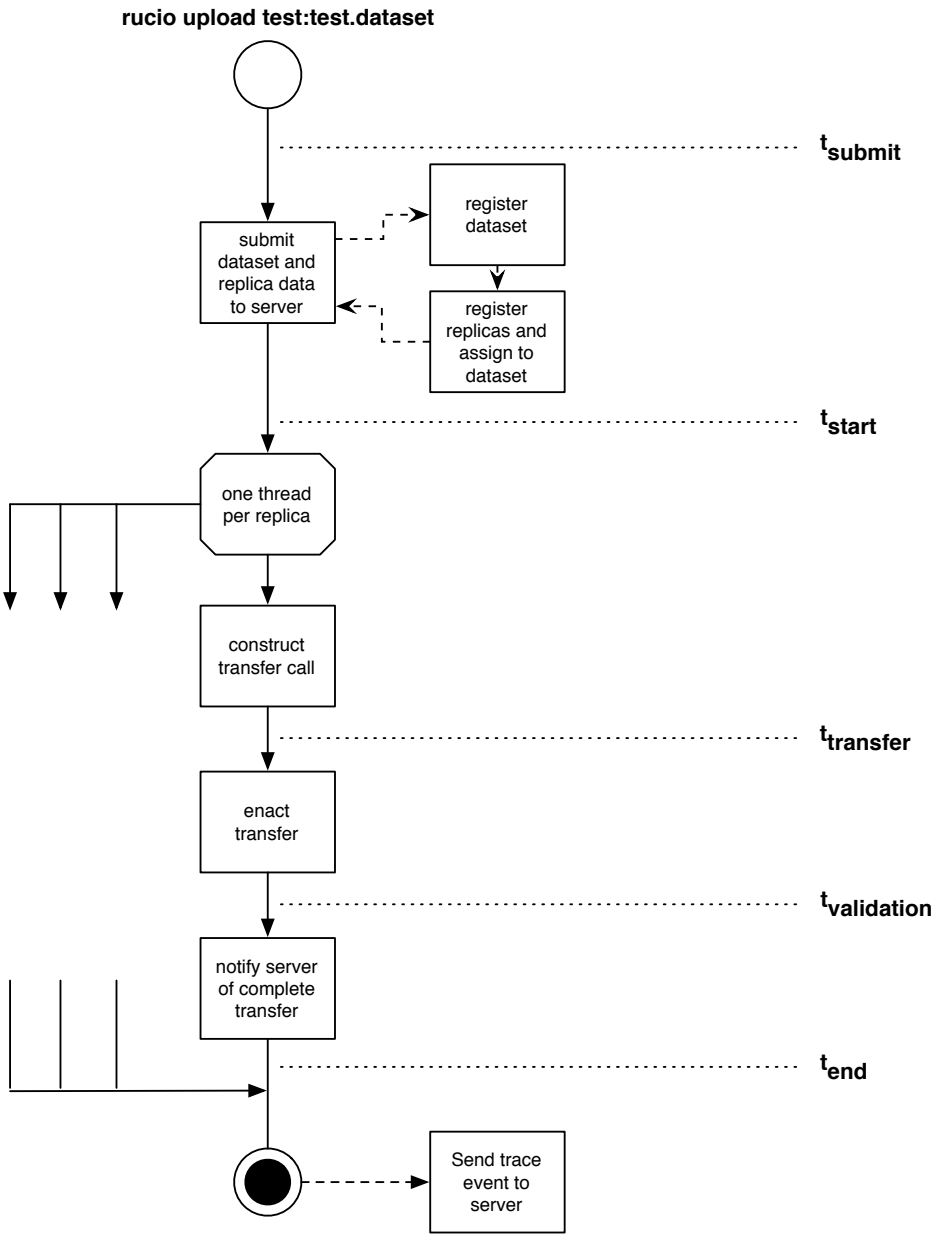

Figure 4.8: `rucio upload` workflow and trace instrumentation.

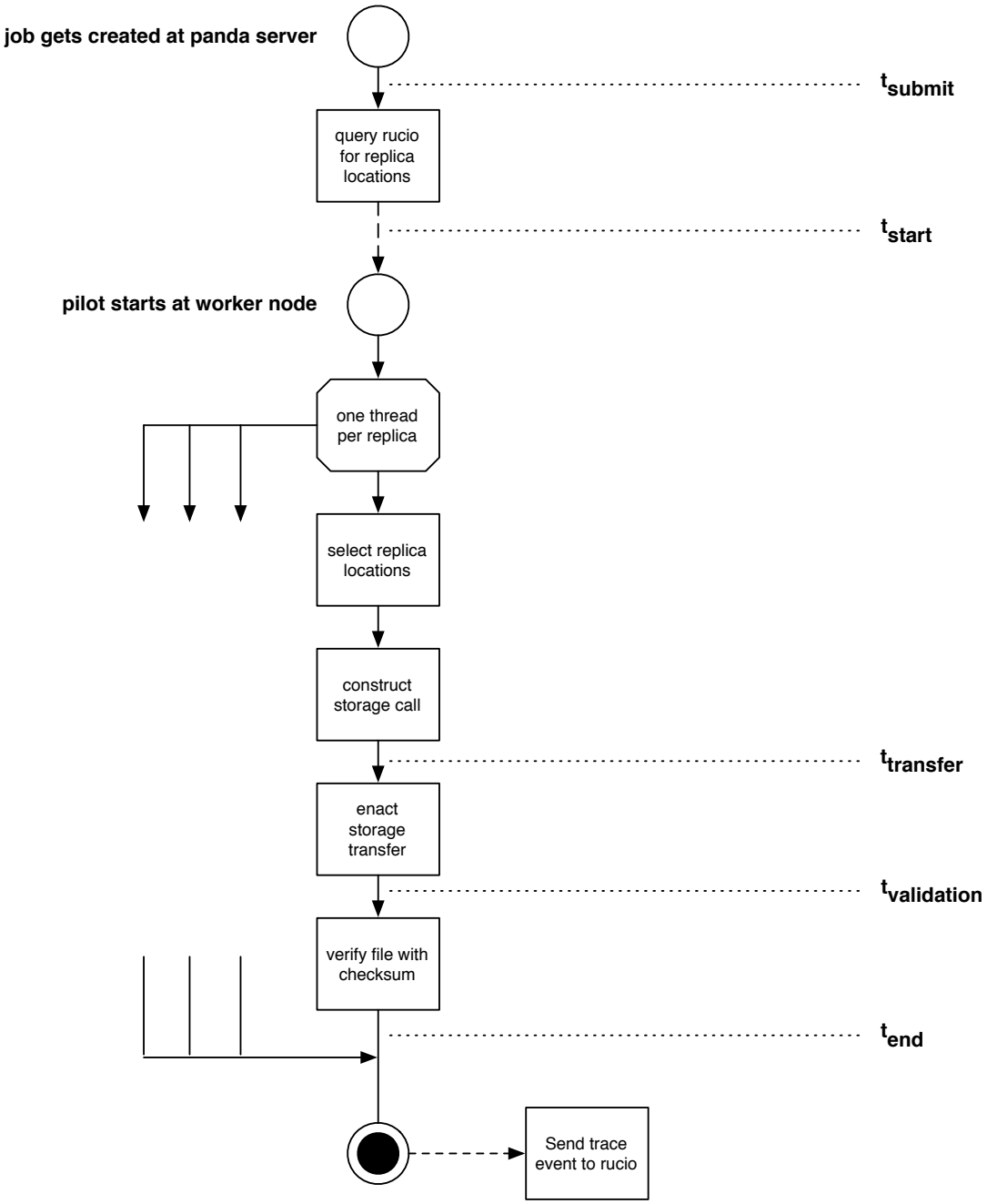

Figure 4.9: PanDA pilot get (download) workflow and trace instrumentation.

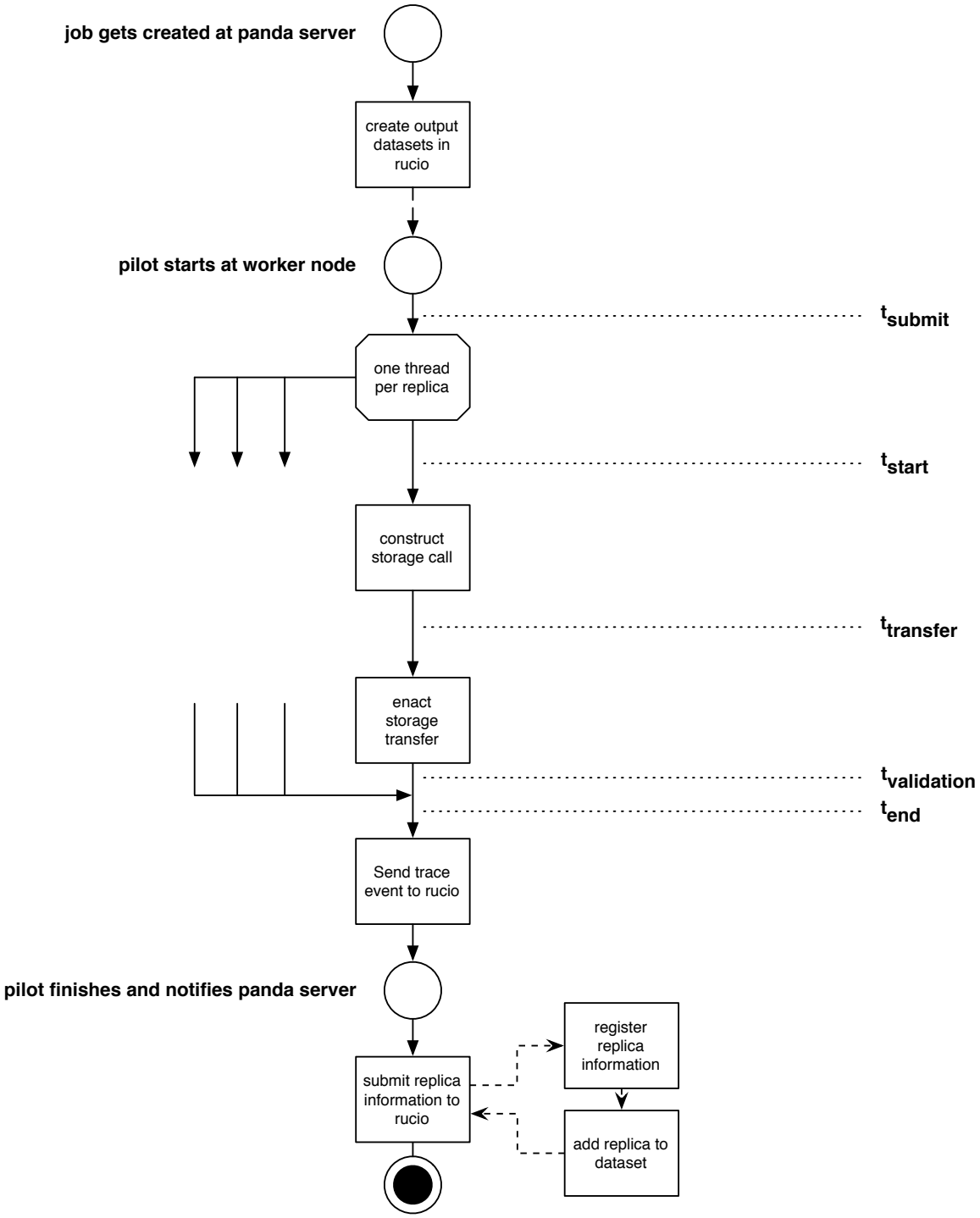

Figure 4.10: PanDA pilot put (upload) workflow and trace instrumentation.

Event traces of internal transfer requests

In contrast to the external operations, the internal operations are scheduled and executed by the data management system itself. The external data grid operations are entirely scheduled by the user or external application with little to no control of the system in scheduling them. All internal traces come from these transfer requests executed by the system. A single transfer event describes a storage to storage file transfer request using the networking system of the data grid. In Rucio, such transfer requests are issued by replication rules, either requested by a user or created by the distributed system itself, to optimize data distribution. For more information on replication rules, see Section 2.2.

Once a replication rule has been submitted to the system, the *Judge* daemon is interpreting the rule and creating the necessary transfer requests in the database. The *Conveyor* daemon asynchronously processes these transfer requests and schedules them for transfer, based on the system load, the priority of the transfer, the load on the network and the storage system, and other factors. Once the transfer request is scheduled for transfer it is submitted to the File Transfer Service [Kiryanov et al., 2015] at CERN which takes care of the storage to storage transfer calls. Each request is described by multiple attributes shown in Table 4.2.

The full workflow of a transfer request is shown in Figure 4.11. The replication rule submissions can be done by either using the python clients or directly the REST interface of the Rucio server. Usually replication rules are created for datasets or containers, rarely for single files. Once a rule is received the Judge resolves the data identifier to the relevant files and replicas, selects the target storage element and creates the necessary transfer requests in the database. For each required file transfer one transfer request is created. At this time also the $t_{created\_at}$ timestamp is recorded. As the transfer requests belong to the same rule and are created at the same time, they all have the same $t_{created\_at}$ timestamp. The Conveyor daemon continuously scans the database for newly created transfer requests. Once it finds newly arrived requests and the conditions of the system, network and storages allow for the transfers to be submitted, it submits the transfer jobs to the File Transfer Service (FTS). At this time the $t_{submitted\_at}$ timestamp is recorded. The File Transfer Service also has a scheduling layer in front of the actual transfer execution. This is to ensure that different priorities are taken into account as well as not to flood

Attribute	Description
created_at	Timestamp recorded when the transfer request is created.
updated_at	Timestamp recorded each time the transfer request is updated.
state	State of the transfer request.
request_type	Request type, which can be either a transfer or a stage in request from tape.
external_id	The external id of the file transfer service job.
scope	Scope of the file requested for transfer.
name	Name of the file requested for transfer.
dest_rse_id	RSE id of the destination storage element.
id	Unique request id.
previous_attempt_id	The previous request id in case the request is a retrial.
retry_count	The retry count.
attributes	Additional attributes stored as a serialized dictionary.
err_msg	The error message if the request failed.
rule_id	The rule id of the rule requesting the transfer.
bytes	The size of the file to be transferred.
md5	The md5 checksum of the file.
adler32	The adler32 checksum of the file.
dest_url	The generated url of the destination file.
external_host	The file transfer service host used to request the submission job.
activity	The Rucio activity being used for different prioritization of transfers.
submitted_at	Timestamp recorded when the transfer is submitted to the file transfer service.
transferred_at	Timestamp recorded when the transfer has finished.
submitter_id	The id of the actual submission daemon in the file transfer service.
started_at	Timestamp recoded when the transfer is started.
source_rse_id	RSE id of the source storage element.

Table 4.2: The attributes describing each transfer request in Rucio.

certain network links. Once scheduled, a transfer job is dispatched to one of the job processors in the pool which then executes the transfer job based on the protocols both the destination and source storage endpoint support. The job processor also records the $t_{started\_at}$ timestamp once started. When successful, the $t_{transferred\_at}$ timestamp is recorded. After the transfer finished successfully the job processor initiates a checksum validation. The job processor then terminates the job and the FTS central server notifies the Rucio conveyor about the transfer, including all the recorded meta data.

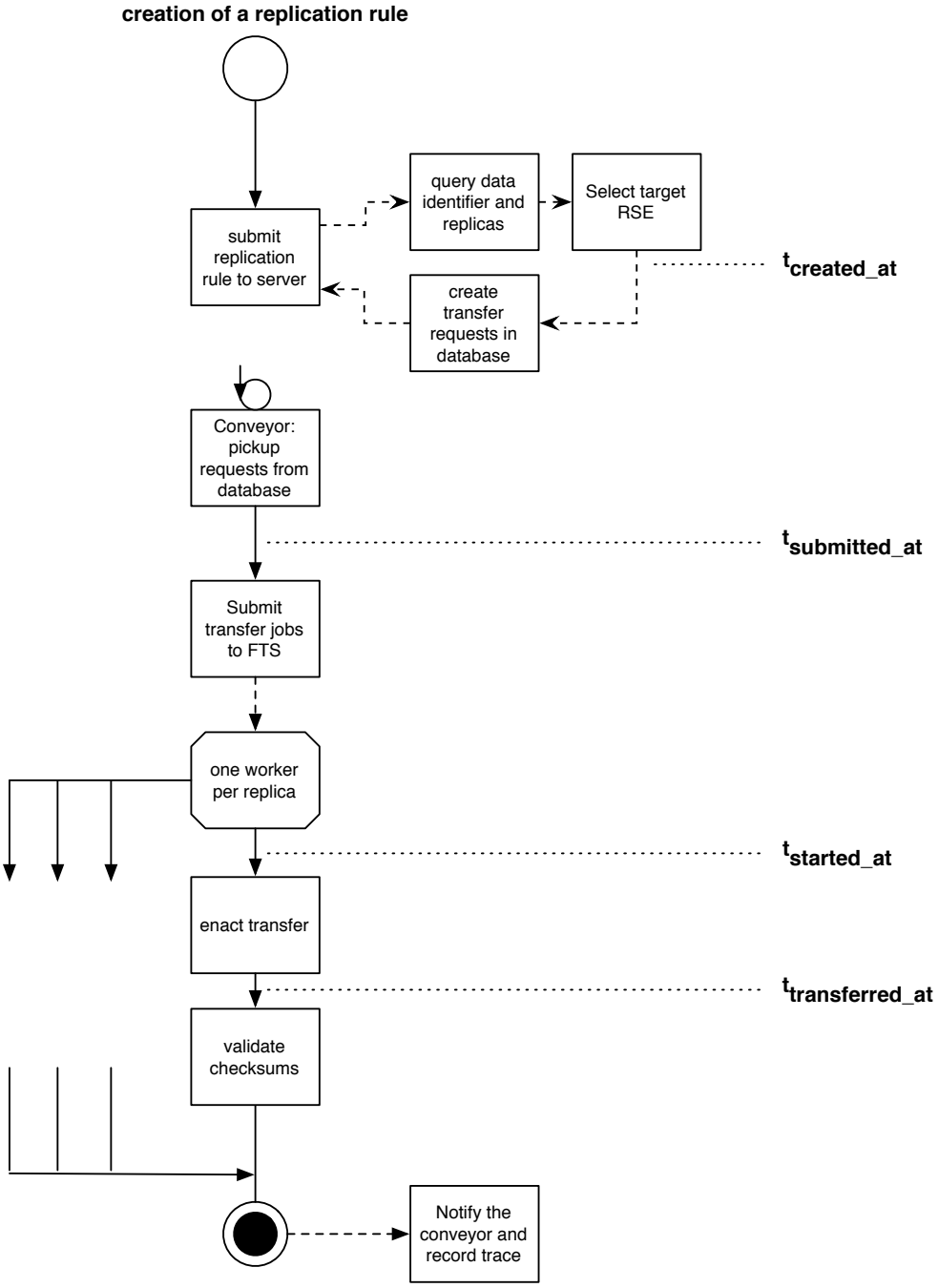

Figure 4.11: Rucio internal transfer workflow.

System analysis

The objective of this section is to take the trace archive of Rucio and conduct a quantitative analysis on it. The outcome of this analysis is a detailed understanding of how much time is spent in the different workflows, but especially, how certain components influence these workflows. These components can later on be modeled and included in a predictive model of the data-intensive system. This analysis is conducted in a generic way and the findings are presented, wherever possible, in the terminology established in the data grid reference architecture[Foster et al., 2001]. Thus the analysis is, for the most part, applicable to other data-intensive systems, however, there might be some peculiarities which are specific to the Rucio system. The analysis is based on the trace archive of Rucio of the year 2015. The dataset for 2015 includes about 2 billion external trace events and 617 million internal trace events.

As specified by the research objective the focus of this thesis is on modeling user operations, or so called data grid operations. These read (download) or write (upload) operations are executed on a collective data identifier such as a dataset or container. The user is usually reading or writing multiple files in one operation. Figure 4.12 shows an overview of data grid operations for download, upload and transfer, partitioned by the aggregated data grid operation size. Transfer operations are a special subset of write operations, when a user requests data being moved from one storage system to another. In this case the user does not execute the transfer himself, but requests a transfer being done on his behalf by the distributed data management system itself. There are two orders of magnitude more upload operations than download operations, especially in the smallest partition of data grid operations smaller than one Gigabyte. The reason for this is the PanDA pilot upload workflow shown in Figure 4.10. Many computation jobs are started on different computation elements to process the input data of a job. Each computation element writes the output of the job to the same output dataset, but as the computing jobs do not have any knowledge of each other, each upload is done with a different trace uuid, thus, considered as a different data grid operation. For the bigger partitions there are multiple orders of magnitude more data grid operations for downloads than for uploads. This is also for the reason described above. For the year 2015 there were slightly over 20,000 data grid upload operations larger than 100 Gigabytes, while there were over 340,000 data grid download operations for the same parti-

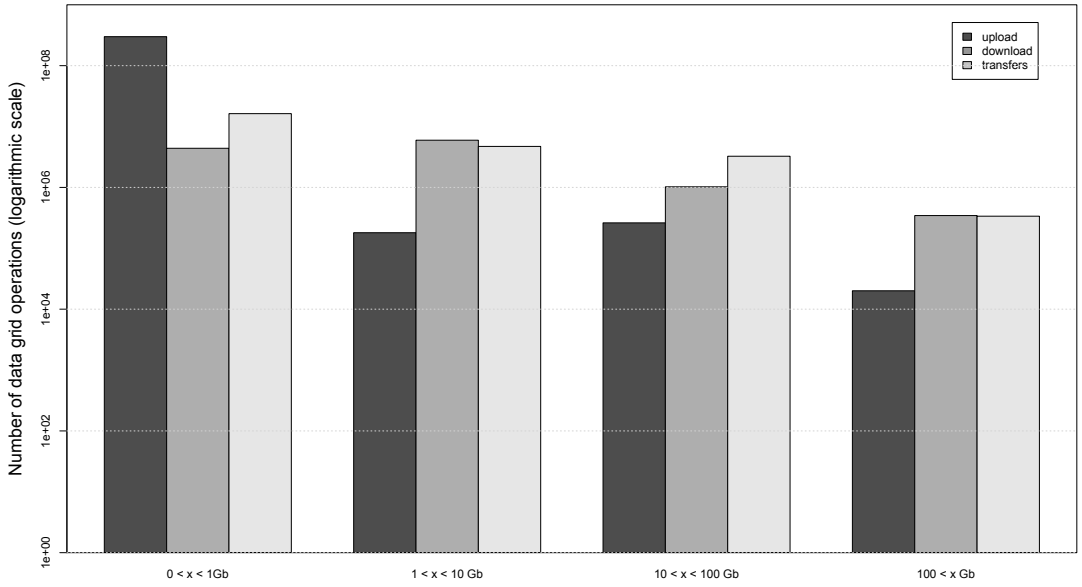

Figure 4.12: Number of data grid operations partitioned by data grid operation size for the year 2015 (Logarithmic scale).

tion. The amount of transfer operations is usually similar to the amount of download operations. In the 10 to 100 Gigabytes partition there are interestingly more transfer operations than download operations, which would suggest that some data is never read, which, to some extent, is true as the data is optimistically replicated to increase availability. But a second explanation for this phenomenon is that data is transferred as part of big datasets and later split up into smaller chunked datasets. Then only these smaller datasets are read from the storage.

The effect described above is also shown in Figure 4.13. This barplot shows the average size of data grid operations also partitioned by the operation size. While in the larger partitions the average size of uploads, downloads and transfers is essentially similar, the small partition shows an order of magnitude smaller average transfer size for uploads. For uploads in the smallest partition, the average transfer size is only slightly over 2 Megabytes, while for downloads the average transfer size is over 130 Megabytes (120 Megabytes for transfers). This is because output datasets are populated by many different data grid operations during upload, but downloaded in one data grid operation which yields a higher aggregated size for the download operations.

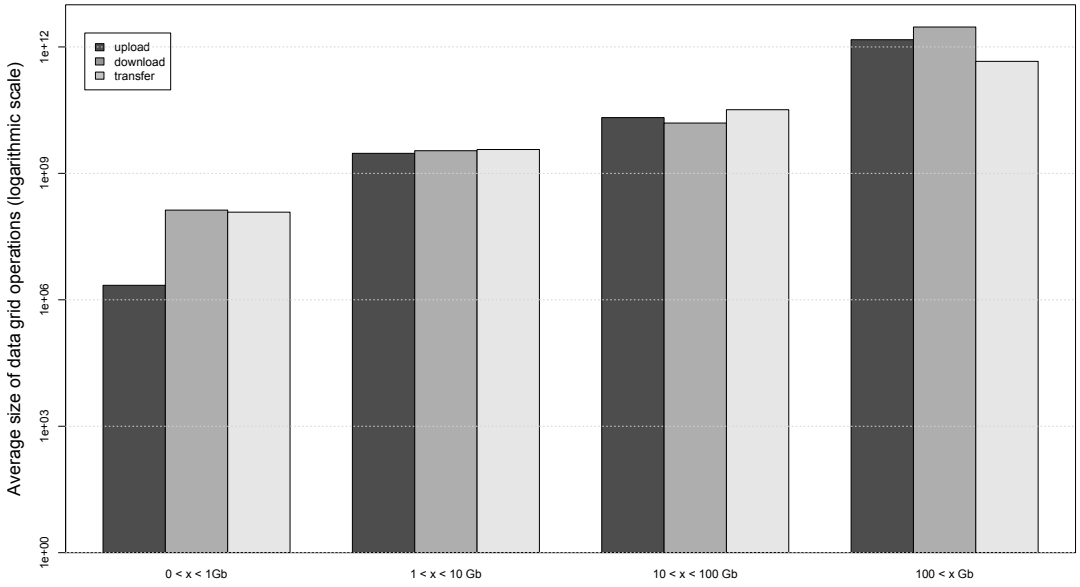

Figure 4.13: Average size of data grid operations partitioned by data grid operation size for the year 2015 (Logarithmic scale).

For further analysis the data grid operations are separated into *local* and *remote* operations. A data grid operation is considered as local if the source and destination is within the same local network. This is the case for computing elements accessing data from the storage element in the same data center, or users accessing data from storage elements in the same university network. In these cases the networks are usually over-provisioned and thus are not the bottleneck of the data grid operations. In contrast remote operations are computing elements or users accessing storage elements in remote data centers, thus having to use wide area network transfers over dedicated networks or the internet. In these cases, the network part of the workflow can be performance critical to the whole operation. The distinction is made based on the $localSite$ and $remoteSite$ attribute described in Table 4.1. Local and remote operations have to be treated separately in further analysis, as from a modeling point of view they involve different components.

Figure 4.14 shows the number of data grid operations for local and remote upload/download operations. The bar plot is also partitioned on operation size. In all partitions there are orders

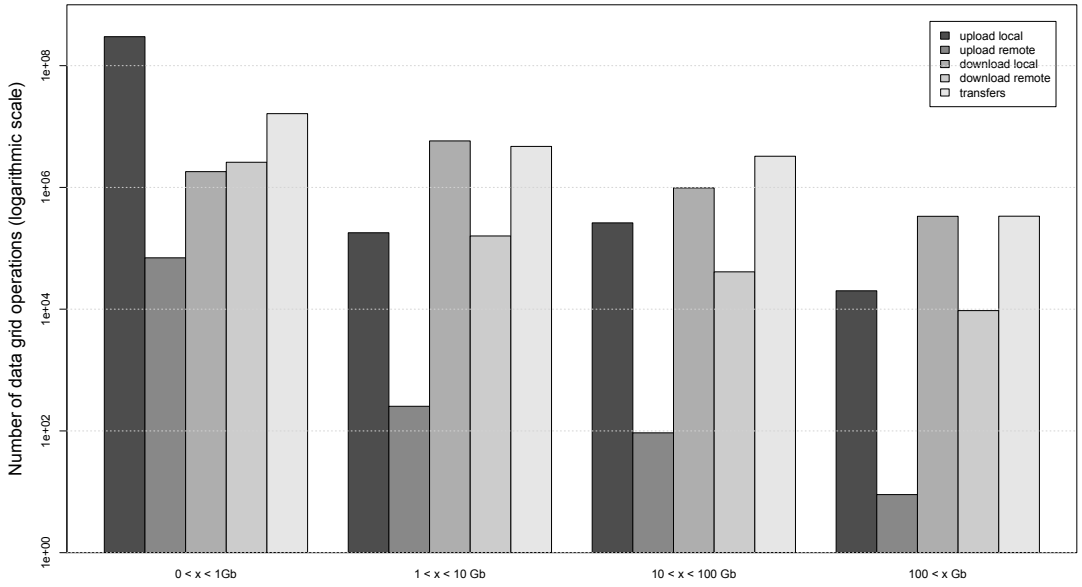

Figure 4.14: Number of data grid operations per operation type partitioned by data grid operation size for the year 2015 (Logarithmic scale).

of magnitude more local upload operations than remote upload operations, which is expected as the pilot of the workload management system writes the analysis output to local storage. Similar for all but the smallest partition there are more local download operations than remote download operations. This is also due to the same reason, that a pilot gets usually dispatched to where the data is, thus reading the data from the local storage element.

While the Rucio internal transfers, executed by FTS [Kiryanov et al., 2015], are more than 99% done with the LCG access protocol, different protocols are used for the user uploads and downloads. This is potentially critical to the performance of the read and write operations, due to the overhead of the protocols themselves. Also depending on the protocol, different servers are used for a storage endpoint and thus may deliver different performances due to hardware or utilization. Figure 4.15 shows an overview of the amount of data grid operations per protocol used based on the external traces. The previously described effect of having more upload data grid operations is also visible here. The largest protocol, both in uploads and downloads is from the LHC computing grid protocol stack (LCG, GSIFtp, GFAL, SRM) mostly based on

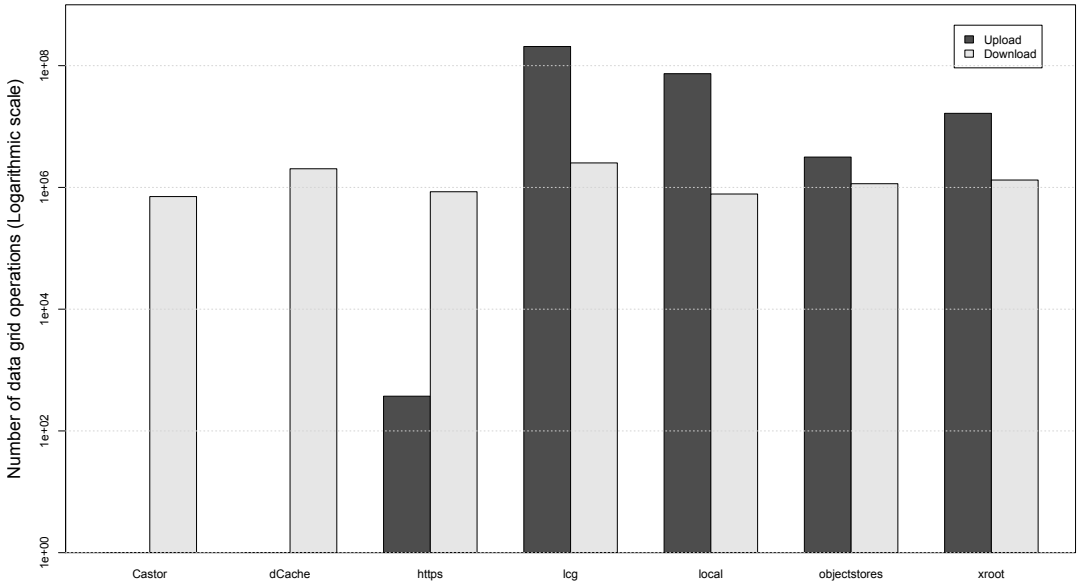

Figure 4.15: Number of data grid operations per protocol for the year 2015 (Logarithmic scale).

the Globus toolkit by Foster [2006]. The Castor (CERN Advanced Storage Manager protocol [CERN, 2016]) protocol as well as the dCache protocol by Fuhrmann and Gülzow [2006] are only used for downloading data. The reason for this is that in the ATLAS data grid Castor is only used for tape-based storage systems into which users have no permission to write. All write operations to the tape system are done using storage to storage transfers via replication rules and thus only shows up in the internal traces. For dCache the reason seems to be that the protocol offers advantageous reading performance but users choose different protocols for writing to the storage systems. Another observation is that the protocols for downloads are more or less equally used while there are strong differences for the upload protocol. A general larger amount of upload data grid operations is explained by the previously detailed effect of much smaller upload operations due to the workload management system workflow. But there are also large differences between the upload protocols, with the LCG protocol stack being used the most followed by local protocols (POSIX based) and with https as the least used protocol. At the time of writing this thesis the only protocol used for objectstores is Amazon S3 by Palankar et al.

[2008]. The xrootd protocol by Dorigo et al. [2006] also amounts for an order of magnitude more upload operations than download operations, also caused by the previously mentioned effect.

The next step of this quantitative analysis is to investigate the actual performance of the operations. The performance metric in focus is the response time of data grid operations, or metrics related to it such as transfer rates. This analysis shows how much time the system spends in different parts of the workflow and to some extend gives evidence to the complexity of said workflows. The analysis is based on aggregated time spent in a workflow, in contrast to end-to-end time spent in the workflow. Many steps in the workflow are executed concurrently, thus to most useful measurement in respect to this analysis is aggregated time spent in the workflow. The specific calculation will be presented for each timing period definition later on in this section. This information is essential to make a decision which system components should be included in a system model and even what kind of modeling approach might be applicable to certain processes in the workflow. To achieve this, the workflows for the download-local, download-remote, upload-local, upload-remote as well as transfer will be partitioned into logical consistent timing periods, where each timing period corresponds to one, or multiple, components. As both the internal as external trace archive only offers a limited number of observed timestamps, these periods cannot be chosen completely arbitrary. The measurements for each period are then partitioned based on the data grid operation size defined by: $0 < x < 1$ Gb, $1 < x < 10$ Gb, $10 < x < 100$ Gb and $100 < x < \infty$ Gb.

The first workflow analyzed is the **read** workflow, which based on the data grid reference architecture by Allcock et al. [2002] is defined as follows:

(i) The *metadata service* is queried for the list of logical file names.

(ii) The *replica management service* is queried for available replicas for each file.

(iii) The *replica selection service* selects the optimal replica for each file.

(iv) The files are read from the storage system.

(v) The files are transferred over the network.

(vi) Data integrity validation of the transferred files.

This is roughly mappable to the rucio download (local and remote) workflow depicted in Figure 4.7 as well as the PanDA pilot get workflow in Figure 4.9. The following periods are defined:

- **resolve**: This period involves the steps (i) and (ii) of the data grid reference which are responsible for resolving the logical identifier into the available replicas. The period starts with the t_{submit} timestamp and ends with the t_{start} timestamp. As this workflow step is only executed once per data grid operation, both timestamps are similar for all traces and the period is defined as $\Delta t = t_{start} - t_{submit}$
- **select**: This period represents the replica selection (step (iii)) of the data grid reference architecture. The period is enclosed by timestamp t_{start} and $t_{transfer}$. This step is also only executed once per data grid operation, thus the period is defined as $\Delta t' = t_{transfer} - t_{start}$
- **transfer**: The transfer period represents both the fetching of the data from the storage system (step (iv)) as well as the transfer of the data over the network (step (v)). It is enclosed by the timestamps $t_{transfer}$ and $t_{validate}$. As the transfers are executed concurrently in the workflow, the aggregated time spent in the workflow is defined as $\Delta t'' = \sum_{i=1}^{n} t_{validate}(i) - t_{transfer}(i)$
- **validate**: The validation period comes at the end of the workflow (step (vi)). It is enclosed by the timestamps $t_{validate}$ and t_{end}. Also the validation step is executed concurrently in the workflow. The period is defined as $\Delta t''' = \sum_{i=1}^{n} t_{end}(i) - t_{validate}(i)$

The full workflow, timestamps, and periods are shown in Figure 4.16.

The analysis is conducted twice, once for the download-local and once for the download-remote workflow. The results for the local read workflow are shown in Table 4.3. vol describes the aggregated volume of each data grid operation, thus the value defines into which partition the data falls into. In the first row, only local data grid download operations of a size between 0 and 1 Gb are described. n describes the number of data grid operations in this partition while $\overline{vol}$ describes the mean volume of these data grid operations. The bottom row aggregates all

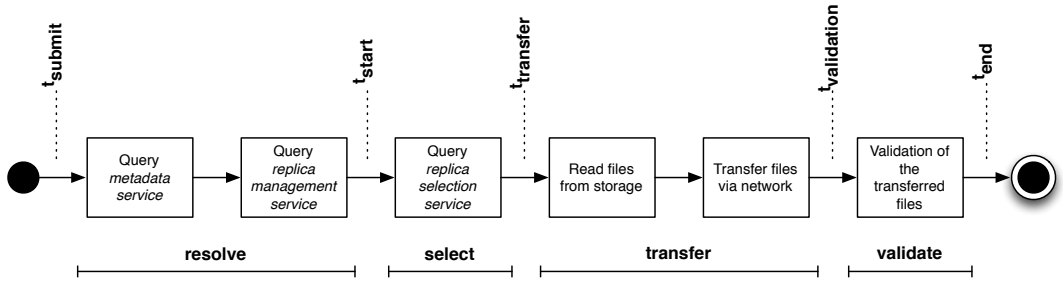

Figure 4.16: Periods of the read workflow.

			resolve		select		transfer		validate	
vol [Gb]	n	$\overline{vol}$	$\tilde{\Delta}t$ [s]	$\sigma(\Delta t)$	$\tilde{\Delta}t'$	$\sigma(\Delta t')$	$\tilde{\Delta}t''$	$\sigma(\Delta t'')$	$\tilde{\Delta}t'''$	$\sigma(\Delta t''')$
[0, 1)	1.4M	0.2	0.016	7.3	~ 0	24.8	4.7	4955	0.4	54.5
[1, 10)	4.3M	3.5	0.014	2.5	~ 0	9.8	19.9	495	8.1	253
[10, 100)	0.7M	13	0.016	2.2	~ 0	27.5	102	8600	35.2	789
[100, ∞)	0.1M	2554	0.019	0.5	2.46	284	1220	20699	584	19949
*	6.5M	4.2	0.014	4.14	~ 0	60.3	12.34	4124	5.8	1590

Table 4.3: Performance analysis statistics of the local read workflow.

partitions and shows the statistics for the full dataset. For each of the four periods the median time $\tilde{\Delta}t$ is shown, which is the time the operation spends in this part of the workflow. Also the standard deviation $\sigma(\Delta t)$ is given.

A detailed boxplot of the obtained performance analysis is also shown in Figure 4.17. As a first observation it is immediately obvious that the two dominating parts of the workflow over all partitions are the transfer and validation periods. The resolve and the select periods are insignificant to the performance of the operation. The select period of the workflow is instantaneous, which makes sense for these operations as they are initiated as local downloads. Only the local replica is eligible for a download. No selection has to be done. The time spent in this period is in the microsecond area. Only for the largest partition is the timing in the area of multiple seconds. This is most likely due to the very large number of transfers being requested. The transfer period, even for the small partitions, shows a significantly higher median and mean duration. The high standard deviation is more interesting as strongly scattered data also hints to a

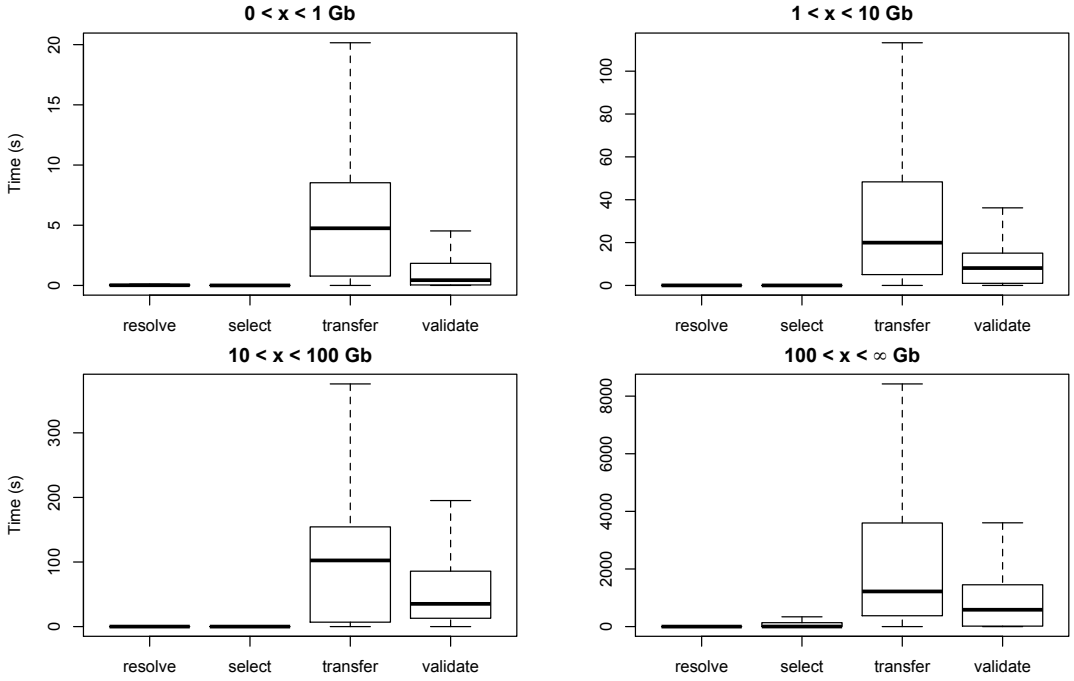

Figure 4.17: Performance analysis of the local read workflow.

more problematic relation of the data and a higher complexity when modeling it. For example, a strong linear relation between the size and the duration can be easily represented by a simple linear model. To investigate if there is any linear relation between the volume and the timing the Pearson product-moment correlation coefficient, as shown in Equation 4.1, is used.

$$\rho_{X,Y} = \frac{COV(X,Y)}{\sigma_X \sigma_Y} \tag{4.1}$$

The calculation of the pearson product-moment correlation coefficient for each of the workflow periods is shown in Table 4.4. Most of the data shows no correlation at all. No linear relation between the size and the time of the transfer is found. Only for the very large partition a small correlation is found. Interestingly, also for the validation workflow, which is strongly dependent on the used checksum algorithm, no linear correlation is found between the size and the duration of the workflow. Judging from this plots the resolve and select period are, from a duration point of view, not really relevant to the performance of the operation, as they take an insignificant amount of time. For the local read workflow the most critical periods are transfer and validation which identifies the local network, storage, as well as data integrity validation components to be

vol [Gb]	resolve $\rho_{\Delta t,vol}$	select $\rho_{\Delta t',vol}$	transfer $\rho_{\Delta t'',vol}$	validate $\rho_{\Delta t''',vol}$
[0, 1)	-0.008	-0.01	0.001	0.03
[1, 10)	-0.0001	0.0004	0.087	0.094
[10, 100)	0.003	0.119	0.024	0.111
[100, ∞)	-0.0002	0.711	0.493	0.107

Table 4.4: Pearson product-moment correlation coefficient of the local read workflow.

			resolve		select		transfer		validate	
vol [Gb]	n	$\overline{vol}$	$\tilde{\Delta t}$ [s]	$\sigma(\Delta t)$	$\tilde{\Delta t'}$	$\sigma(\Delta t')$	$\tilde{\Delta t''}$	$\sigma(\Delta t'')$	$\tilde{\Delta t'''}$	$\sigma(\Delta t''')$
[0, 1)	1.9M	0.03	0.549	34	4.02	210	7.0	815	0.01	32.3
[1, 10)	0.1M	3.2	0.82	27.5	12.0	378	410	3195	12.2	406
[10, 100)	28k	31	0.94	26.4	102	543	4092	13586	204	2743
[100, ∞)	5400	300	1.04	9.9	377	770	28976	76800	2506	19921
*	2M	1.4	0.5	33.2	5	256	10	6589	0.03	1285

Table 4.5: Performance analysis statistics of the remote read workflow.

the most significant ones. These components must be accurately represented in a system model to achieve precise results.

The results for the remote read workflow are shown in Table 4.5 and a detailed boxplot is displayed in Figure 4.18. A couple of differences immediately stand out when comparing with the local download workflow statistics. For one, the select period shows much larger durations compared to the local select period. This is because for the local workflow the selection can be skipped, while for the remote workflow the replica selection service has to be contacted to know which replica to select. A second observation is that the validation period duration seems to be much smaller than for the local workflow, which is somewhat unintuitive. When looking at the data in more detail it becomes clear that the average volume, per partition, is quite different in both workflows. In the remote workflow, for the smallest partition, the average data grid operation is only 30 Mb in size, while for the local workflow it is 200 Mb. In the remote workflow very often only a couple of files are transferred remotely as the rest is fetched locally hence the smaller volume. This is the reason why the validation period is much smaller. For

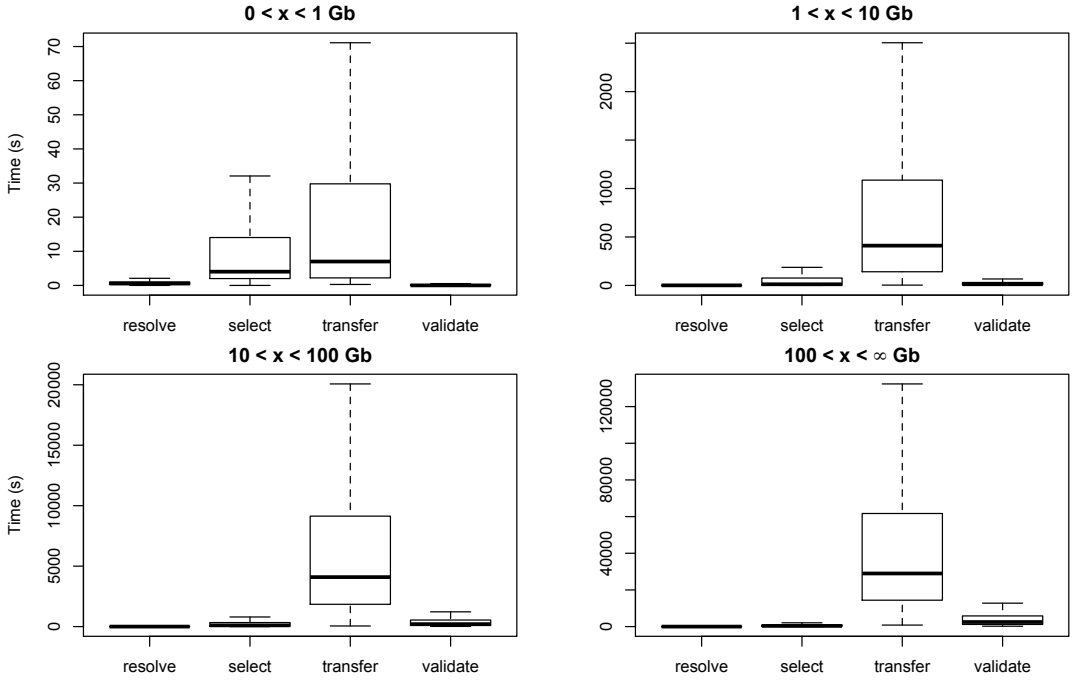

Figure 4.18: Performance analysis of the remote read workflow.

vol [Gb]	resolve $\rho_{\Delta t,vol}$	select $\rho_{\Delta t',vol}$	transfer $\rho_{\Delta t'',vol}$	validate $\rho_{\Delta t''',vol}$
[0, 1)	0.003	0.105	0.141	0.086
[1, 10)	0.005	0.078	0.174	0.102
[10, 100)	0.013	0.123	0.342	0.175
[100, ∞)	-0.001	0.263	0.618	0.297

Table 4.6: Pearson product-moment correlation coefficient of the remote read workflow.

the second partition ($1 < x < 10$ Gb) the volumes are about the same as well as the validation duration. However, the transfer duration is much higher for the remote workflow, which is also expected due to the usage of the network infrastructure.

Table 4.6 investigates again the linear relation between size and time for the different workflow periods, using the pearson product-moment correlation coefficient. The result is quite similar to the local download workflow. Almost no correlation except for the largest partition of the transfer period exists. This also suggests that for most of the workflow models no simple

linear model would suffice. Due to the much larger transfer durations the other periods become less significant as the majority of the workflow duration is spent in the transfer workflow. However, for the smallest partition, which holds the largest amount of operations, the select period gets even more dominant compared to the local workflow. As the transfer duration is disproportionally higher the validation period is less critical. The resolve part does not have much of an impact to the overall workflow duration. This makes the network and storage components the most significant ones for this workflow, but also a representation of the replica selection service component should be included in the system model.

The data grid reference architecture for the *write* workflow is defined as follows:

(i) The logical file names are registered with the *metadata service*.

(ii) The (unavailable) replicas are registered with the *replica management service*.

(iii) The files are transferred via the network.

(iv) The files are written to the storage system.

(v) The *replica management service* is notified about the availability of the replicas.

For the `rucio upload` workflow (Figure 4.8) as well as the PanDA pilot put workflow (Figure 4.10) this maps to the following periods:

- **register**: This period involves step (i) and (ii) which is responsible for registering the logical file and dataset information as well as the (unavailable) replica information on the server. The period starts with the t_{submit} timestamp and ends with the t_{start} timestamp. The full period is defined as $\Delta t = t_{start} - t_{submit}$

- **transfer**: The transfer period represents both the transfer of the data via the network (step (iii)) and the writing of the data to the storage system (step (iv)). It is enclosed by the timestamps $t_{transfer}$ and $t_{validate}$. As this workflow part is executed concurrently, the period aggregation is defined as $\Delta t' = \sum_{i=1}^{n} t_{validate}(i) - t_{transfer}(i)$

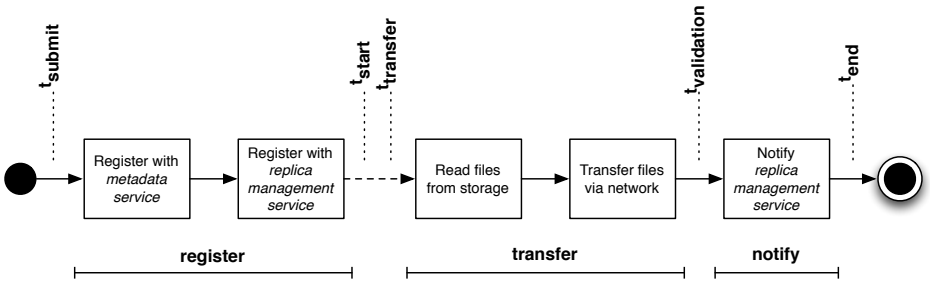

Figure 4.19: Periods of the write workflow.

			register		transfer		notify	
vol [Gb]	n	$\overline{vol}$	$\tilde{\Delta}t$ [s]	$\sigma(\Delta t)$	$\tilde{\Delta t'}$	$\sigma(\Delta t')$	$\tilde{\Delta t''}$	$\sigma(\Delta t'')$
[0, 1)	201M	0.003	0.639	15.4	3.76	278	0.467	27.3
[1, 10)	177k	2.9	6.9	182	233	9305	0.56	149
[10, 100)	260k	21	106	566	402	8575	27	164
[100, ∞)	20k	251	331	1768	23025	34257	634	809
*	201M	0.057	0.64	72.5	3.7	684	0.46	28

Table 4.7: Performance analysis statistics of the local write workflow.

- **notify**: The notify period is the last step in the workflow, when the server is notified about the availability of the replicas (step (v)). It is enclosed by the timestamps $t_{validate}$ and t_{end}. The period is defined as $\Delta t'' = \sum_{i=1}^{n} t_{end}(i) - t_{validate}(i)$

The full workflow partitioning is shown in Figure 4.19.

The results for the local write workflow are shown in Table 4.7 and a detailed boxplot is displayed in Figure 4.20.

At a first glimpse the results are unsurprising: Relatively small register and notification times and large transfer times. For the smallest partition, with data grid operation sizes below 1 Gb, the 3 periods have almost similar timings. The data grid operations in this partition are very small, with an average of only 3 Mb per operation, thus only a couple of files. The overhead in contacting the server for registration and notification is almost similar to the transfer time. These small data grid operations are due to the effect explained earlier, with an input dataset split up into many computation jobs resulting in many small upload operations to upload the analysis

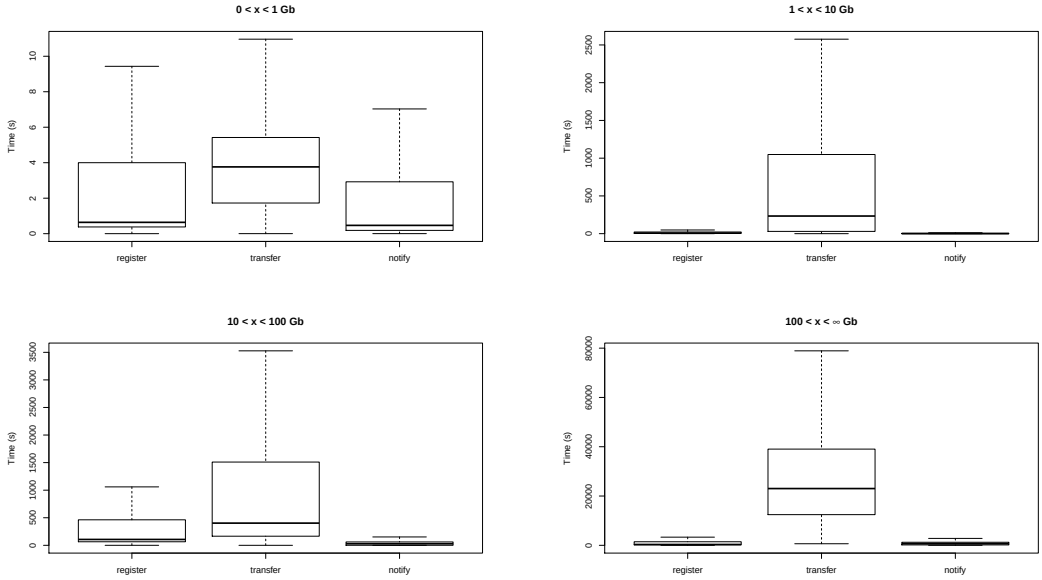

Figure 4.20: Performance analysis of the local write workflow.

output. With the larger partitions the majority of the workflow is clearly spent in the transfer partition. The registration and notification time for the largest partition of transfers larger than 100 Gb seems unreasonable high. This could point to a bottleneck in the server with handling very large lists of replicas to register. The linear relation between size and time for the different workload periods is shown in Table 4.8. All workflows and periods show no linear relation whatsoever. For the smallest partition all three parts of the workflow are significant, while for the other partitions the transfer workflow dominates the duration and thus is the most significant one. This means that a system model should focus on the storage and network components followed by slightly less important metadata service and replica management service components.

For the remote write workflow the system registered about 70 000 operations, but due to a bug in the tracing software stack all timing attributes were corrupted and unusable. However, due to this small amount of data grid operations the loss of these statistics is negligible.

While the storage to storage transfer workflow is not specifically mentioned in the data grid reference architecture, as it is somewhat considered a special case of the write workflow, it is still beneficial to specify the workflow and periods explicitly. The main difference of the transfer

vol [Gb]	register $\rho_{\Delta t,vol}$	transfer $\rho_{\Delta t',vol}$	notify $\rho_{\Delta t'',vol}$
[0, 1)	0.012	0.073	0.038
[1, 10)	0.267	0.080	0.055
[10, 100)	-0.014	0.413	0.090
[100, ∞)	-0.01	0.388	0.217

Table 4.8: Pearson product-moment correlation coefficient of the local write workflow.

workflow to the write/upload workflow is that for the upload there is no scheduling done and the transfer is executed immediately, while for the transfer workflow the data management system decides when to schedule the transfer. The workflow is defined as follows:

(i) The *metadata service* is queried for the list of logical file names.

(ii) The *replica management service* is queried for available replicas for each file.

(iii) The transfer requests are registered with the *replica management service*.

(iv) The transfer request is read from *replica management service*.

(v) The transfer request is submitted to the *transfer service*.

(vi) The files are transferred via the network.

(vii) The files are written to the storage system.

(viii) The *replica management service* is notified about the availability of the replicas.

For the rucio transfer workflow (Figure 4.11) the following periods are selected:

- **submit**: This period involves step (i) to (v) which is responsible for resolving the dataset, registering the replicas and requesting the transfers from the replica management service and finally submitting the transfer request to the transfer service. The period starts with the $t_{created\_at}$ timestamp and ends with the $t_{submitted\_at}$ timestamp. The workflow is only executed once per data grid operation and the period is defined as $\Delta t = t_{submitted\_at} - t_{created\_at}$

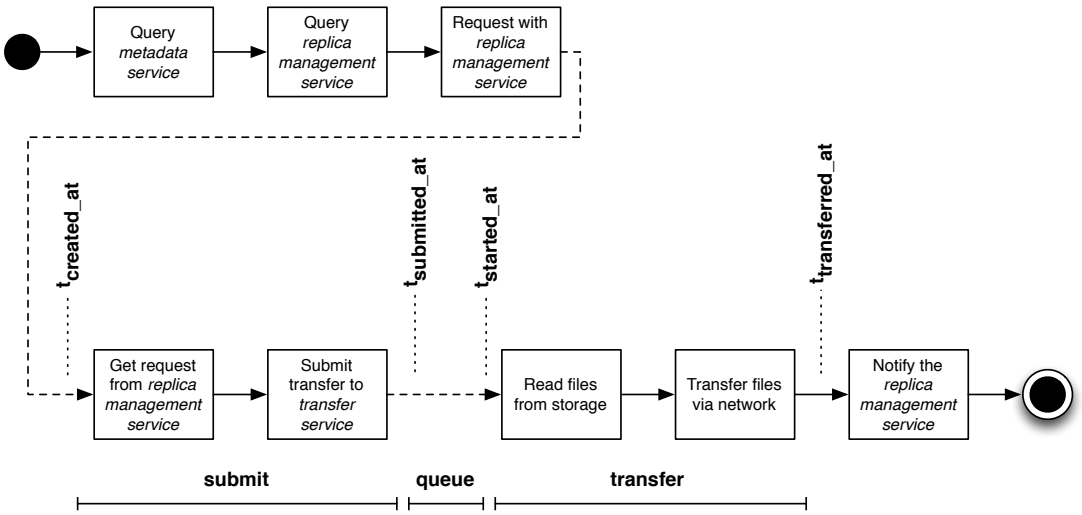

Figure 4.21: Periods of the transfer workflow.

- **queue**: The queue period represents the time the transfer sits in the waiting queue of the transfer service, until the actual flow of data starts. This is in between step (v) and (vi). The period is enclosed by the timestamps $t_{submitted\_at}$ and $t_{started\_at}$. The workflow is executed once per file transfer, thus the aggregated period is defined as $\Delta t' = \sum_{i=1}^{n} t_{started\_at}(i) - t_{submitted\_at}(i)$

- **transfer**: The transfer period represents both the transfer of the data via the network (step (vi)) and the writing of the data to the storage system (step (vii)). It is enclosed by the timestamps $t_{started\_at}$ and $t_{transferred\_at}$.
  The period is defined as $\Delta t'' = \sum_{i=1}^{n} t_{transferred\_at}(i) - t_{started\_at}(i)$

Unfortunately there is no timestamp taken when the server is notified about the success of the transfer operation by the transfer service. The workflow is shown in Figure 4.21.

The statistics for the transfer workflow are shown in Table 4.9 and the boxplot is displayed in Figure 4.22.

The first characteristic is the relatively large submission time for the partition of transfers smaller than 1 Gb. Overall the submission time is constant, as it is a simple push of transfer requests to the file transfer service, but in the case of the smallest partition the submission takes

vol [Gb]	n	$\overline{vol}$	submit $\tilde{\Delta t}$ [s]	submit $\sigma(\Delta t)$	queue $\tilde{\Delta t'}$	queue $\sigma(\Delta t')$	transfer $\tilde{\Delta t''}$	transfer $\sigma(\Delta t'')$
[0, 1)	12.9M	0.1	352	93191	5	89857	56	414058
[1, 10)	3.2M	3.5	274	101103	331	192539	953	851624
[10, 100)	1.7M	31.7	299	175085	3465	341279	5339.5	977264
[100, ∞)	127k	344	316	574428	10384	552485	49694	2617819
*	17.9M	5.8	331	115515	6	147154	203	628377

Table 4.9: Performance analysis statistics of the transfer workflow.

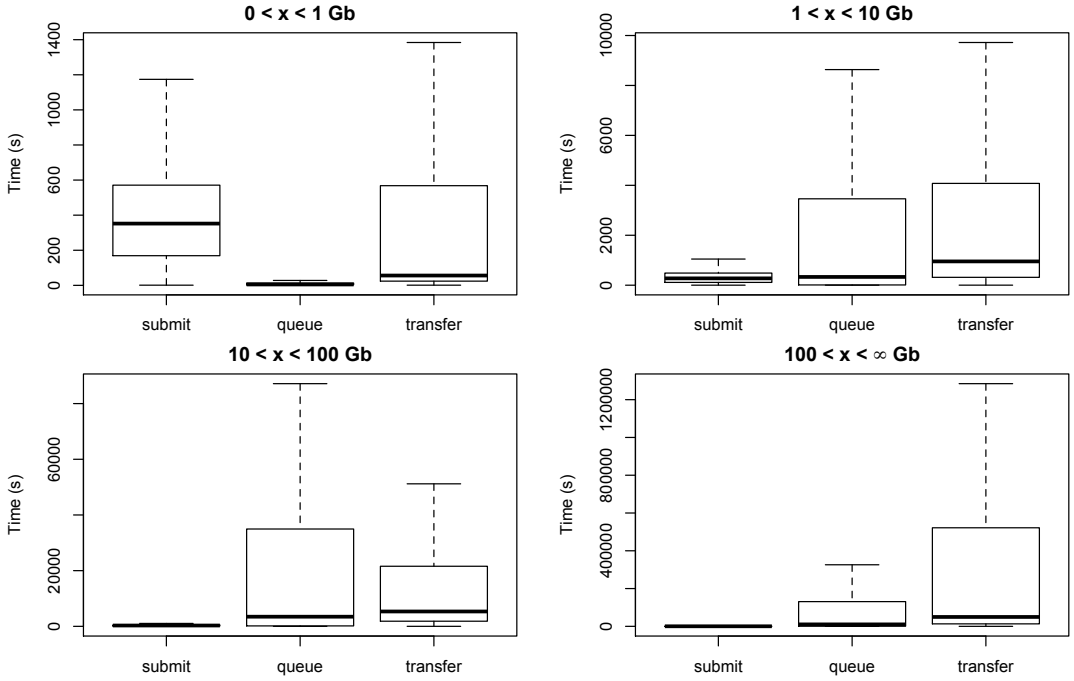

Figure 4.22: Performance analysis of the transfer workflow.

the biggest part of the overall workflow, even bigger than the transfer duration. The queue time, which is the time a transfer is kept in the queue until it is actually executed, scales with the volume size. This can be explained by the number of files being larger for larger transfer requests, thus the aggregated time these files stay in the queue is also larger. For the $1 < x < 10$ Gb and the $10 < x < 100$ Gb partition the queuing time is similar to the actual transfer time. Thus the workflow spends the same amount of time in queuing transfers and actually transferring

vol [Gb]	submit $\rho_{\Delta t,vol}$	queue $\rho_{\Delta t',vol}$	transfer $\rho_{\Delta t'',vol}$
[0, 1)	-0.001	0.086	0.035
[1, 10)	0.0008	-0.025	-0.002
[10, 100)	0.007	-0.038	0.043
[100, ∞)	0.009	0.085	0.185

Table 4.10: Pearson product-moment correlation coefficient of the transfer workflow.

them. However, this makes sense when looking at the Rucio queue statistics, where on average, half a million transfers are kept in the queue. Only the largest partition of transfers, bigger than 100 Gb, shows the effect one would expect from the start. The workflow is dominated by the transfer time, followed by the queuing time and a relatively insignificant submission time. The standard deviations for all time frames in all partitions is very high. To some extent this is also expected as the transfers are executed on many different network links with drastically different performance measures, thus resulting in very different transfer times.

The linear relation between the different timeframes and the transfer size is analyzed in Table 4.10 using the Pearson product-moment correlation coefficient. Again, all the workflows show no linear relation between the operation timing and size. This also suggests that for most of the workflows in the transfer workflow no simple linear model would be sufficient. The submit period is only significant in the smallest partition of operations. All the other partitions are dominated by the queue and transfer period, with slightly more time spent in the transfer part of the workflow.

Component selection

The set of identified components to be included in the hybrid simulation model is shown in Table 4.11. Storage and network components are included for all workflows. Some other components of the workflow can be completely skipped though, as they have no significant impact on the performance of user operations.

If a component model is not necessary for the performance estimation of a given workflow, a functional model will be used instead of the actual simulation model. Functional models

	local-read	remote-read	write	transfer
Metadata service			✓	
Replica management service			✓	
Replica selection service		✓	-	
Storage interaction	✓	✓	✓	✓
Network interaction		✓	✓	✓
Date integrity validation	✓	✓	-	-
Transfer service	-	-	-	✓

Table 4.11: Selected components to be represented as component models in the hybrid simulation model.

do not offer any performance prediction, thus they are not useful in terms of the simulation objective, but they do produce output operands based on the given input operand. For example, a functional model for the metadata service component is a simple persistent dictionary. The functional model can be queried for the files contained in a dataset, which is information needed in subsequent steps in the workflow, but the model does not return any performance estimation how long said query takes. From the simulation point of view the functional model returns its answer instantaneously.

The details of the specific component models are presented in Chapter 5 for the component model of the storage interactions, Chapter 6 for the component model of the network interactions, Chapter 7 for the component model of the data integrity validation workflow, and Chapter 8 for the component models of the different service components.

Part II

Modeling system components

CHAPTER 5

Modeling storage systems

5.1 Introduction

Storage systems are one of the core resources of data-intensive systems. A data-intensive system interconnects a large amount of mass-storage systems, located in different data centers worldwide, to offer the users efficient access to large storage capacities. As users establish different workflows they also have different requirements to these storage systems, their data availability, and their access performance. For example, long-term storage of data where access time is non-critical, for cold storage of data, medium-term storage for data which is stored until it gets distributed to long-term storage systems, or high performance short-term storage areas, for data which is staged to computation elements. These workflows lead to the deployment of different technologies, such as magnetic tape libraries, network attached storage in the form of disk arrays for hard disk drives and solid-state drives, or even distributed file systems, which are integrated into the data-intensive system.

The objective of this chapter is to establish a modeling technique which is able to build an accurate component model for all storage systems used in data-intensive systems. The modeling technique has to be in accordance with the requirements specified in Section 4.2. The model is built based on the available observational data and predicts individual *read* or *write* operation metrics, such as the transferrate.

Modeling storage system is a research field with a long history. The field developed alongside the success of hard disk drives and the need of manufacturers and enterprise users to model and simulate their storage system needs. Unsurprisingly, researchers were concentrating on the simulation of single disk drives, such as Ruemmler and Wilkes [1994]. Over the years also models for very specific systems were introduced, such as the work of Hillyer and Silberschatz [1996] modeling the DLT4000 tape drive system. Only later efforts were made to simulate multiple disks in a system, such as Barve et al. [1999].

The remainder of this chapter is structured as follows. At first the most prominent modeling techniques for mass storage systems are introduced. For each technique the underlying concept is presented and the applicability of the technique in the context of data-intensive systems is discussed. Section 5.2 presents related work in modeling storage systems analytically. Section 5.3 introduces execution-driven simulation methods for storage systems, and section 5.4 discusses black-box storage models based on Classification and Regression Trees. Furthermore the chapter continues in Section 5.5 presenting different machine learning techniques, which offer a different approach to storage system modeling. Finally the applicable modeling techniques are evaluated based on historic workloads from the ATLAS data grid in Section 5.6. Section 5.7 shows the conclusions of the chapter.

5.2 Analytical models

Idea

There are numerous articles about how to model storage systems analytical, but for the sake of conciseness this section focuses on Uysal et al. [2001] as the proposed approach is very transparent and straightforward. However, other analytical modeling techniques work very similar. Analytical storage models, at first for disk drives and later for mass storage systems, were among the first type of storage models. Only later on techniques based on simulation or machine learning were used, which are presented in this chapter.

Uysal et al. [2001] created an analytical model which is able to predict the throughput of individual operations of a disk array mass storage system. The authors put much effort into val-

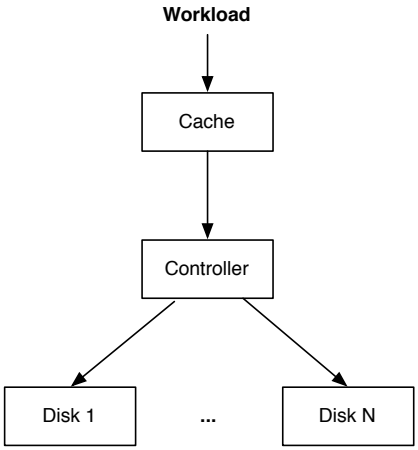

Figure 5.1: Decomposed component workflow for analytical storage model [Uysal et al., 2001].

idating their model against workloads taken from a real disk array system, the Hewlett-Packard SureStore E Disk Array FC60.

Their novel modeling approach is based on hierarchical decomposition of the internal architecture of the system. The idea is that individual components of the system are easier to model than their composition. However, this approach requires detailed domain-specific knowledge of the actual operated storage system.

Method

The disk array consists of up to 60 disks in up to six different trays. For failover reasons, the system consists of two controllers each having a fibre channel connection to the network. Each controller is connected to a battery-powered cache. The controller is then connected via an ultra wide SCSI bus to the disk trays.

The authors identified the *controller*, the *caches*, and the *disks* as individual components which have to be modeled individually. The decomposed component workflow is shown in Figure 5.1. Each component model transforms an input workload specification into an output specification that becomes the input for the next model in line. Therefore the models only communicate with each other by passing specifications. The cache component is the first in line to receive the workload specification. It will absorb some of the access requests (cache hits) and

relay the rest of the workload to the controller model. The controller translates logical unit (LU) disk accesses to individual disk accesses and relays these to the individual disk models. The disk models then compute their throughput estimators without using any lower level model.

The cache model receives the input specification as a list of I/O request streams. It outputs a list of streams which did not have any cache hits. In steady state the disks are only accessed when both read and write requests cause cache misses. Read requests are just kept in cache and write requests are only written to disk if not accessed. The $total\_cache\_size$ is defined in the system description, the n streams $S_1, S_2, ...S_n$ and their corresponding output streams $S'_1, S'_2, ...S'_n$ are part of the input specification. For the model's purposes the cache is divided into n parts of size $cache\_size(S_i)$ by:

$$cache\_size(S_i) = total\_cache\_size \cdot \left(\frac{request\_rate(S_i)}{\sum_{j=1}^{n} request\_rate(S_j)} \right) \tag{5.1}$$

The probability of a cache hit is then approximated by the probability that the number of bytes accessed by the system between two accesses to the same block is less than $cache\_size(S_i)$. The authors define this as the re-reference distance which is a distribution given in the input workload specification. The request rate is thus defined as:

$$request\_rate(S'_i) = request\_rate(S_i) \cdot P[re\_reference\_distance(S_i) > cache\_size(S_i)] \tag{5.2}$$

The controller model first limits the total rate of requests and the bandwidth of the requests by the following equations:

$$\sum_{i=1}^{n} request\_rate(S_i) \leq max\_controller\_throughput \tag{5.3}$$

$$\sum_{i=1}^{n} request\_rate(S_i) \cdot E[request\_size(S_i)] \leq max\_controller\_bandwidth \tag{5.4}$$

The values of $max\_controller\_throughput$ and $max\_controller\_bandwidth$ are part of the device description. In the studied system the authors use a RAID 1/0 configuration. Different equations are used for reads, large writes, and small writes. Also, in case of the limit being exceeded a queue length is calculated by the model. The read is rather simple, as the FC60 controller only reads whole stripe units from disk. Thus if an I/O request is smaller than a stripe

unit, the entire unit is read into the cache anyway. The number of stripe units read per read request is defined as:

$$disk\_accesses\_per\_read(S_i) = 1 + \frac{read\_request\_size(S_i)}{stripe\_unit\_size} \tag{5.5}$$

The $stripe\_unit\_size$ is also part of the system specification. The read request rate, for a specific disk j is then estimated by:

$$read\_request\_rate(S_{ij}) = \frac{disk\_accesses\_per\_read(S_{ij}) \cdot read\_request\_rate(S_i)}{LU\_disks} \tag{5.6}$$

As write requests have to be written at least to two disks for each logical unit, due to the RAID 1/0 operation, the $write\_request\_size$ for large writes is defined as follows:

$$write\_request\_size(S_{ij}) = 2 \cdot \frac{write\_request\_size(S_i)}{LU\_disks} \tag{5.7}$$

For small writes, the write request size is impacted by the stripe unit size as well. The write requests for large write operations can be coalesced into a single disk write request, thus:

$$write\_request\_rate(S_{ij}) = write\_request\_rate(S_i) \tag{5.8}$$

For small writes though the accesses touch multiple logical units, therefore:

$$write\_request\_rate(S_{ij}) = \frac{2 \cdot write\_request\_size(S_i) \cdot write\_request\_rate(S_i)}{write\_request\_size(S_{ij}) \cdot LU\_disks} \tag{5.9}$$

The queue length is defined as the number of requests outstanding per disk D_j. For simplification it is assumed that the queue length is divided between reads and writes according to the respective read and write rate. The queue length is calculated as follows:

$$queue\_length(S_{ij}) = queue\_length(S_i) \cdot \left(\frac{read\_accesses(S_i) + write\_accesses(S_i)}{LU\_disks \cdot request\_rate(S_i)} \right) \tag{5.10}$$

The disk model enforces throughput limits through inequality:

$$\sum_{i=1}^{n} read\_utilization(S_{ij}) + write\_utilization(S_{ij}) < 1 \tag{5.11}$$

The utilizations are calculated as follows, whereas the service times are part of the system description:

$$read\_utilization(S_{ij}) = read\_request\_rate(S_{ij}) \cdot disk\_read\_service\_time(S_{ij}) \tag{5.12}$$

$$write\_utilization(S_{ij}) = write\_request\_rate(S_{ij}) \cdot disk\_write\_service\_time(S_{ij}) \tag{5.13}$$

The disk read service time only depends on the datum being found in the disk cache, which is estimated as follows:

$$\begin{aligned} disk\_read\_service\_time(S_{ij}) =& (1 - disk\_cache\_hit\_prob) \cdot \\ & \left(disk\_read\_pos\_time(S_{ij}) + \frac{read\_request\_size(S_{ij})}{disk\_transfer\_rate} \right) \end{aligned} \tag{5.14}$$

The disk cache hit probability can be calculated based on the read ahead distance and the request size. The disk positioning time is estimated by:

$$disk\_read\_pos\_time(S_{ij}) = \frac{mean\_read\_disk\_seek\_time}{\sum_{k=1}^{n} queue\_length(S_{kj})} + \frac{disk\_rotation\_time}{2} \tag{5.15}$$

Furthermore the mean read disk seek time and the disk rotation time are also parameters of the device description. The model adjusts for the performance of sequential reads as there are no repositioning delays on the disk. Also the read ahead buffer is incorporated in the model. For the write positioning and service time only the first request in the stream experiences a seek time, the other requests just experience rotational delays.

$$\begin{aligned} disk\_write\_pos\_time(S_{ij}) =& \frac{disk\_rotation\_time}{2} + \\ & \frac{mean\_write\_disk\_seek\_time}{write\_run\_count(S_{ij}) \cdot \sum_{k=1}^{n} queue\_length(S_{kj})} \end{aligned} \tag{5.16}$$

The actual service time is then the sum of positioning and transfer times, where the disk transfer rate is a parameter of the system description.

$$disk\_write\_service\_time(S_{ij}) = disk\_write\_pos\_time(S_{ij}) + \frac{write\_request\_size(S_{ij})}{disk\_transfer\_rate} \tag{5.17}$$

The experimental evaluation is done with a FC60 mass storage system with 30 disks and two controllers in RAID 1/0 configuration. The system description, which is used to initialize the analytical model, is based on 11 parameters. The workloads injected into the system are created using a synthetic workload generator. The same workloads are then injected into a simulator

running the analytical model. The average error of the model predicting the throughput is at about 15% and does not exceed 30%. The highest relative error was detected at 42% but only occurring at a certain system setup.

Discussion

The discussion of each model will be guided by the requirements previously defined in Section 4.2.

The first requirement is the prediction of response times of individual read and write operations. The analytical model by Uysal et al. [2001] clearly meets this requirement. The authors even go a step further and estimate the individual I/O streams forming the file-level access. Although the model predicts the throughput of this streams, this can easily be transformed to the response time for a read and write operation.

The second requirement asks for the model to be black-boxes and that only external observations are used to construct the model. This requirement is clearly not met, as domain specific knowledge is necessary to even decompose the system but also internal measurements and specifications, such as the number of disks, transfer rates, and rotation rates are necessary to initialize the model. But even when assuming that this information is available, the model is clearly specified for a very small spectrum of types of systems. Thus in the context of a data-intensive system, only a very limited amount of storage systems would be applicable to the model as it is unclear how the model performs with other types of, disk-array based, systems. Besides the question of performance, it is furthermore undefined if such a model could be even initialized, even if only small alterations are done to the system. For example the usage of non-rotational disks while the model asks specifically for the disk rotation time.

The third requirement prohibits the injection of experimental workloads to measure the system under different load conditions. While the authors do use this technique to achieve more accurate measurements for some of the system specifications, it is plausible that the model is initializable without such workload injections.

Overall the model is not a good fit for usage in the context of data-intensive system, as it is just not generic enough to cover a wide range of storage system technologies.

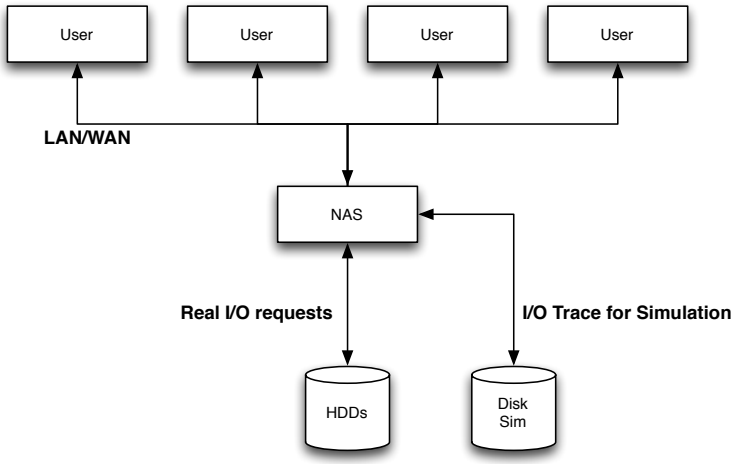

Figure 5.2: Execution-driven simulation overview [Kaeli, 2004].

5.3 Execution-driven simulation

Idea

The idea of execution-driven simulation models is that, the workload is partially executed in a real system while other parts are estimated using a prediction model. This technique originally comes from studying microcontroller architectures. In literature, there are multiple publications featuring execution-driven simulation for mass storage systems. This section specifically focuses on Kaeli [2004] due the simplicity of their approach, but the findings also apply to other execution-driven methods.

Method

Kaeli [2004] focus on simulation models of three types of mass storage systems: Direct Attached Storage (DAS), Network Attached Storage (NAS), and Storage Area Networks (SAN) systems. Figure 5.2 shows an overview of the execution-driven approach in the case of NAS. While some parts of the NAS system, like the RAID controller or network interfaces are not simulated, the real disk I/O operations are predicted using a hard disk model provided by DiskSim [Ganger et al., 2008]. DiskSim itself is an analytical simulation model for hard drives, which has to be

initialized using detailed hard disk characteristics.

When the NAS system is configured in normal mode the RAID controller of the system creates real I/O requests and relays them to the hard disks to retrieve the data. When configured in simulation mode no I/O requests are relayed to the hard disks, but instead I/O traces are extracted and fed into the DiskSim model, which reports back a performance estimator for this specific trace.

The advantage is that it is not necessary to simulate the full workflow, as parts of it are just executed in the real system. Usually this is done to quickly study caching techniques or the performance of certain RAID settings, without running a full benchmark suite on the real disk drives. It is also very useful to estimate the performance of newer disk drive models, without actually benchmarking the whole system with a new set of disks.

The authors validate their model against a NAS and a SAN setup on a Beowulf cluster [Sterling, 2002] using Western Digital 80GB hard disks. The relative simulation error of the model is consistently no more than 3%.

Discussion

The first requirement of predicting response times of individual read and write operations is met.

The second requirement, stating that the components should be seen as black-boxes and that only external observations are to be used, is not met. The DiskSim models require very characteristic information about the actual disk drives in use, which conflicts with the requirement. As some parts of the system are not modeled, but executed on the real system, it is debatable if this requirement is met. But even if it is, the method is still not a good fit for distributed data-intensive systems, as it would require access and model integration of all storage systems in the data grid. Even if this is scaled down and one only requires one storage system, per type of system, this is still an impracticable, and most likely impossible, amount of resources.

The third requirement is, in principle, respected. However, experimental workload injections are necessary for the construction of hard drive characteristics for the DiskSim models. Thus, if disk drives are used where no drive characteristic are available, the model can only be constructed by injecting experimental workloads.

Overall the model is impracticable in the context of data-intensive systems.

5.4 Black-box models using Classification and Regression Trees

Idea

Wang et al. [2004] decided to focus on machine learning techniques to model and predict the performance of storage devices. The authors use Classification And Regression Trees (CART) [Breiman et al., 1984], to model single disk devices as well as disk arrays. CART models are a form of black-box model, thus there is no knowledge about the internal composition of the storage device and the model is strictly built based on training of historic workloads. This offers the advantage that the modeling technique is generic thus it applies to any storage device.

Method

The detailed theory and background of CART models is presented in section 5.5. CART models approximate functions on a Cartesian space using constant functions. The goal of the model is to predict device performance as a function of I/O workload. The models input is a workload represented as a sequence of disk requests, with each request r_i described by four attributes: arrival time ($ArrivalTime_i$), logical block number (LBN_i), request size in number of disk blocks ($Size_i$), and read/write type (RW_i). As the model is purely trained on such structured workloads, it is irrelevant if the device is just a single disk drive or a RAID storage system with multiple disks.

The authors present two ways of generating the storage model. The first mode is request-level device models, which predicts per-request response time. The second way are workload-level models, which predict the aggregate performance of the entire workload. The transformation of the workload into a multi-dimensional feature space, which is exercised in the CART model to create the predictor, is shown in Figure 5.3.

The request-level device models predict the response time of individual requests based on a request description. Each request r_i in the input workload is transformed into a request descrip-

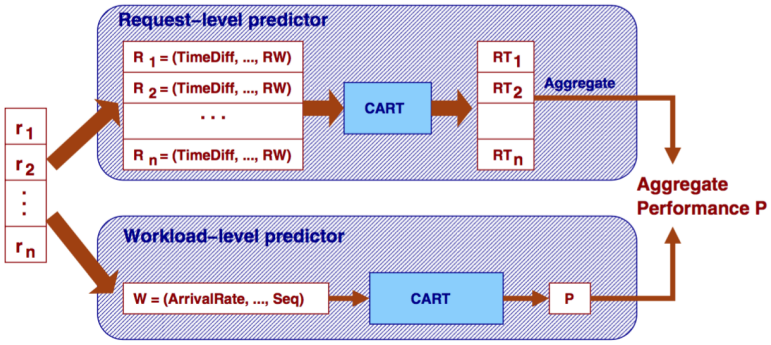

Figure 5.3: Workload transformation for CART models [Wang et al., 2004].

tion R_i, containing the following variables:

$$\begin{aligned} R_i = \{&TimeDiff_i(1), \ldots, TimeDiff_i(k), \\ &LBN_i, LBNDiff_i(1), \ldots, LBNDiff_i(l), \\ &Size_i, RW_i, \\ &Seq(i)\} \end{aligned} \tag{5.18}$$

where

$$TimeDiff_i(k) = ArrivalTime_i - ArrivalTime_{i-2^{k-1}} \tag{5.19}$$

$$LBNDiff_i(l) = LBN_i - LBN_{i-l} \tag{5.20}$$

The $TimeDiff$ measurements are used to describe the temporal burstiness of the workload at the moment when r_i arrives. $LBNDiff$ is used to measure the spatial locality of the data on the disk, to support accurate prediction of seek times on disk. The parameters k and l define how far the predictor looks back to measure bursts and data locality. Too small values of these variables lead to inferior device models, while too large values lead to unnecessarily long training and prediction times.

For the workload-level models the entire workload is transformed into a single workload description and the model predicts aggregate device performance directly. The workload de-

scription W contains the following attributes:

$$\begin{aligned} W = \{&AverageArrivalRate, \\ &ReadRatio, \\ &PercentageOfSequentialRequests, \\ &TemporalAndSpatialBurstiness, \\ &CorrelationsBetweenPairsOfAttributes\} \end{aligned} \tag{5.21}$$

The compression of the workload into these workload descriptors is more complex than for the request-level models, thus the difference between request-level and workload-level models is that the former is fast in training and slow in prediction, and the latter is the opposite.

The experimental results are based on on two different simulated devices: A single 9GB Atlas 10K disk with a rotational latency of 3ms and a RAID 5 disk array consisting of 8 Atlas 10K disks with 32KB strip size. In both cases no real system was used but only a validated model provided by DiskSim. The relative error $\frac{|\hat{Y}-Y|}{Y}$ is used for the evaluation. For the single disk model the average relative error for the per-request model is 19% and 15% for the per-workload model. The models for the disk array offer a similar performance.

Discussion

Requirement one is met, as the per-request level model is specifically predicting the response time of single requests. The model also goes one step further, as it models single logical block requests, thus it predicts at a sub-file level.

The second requirement of not using internal component knowledge is also met and demonstrated, as CART is a pure black-box modeling approach.

The third requirement of not injecting experimental workloads to generate the model is also respected. The authors identify inadequate training data as one of the major error sources of the model and mention that injected training workloads could improve this problem, however, the model also works without injecting these workloads.

5.5 Machine Learning techniques

Arthur Samuel [Phil Simon, 2013] defined machine learning as *the field of study that gives computers the ability to learn without being explicitly programmed.* It is the study of creating algorithms that learn from a given set of data and make predictions on a different set of data. Before the wider establishment of machine learning techniques, regression analysis was used to tackle most prediction problems, thus there is a strong overlap between machine learning and regression analysis. Regression analysis is a statistical method for investigating and estimating the relationship between variables. The goal of regression analysis is the creation of a function of the independent variables (or predictors) based on the dependent variables. This function is called a regression function.

In its simplest form the regression model assumes that for a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ of n statistical units the relationship between the dependent variable y_i and the independent variables x_i is linear. Thus the regression model takes the form:

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = x_i^T \beta + \epsilon_i, \qquad i = 1, \ldots, n, \tag{5.22}$$

where ϵ_i is used as disturbance term. An example of a linear regression fit is shown in the first plot of Figure 5.4. It can be observed that in this example the relationship between the two variables is almost linear and the resulting regression function represents an almost-perfect fit. However, when applied to data which does not represent a linear relation the regression function does not approximate the function at all, which is shown in the right plot of Figure 5.4.

In these cases non-linear regression methods have to be used. One of the most common approaches is non-linear least squares, which fits a set of m observations with a model that is non-linear in n unknown parameters ($m > n$). This approach assumes a set of m data points of form $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, and a model function $y = f(x, \beta)$. x additionally depends on n parameters of the form $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ with $m \geq n$. The objective is to find the vector β such that the resulting curve fits the data best while the sum of squares

$$S = \sum_{i=1}^{m} r_i^2 \tag{5.23}$$

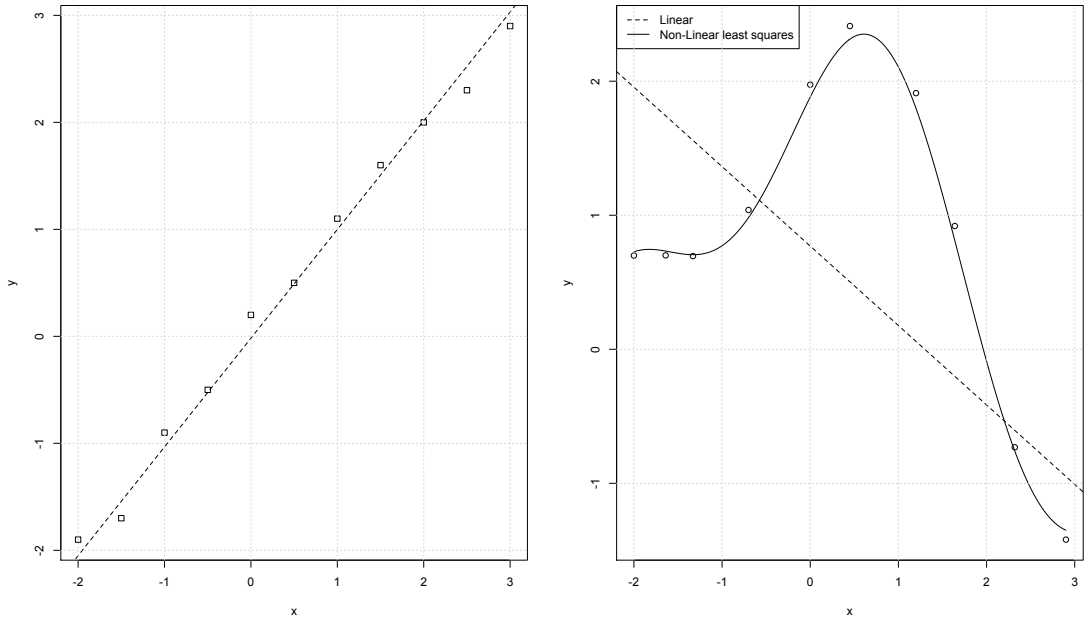

Figure 5.4: Examples of linear and non-linear regression.

is minimized. The residuals are given by $r_i = y_i - f(x_i, \beta)$. Consequently, the minimum of S occurs when the gradient of the function is zero. As the model consists of n parameters there must be n gradients of form:

$$\frac{\partial S}{\partial \beta_j} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_j} = 0 \qquad\qquad (j = 1, \ldots, n) \tag{5.24}$$

In non-linear systems, the derivates $\frac{\partial r_i}{\beta_j}$ do not have a closed solution. Therefore initial values must be chosen for these parameters and the values are obtained by approximation:

$$\beta_j \approx \beta_j^{k+1} = \beta_j^k + \Delta\beta_j \tag{5.25}$$

With each iteration the model is linearized by approximation to a Taylor series expansion:

$$f(x_i, \boldsymbol{\beta}) \approx f(x_i, \boldsymbol{\beta}^k) + \sum_j \frac{\partial f(x_i, \boldsymbol{\beta}^k)}{\partial \beta_j} \left(\beta_j - \beta_j^k\right) \approx f(x_i, \boldsymbol{\beta}^k) + \sum_j J_{ij}\, \Delta\beta_j. \tag{5.26}$$

The function J in aboves equation is called the Jacobian [Seber and Wild, 1989], which is a function of the independent variable and the parameters represented in β. In terms of a linearized

model $\frac{\partial r_i}{\partial \beta_j} = -J_{ij}$. When transforming the Jacobin into its residuals and substituting them into the gradient equation, they become n simultaneous linear equations, which can be written in matrix notation:

$$(J^T J)\Delta\beta = J^T \Delta y \tag{5.27}$$

These functions form the basis of any algorithm fitting a non-linear model, such as the Gauss-Newton algorithm [Seber and Wild, 1989].

However, the application of regression analysis becomes extremely difficult with an increasing number of variables and a complex interaction between these variables, especially when the predictors have a non-linear relationship. Also many non-linear regression algorithms usually require more manual input than their non-supervised machine learning counterparts. Thus, the application of machine learning in the big data domain becomes more and more important, especially when predictors are the objective of the research and not the discovery of inferences.

In the remainder of this chapter a subset of machine learning techniques, namely Classification and Regression Trees, Neural Networks and Support Vector Machines are discussed. There also exists a wider body of predictive modeling techniques, such as Radial basis functions [Buhmann, 2000], Naïve Bayes [Russell et al., 2003], or k-nearest neighbors [Altman, 1992], but these methods were excluded as the three presented methods are most widely used and, according to related work, most often deliver superior results.

Classification and Regression Trees

Classification and Regression Tree (CART) analysis is an umbrella term referring to classification tree and regression tree analysis, introduced by Breiman et al. [1984]. Both methods are a form of decision tree learning. In the context of this work, the focus lies on regression trees, but the basic principles of both analysis methods are the same. Their main difference is that in a classification tree model the target variable can only take a finite set of values, while in regression trees the target variable can take continuous values.

The general idea of a decision tree is that each interior node corresponds to one of the input variables (dependent variables) and there are edges to children for each of the possible values of the variable. The leaf nodes represent the value of the predictor given the values of the dependent

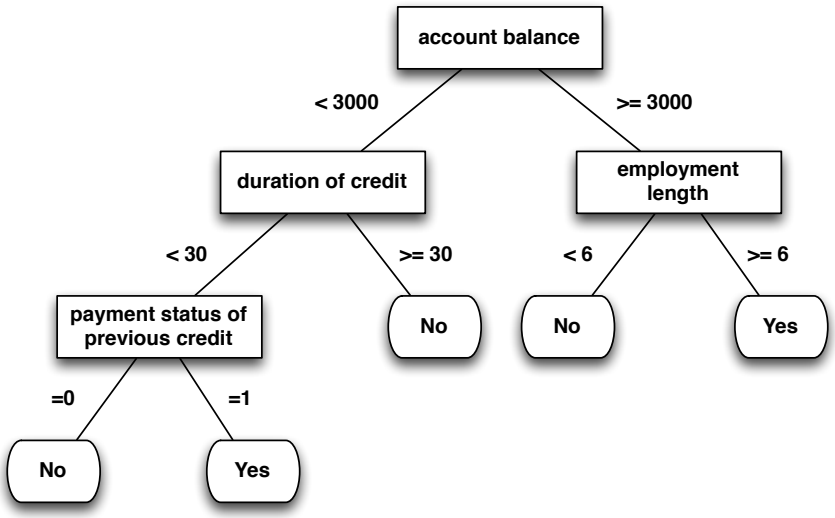

Figure 5.5: Example of a decision tree.

Algorithm 1 Grow phase of the top-down induction of decision trees algorithm [Breiman et al., 1984].

```
1:  function GROW_TREE(T)
2:      N ← create a new node
3:      N.class ← most common class in T
4:      N.test ← best attribute
5:      if N.test is not good enough then
6:          mark N as a leaf and return N
7:      for each value v_j of N.test do
8:          examples_j ← examples with N.test = v_j
9:          if examples_j is empty then N.branch_j ← N.class
10:         else N.branch_j ← build_tree(examples_j)
        return N
```

variables in the path from the root to the leaf. Figure 5.5 shows an example of a decision tree where a loan decision has to be made. In this case the decision tree is a classification tree, as each leaf is labeled with a single class.

The most common approach in learning decision trees is a greedy algorithm called top-down induction of decision trees by Quinlan [1986]. The method is split into two parts, a growing phase and a pruning phase. The growing phase is shown in Algorithm 1. The key step is shown in line 4 and 5, the selection of the best attribute. There are many different metrics in literature

how to identify the best attribute. Most commonly used are Gini impurity, Information gain and Variance reduction. Gini impurity takes the distribution of labels and randomly labels an element according to this distribution. The measure is then defined as how often a randomly selected element would be incorrectly labeled using this method.

The information gain metric is based on information entropy defined in information theory:

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i \tag{5.28}$$

The information gain is then the entropy of the parent minus the sum of entropy of the children:

$$IG(T, a) = H(T) - H(T|a) \tag{5.29}$$

The variance reduction is especially useful where the predictor is a continuous variable, such as in a regression tree. The variance reduction measure of a node N in the tree is defined as the overall reduction of the variance of the target variable x due to the split at that node [Quinlan, 1986]:

$$\begin{aligned} I_V(N) =& \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 - \\ & \left(\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2}(x_i - x_j)^2 + \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2}(x_i - x_j)^2 \right) \end{aligned} \tag{5.30}$$

S represents the set of presplit sample nodes, S_t the set of sample nodes for which the test evaluates true and S_f the set of sample nodes where the split test evaluates false.

The pruning phase of the algorithm works bottom-up. The idea of Algorithm 2 is to remove sections of the tree that provide only little gain to the prediction accuracy of the model. Thus it reduces overfitting. The evaluation criteria mentioned in line 3 of Algorithm 2 is most often the estimated error by testing the model with an evaluation dataset.

As with all machine learning techniques, Classification and Regression trees comply with all requirements. The advantage of the method, especially in the context of data-intensive systems is, that it performs very well with large datasets. While the problem of learning an optimal decision tree is NP-complete, all commonly used tree growing algorithms are based on greedy heuristics which produce local-optimal results. Another characteristic of the algorithm is that

Algorithm 2 Pruning phase of the top-down induction of decision trees algorithm [Breiman et al., 1984].

```
1: function PRUNE_TREE(T)
2:     repeat
3:         Select a node N in T such that pruning maximally improves the evaluation criteria
4:         if t ≠ ∅ then T = prune(T, t)
5:     until t = ∅ return T
```

it is very robust to measurement errors in the data, which are part of all real-world measured datasets. Also CART models are very easy to understand and interpret, as they can be visualized in their actual tree form.

Support Vector Machines

The idea of Support vector machines (SVM) [Hearst et al., 1998] was first introduced by Vladimir Vapnik in 1963. It is a machine learning technique used both for classification problems as well as regression analysis. The principal idea of Support vector machines is to learn a classifier based on a training dataset, where each example is tagged by one of two categories, and categorize new examples into one of the two categories. The SVM method addresses one specific issue: Common approaches were trying to minimize the expected empirical loss on the training data, however, SVMs attempt to minimize expected generalization loss [Hearst et al., 1998]. It is unknown where the future examples will fall, but under the probabilistic assumption, it can be assumed that they are drawn from the same distribution as previously seen examples. This is done by creating a maximum margin separator. The separator is simply defined as a line furthest away from the examples in the training dataset. An example for a maximum margin separator is shown in Figure 5.6. The margin is the width of the area between the two dashed lines whereas the separator is in the middle, at the same distance to the two furthest points of each category. The support vectors in this example are the vectors marked with circles. The advantage of SVMs is that such separators can not only be created for two-dimensional data spaces but they can also be found for finite dimensional spaces. The separator is then called a hyperplane. Most often it is even advantageous to transform a problem into a multi-dimensional space as a the original input data is not linearly separable, but it is easily separable in a higher dimensional space. This

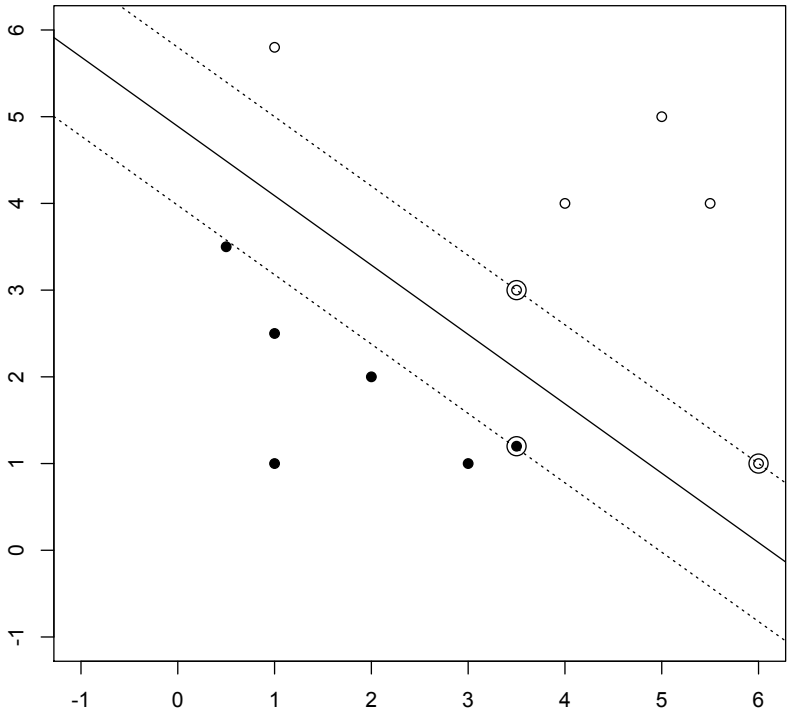

Figure 5.6: Example of a maximum margin separator.

is the so-called kernel trick.

The creation of a maximum margin separator for a linear separation works as follows: With a training dataset of n points defined as:

$$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \tag{5.31}$$

y_i are set to 1 or -1, indicating the category the vector belongs to. The objective is to find a maximum margin separator (hyperplane) that divides the group of vectors with $y_i = 1$ from the group of vectors with $y_i = -1$. This is done while maximizing the distance between the hyperplane and the nearest vector $\vec{x}_i$ from each category. A hyperplane is defined as a set of vectors $\vec{x}_i$ following

$$\vec{w} \cdot \vec{x} + b = 0 \tag{5.32}$$

$\vec{w}$ is the normal vector to the hyperplane and the parameter $\frac{b}{||\vec{w}||}$ specifies the offset of the hyperplane along the normal vector $\vec{w}$. The next step is to find the two hyperplanes which define the margin between the two categories. These can be described as:

$$\vec{w} \cdot \vec{x} + b = 1 \qquad \vec{w} \cdot \vec{x} + b = -1 \tag{5.33}$$

The distance between the two hyperplanes is then $\frac{2}{||\vec{w}||}$. In order to find a maximum margin separator the distance between the two planes has to be maximized which means minimizing $||\vec{w}||$. Off course, the minimization has to be done while the examples still lie on the correct side of the margin, which is defined by this constraint:

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1, \qquad \text{for all } 1 \leq i \leq n \tag{5.34}$$

Thus the separator is expressed by a simple optimization problem to minimize $||\vec{w}||$ subject to $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$, for $i = 1, \ldots, n$. As a consequence, the maximum margin hyperplanes are determined by the vectors $\vec{x}_i$ which lie nearest to it. These vectors are called the support vectors.

However, in practice it is rarely the case that the examples are linearly separable. Figure 5.7 shows a training dataset in a two dimensional space which is clearly not linearly separable. When the data is transformed into a three dimensional space, thus mapping each input vector $\vec{x}$ to a new vector $F(\vec{x})$ the data becomes easily separable by a hyperplane, as shown in Figure 5.8. This method is called the *kernel trick* and was originally proposed by Aizerman et al. [1964]. The phenomenon states that if data is mapped into a space of sufficiently high dimension it will generally become linearly separable. However, the transformation into a higher-dimension feature space increases the generalization error as well. The details of said kernel functions are well-described in literature, and commonly used kernels include polynomial, gaussian radial basis functions, and hyperbolic tangents.

Over the years adaptions and extensions to the SVM method have been introduced. Especially relevant to this thesis are Multiclass Support Vector Machines [Chih-Wei Hsu and Chih-Jen Lin, 2002]. These methods allow to use SVMs to classify multiple classes, instead of only two. Earlier implementations with the general SVM approach only allowed to decompose such mutliclass problems into several binary classification problems. Support Vector Regression Ma-

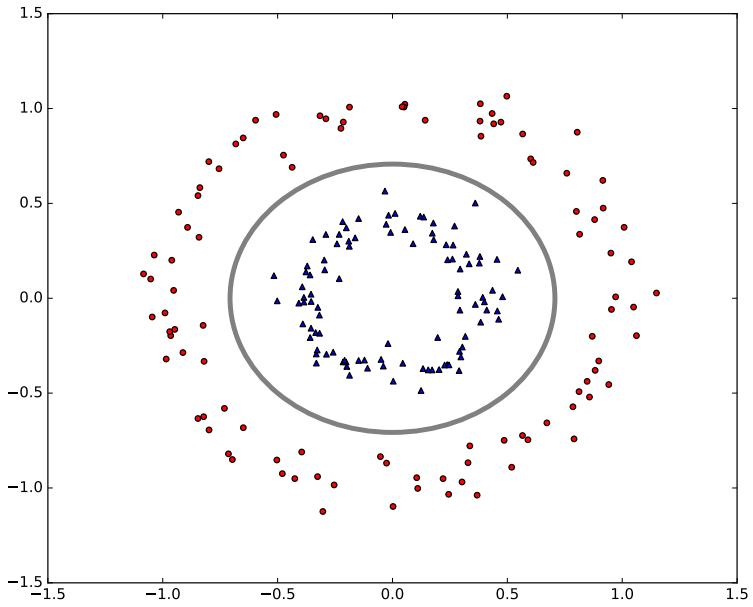

Figure 5.7: Non-linearly separable training dataset in two-dimensional space.

chines (SVR) by Smola and Vapnik [1997] do not use hyperplanes to separate groups of examples but actually try to fit a linear or non-linear regression model.

Similar to Neural Networks, SVMs have the problem that they are essentially black boxes and it is very hard to understand the internals, especially when the data is mapped to multiple dimensions. In contrast to Neural Networks though, there are usually less parameters to adapt in SVMs which makes it easier to generate an accurate model. They are also rather memory intensive and do not cope well with very big datasets.

Neural Networks

Artificial Neural Networks are a group of models which are inspired by biological neural networks, such as the human central nervous system. They are used to estimate unknown functions that depend on a large number of inputs. Typical use-cases include handwriting recognition, speech recognition, and computer vision. The theory behind neural networks was already established half a century ago by Rosenblatt [1958]. Since then the field went through many

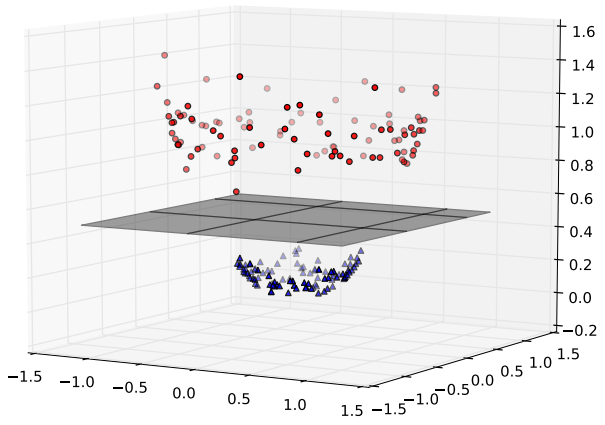

Figure 5.8: Data of Figure 5.7 mapped into a three-dimensional space with a separating hyperplane.

improvements, such as the backpropagation algorithm by Werbos [1990], and recurrent neural networks by Pineda [1987]. With the advancement of deep learning [Deng, 2014] the interest in neural networks has reached a new high.

A neural network is generally structured into nodes (or neurons) which are connected by direct links. A simple mathematical model of a neuron is shown in Figure 5.9. The link between two units i and j thus serves the purpose to propagate the activation a_i from i to j. The network also defines a weight $w_{i,j}$ for each link, which specifies the strength of the connection between the two nodes. Each node also has a dummy input $a_0 = 1$ with a weight $w_{0,j}$. When used, each neuron j computes a weighted sum of its inputs, which is called the input function:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{5.35}$$

Afterwards it applies its activation function g to calculate the output a_j:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \tag{5.36}$$

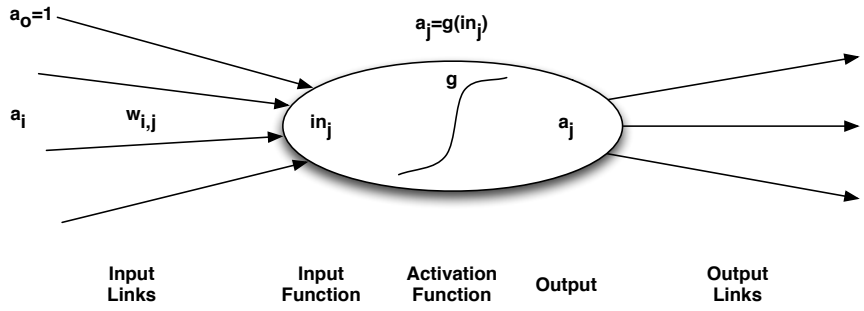

Figure 5.9: Simple model of a neuron.

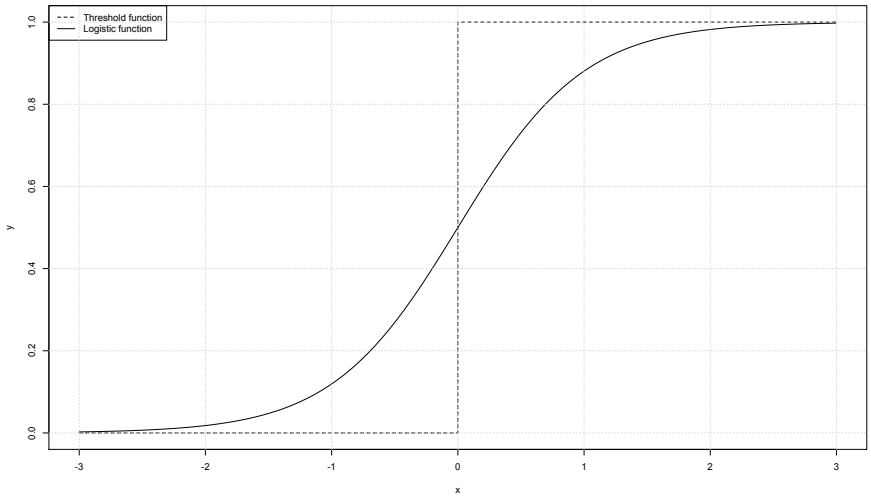

Figure 5.10: Threshold and logistic activation functions for neurons.

The function g is usually one of two types: a hard threshold, in which case the node is called a perceptron, or alternatively a logistic function. Both functions are displayed in Figure 5.10. There are two paradigms in connecting the neurons together to a network. A feed-forward network connects neurons only in one direction, thus it forms a direct acyclic graph. There are no loops in a feed-forward network and the network has no states other than the weights themselves. In a recurrent network the outputs of a neuron can be fed back into its own inputs. Thus the response of a network depends on the state of the network determined by previous inputs. This allows recurrent networks to model more complicated processes, but it also makes them much harder to understand.

Algorithm 3 Back propagation learning algorithm [Russell et al., 2003].

1: **function** BACK_PROPAGATION_LEARNING($examples, network$)
2: **repeat**
3: **for** each $w_{i,j}$ in $network$ **do**
4: $w_{i,j} \leftarrow$ a small random number
5: **for** each example(x, y) in $examples$ **do**
6: *// Propagate the inputs forward to compute the outputs*
7: **for** each node i in the input layer **do**
8: $a_i \leftarrow x_i$
9: **for** $l = 2$ to L **do**
10: **for** each node j in layer l **do**
11: $in_j \leftarrow \sum_i w_{i,j} a_i$
12: $a_j \leftarrow g(in_j)$
13: *// Propagate deltas backward from output layer to input layer*
14: **for** each node j in the output layer **do**
15: $\Delta[j] \leftarrow g'(in_j) \times (y_i - a_j)$
16: **for** $l = L - 1$ to 1 **do**
17: **for** each node i in layer l **do**
18: $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
19: *// Update every weight in network using deltas*
20: **for** each $w_{i,j}$ in $network$ **do**
21: $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
22: **until** the stopping criterion is satisfied **return** $network$

Feed-forward neural networks are typically organized in layers. Each layer only receives input from the directly preceding layer. The network can consist of multiple hidden layers, with an input layer and output layer at the beginning and the end of the network.

The general principle of learning in neural networks is to test the model based on a cost function and update the weights of the network while minimizing this cost function. Most problems will use a task-specific cost function to do this. However, this learning phase gets more complicated when the neural network has multiple layers. While the error is easily specifiable at the output layer, it is hard to specify the error at a specific hidden layer, as it is unclear what the value is supposed to be at this point. This is where the back-propagation algorithm [Werbos, 1990] is used. The idea is to back-propagate the error from the output layer to the specific hidden layer. The algorithm is shown in Algorithm 3. The weight update rule is shown in line 21. The

general idea is that a hidden node j is responsible for some fraction of the error $\Delta[j]$. Therefore the $\Delta[j]$ values are generated by the strength of the connection between the hidden node and the output node. The error is backpropagated to provide the $\Delta[j]$ value for the hidden layer (Line 18).

The algorithm can be summarized into two phases. First generate the Δ values for the output units (the error of the output node). Then, starting with the output layer until the earliest hidden layer: Propagate the Δ values to the previous layer and update the weights between the two layers.

However, training the weights of a fixed network is only one part of the learning phase of neural networks. The second part is choosing the right network structure as in number of nodes and layers. Most approaches boil down to try different structures, train them, and then choose the best one. Several algorithms have been proposed in optimizing this phase, such as [Denker and Solla, 1989] and [Mezard and Nadal, 1989].

One of the key-criticisms of neural networks are their complexity and visualization. Once trained, it is very difficult to understand why a neural network performs a certain way, as the internal processes are very scattered and not easily understandable for a human, especially when compared to the CART algorithms. Another point of critique is that neural networks require vast amounts of memory and computation power, both in training and operation. However, the development of highly optimized general purpose GPUs gave the domain a new high in recent years, as very complex neural networks can now be trained in a fraction of the time.

5.6 Evaluation

The evaluation of the different modeling approaches is done by training and evaluation the storage component models with historic workloads from Rucio. The system analysis in section 4.4 shows that a multitude of protocols is used, even for a single storage system and that many different storage technologies are deployed throughout the data-intensive system. As the component model has to accurately represent all of these technologies, different workloads from different storage systems have to be tested. Thus, a large and small tape-robot based system and a large,

medium, and small disk-pool based storage system is selected. For each system, a two week workload is fetched from the archive and split into a one week training and one week evaluation workload. Only *local* file operations are used to train and test the model, to minimize eventual effects of the networking systems. The impact of *remote* file operations on the storage system should be minimal as they only represent a very small fraction of the workload.

For each modeling technique and for each storage system a model is trained to predict the transfer rate of file operations:

$$transferrate = \frac{t_{validate} - t_{transfer}}{size} \tag{5.37}$$

The transferrate is used as it is normalized based on the transfer size, however, the transfer response time, as required by requirement 1, can easily be calculated once the transferrate is estimated.

As an error metric for the transferrate estimator the relative error is used.

$$\delta_{transferrate} = \left| \frac{transferrate - \hat{transferrate}}{transferrate} \right| \tag{5.38}$$

Linear model

To establish a baseline and to see what to expect from the models a simple linear regression model is evaluated. This model is based on the concept of Anderson [2001], which essentially correlates the number of parallel operations with the transfer rate of a storage system. To achieve this, an additional value, corresponding to the amount of parallel operations at a given time in the workload, has to be calculated and added to the workload. To some extend this is similar to the $TimeDiff_i$ value in section 5.4, which represents the temporal burstiness of the workload. Thus for a given operation i the number of $concurrent\_operations$ is defined as:

$$concurrent\_operations(i) = \sum_{j=1}^{n} concurrent(i,j) \tag{5.39}$$

with $concurrent(x,y)$ defined as:

$$concurrent(x,y) = \begin{cases} t_{transfer}(x) \leq t_{validate}(y) \leq t_{validate}(x) & 1 \\ t_{transfer}(x) \leq t_{transfer}(y) \leq t_{validate}(x) & 1 \\ \text{else} & 0 \end{cases} \tag{5.40}$$

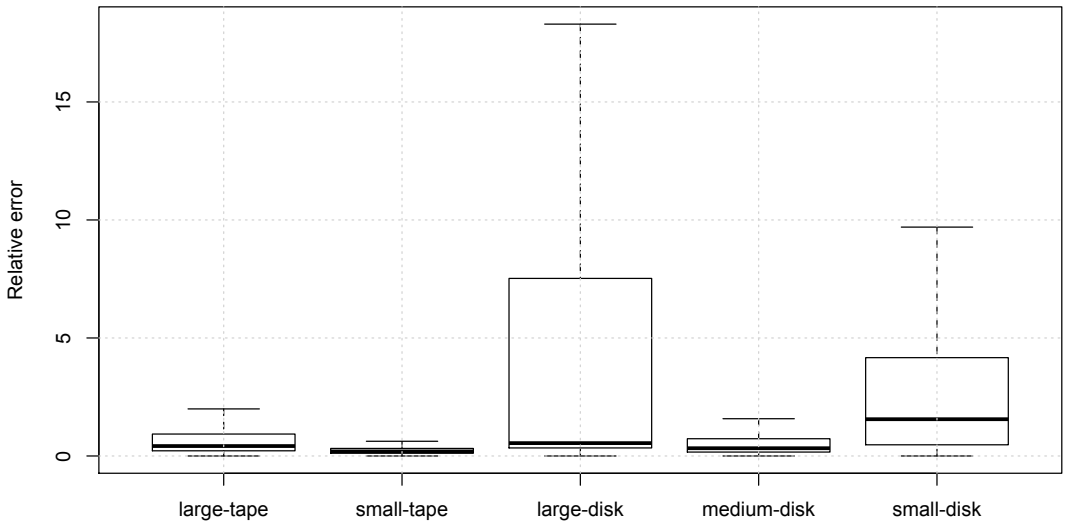

Figure 5.11: Evaluation of the linear model).

	large-tape	small-tape	large-disk	medium-disk	small-disk
$\tilde{\delta_{transferrate}}$	0.42	0.19	0.54	0.32	1.55
$\sigma(\delta_{transferrate})$	8.17	1.55	1198	28.38	10.36

Table 5.1: Statistics of the linear model evaluation.

Figure 5.11 shows a boxplot of the relative error of the linear model estimator and Table 5.1 shows the numerical statistical values. While for most storage systems the median relative error is relative small, ranging from 0.19 to 0.54, the median error of the small-disk model is much higher (1.55). The statistical spread of the error of the estimator is also of interest, as it defines how stable the model is in predicting the values. For the tape and medium disk models the standard deviation is not too high (between 1.55 and 28.38) however, for the large-disk and small-disk model the standard deviation goes up to 1198. In general, a median error of 19% is quite acceptable, as even single hard disk drive models observe errors of around 5-10%, thus an relative error of 19% in a much more complex system is quite acceptable. However, the

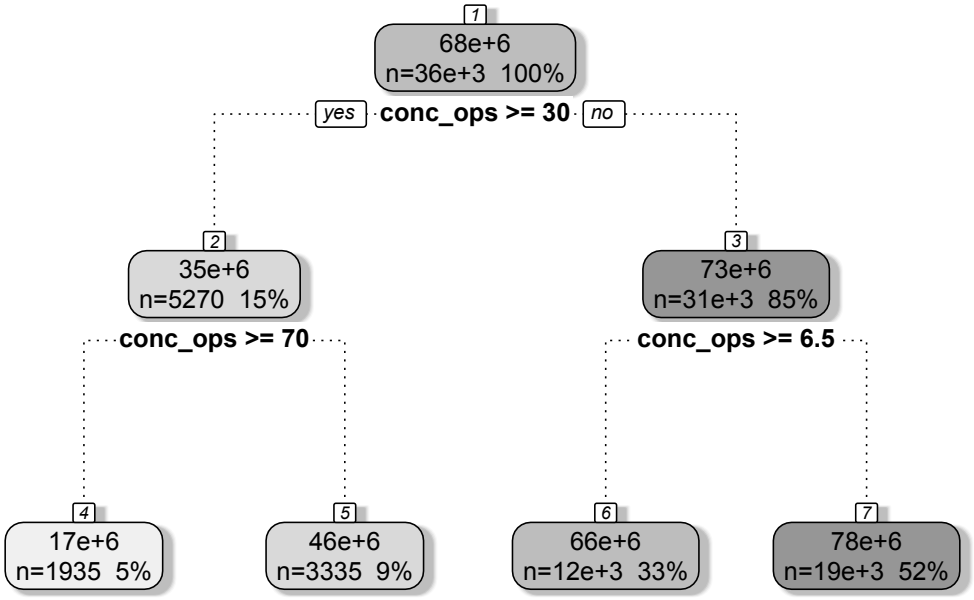

Figure 5.12: CART model of the small-tape storage system.

spread of the relative error, for some storage systems, is just too high for a usable model, with predictions being multiple factors off. The linear model thus is not generally well applicable to the transferrate prediction of storage systems in data-intensive systems, but it sets a baseline for comparison for other models.

Classification and Regression Tree model

The Classification and Regression tree models are trained with the training workload and allowing the pruning phase of the algorithm to decide which variables of the workload are most significant for the prediction. For all of the five models the algorithm only picked the filesize, protocol and concurrent_operations fields, which would also be the naive expectation for relevant columns. Figure 5.12 shows the pruned classification and regression tree model for the small-tape workload. Here the operations are only predicted based on the concurrent_operations field with 52% of the operations falling into one category. The evaluation of all workloads are detailed in Figure 5.13 and Table 5.2. While the median relative error is consistently smaller, the biggest improvement is the stability of the prediction expressed by a much smaller standard

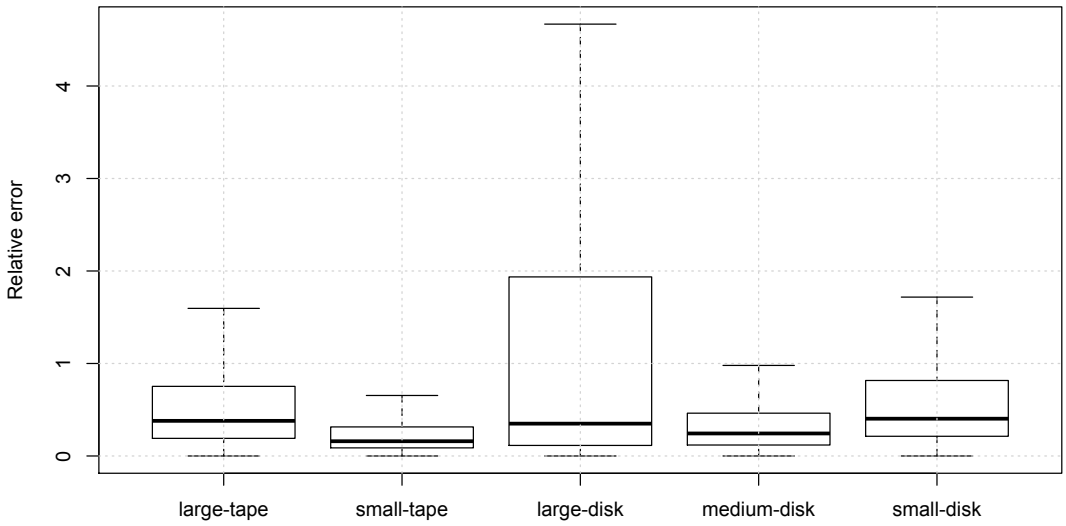

Figure 5.13: Relative error of the CART model.

	large-tape	small-tape	large-disk	medium-disk	small-disk
$\tilde{\delta_{transferrate}}$	0.37	0.15	0.34	0.24	0.40
$\sigma(\delta_{transferrate})$	3.01	0.78	184	12.1	3.12

Table 5.2: Statistics of the CART model evaluation.

deviation of the relative error. Only the prediction of the large-disk workload still shows a very skewed distribution of the relative errors with very strong outliers.

Support Vector Machines

The Support Vector Machine models are also trained with the training workload and evaluated with the evaluation workload. As the predicted variable is a vector, instead of a class, the trained SVM model is a regression machine, instead of a multiclass SVM. The model is trained using an ϵ Support Vector regression algorithm [Chang and Lin, 2001], as it showed slightly better results than the ν Support Vector regression algorithm. As a kernel function the training-phase

is parameterized with a gaussian radial basis function.

While the model training works well and in reasonable amount of time for the small-tape, small-disk and large-tape workloads, the training does not finish with the medium-disk and large-disk workloads due to the size of the datasets. While this problem could most likely be tackled by using GPU-optimized SVM training, the lack of this specialized and expensive hardware requires a different solution for the problem. While experimenting with different workload sizes, it showed that a workload larger than 100.000 rows pushes the training phase to over one hour. Above this threshold, the time it takes to generate the model increases almost exponentially. However, the large-disk dataset consists of over 1 million rows. The idea is to reduce the dataset to a smaller dataset by the means of sampling. However, the loss of information due to the sampling should not decrease the quality of the model. To achieve this, the method of stratified sampling [Neyman, 1934] is used to decrease the dataset size while preserving the representativeness of the sample to the workload. Instead of sampling the full dataset randomly, the dataset is divided into subpopulations (stratum) which are then sampled randomly or systematically. This strategy ensures that at least some samples of operations defined by a specific characteristic are represented in the sample. The optimal allocation strategy is used to define the size of each stratum, thus the cardinality of each stratum is proportional to the standard deviation of the distribution of the variable. As the defining stratum characteristic the $concurrent\_operations$ variable is used, thus there is one stratum for each value of the variable. To evaluate the loss of model quality when using sampling techniques, a support vector machine is trained for the full and different sampled sizes of the large-tape dataset. For comparison also simple random samples (without repetition) are created, trained, and evaluated. Figure 5.14 shows the result of this investigation. Using the stratum sampling technique, a sample of 80%, 60%, 40%, 20%, and 10% of the original dataset size is created, the model trained and then evaluated against an evaluation dataset. The results show that there is no loss in prediction quality, except for the 10% sample, which has a very minimal loss. As a comparison, a random sample with 20% and 10% of the original dataset size is created, trained, and evaluated as well, which shows a significantly worse evaluation result. Thus, the usage of the stratum sampling technique should be valid to downsize the very large datasets for training.

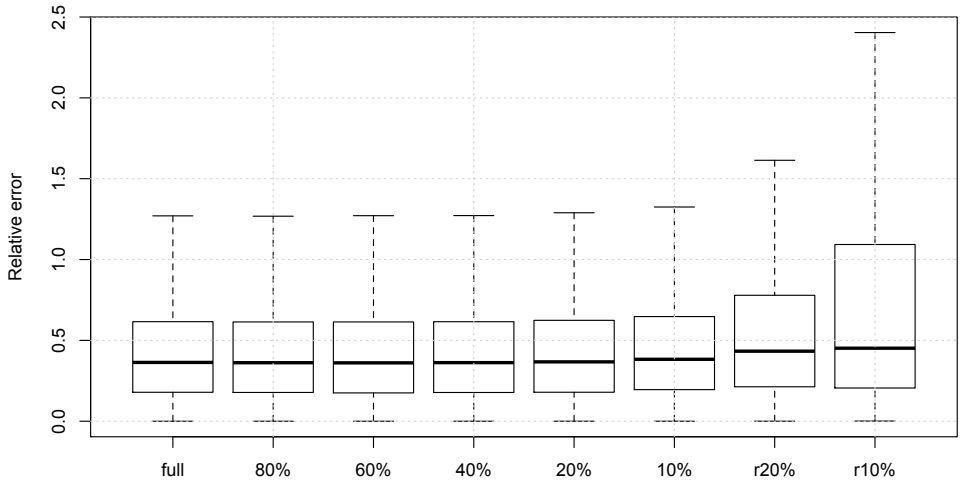

Figure 5.14: Impact of stratum (80%-10%) and random (r20%-r10%) sampling of the workload on the quality of the trained model.

	large-tape	small-tape	large-disk	medium-disk	small-disk
$\tilde{\delta_{transferrate}}$	0.36	0.13	0.15	0.20	0.52
$\sigma(\delta_{transferrate})$	2.27	0.6	70.13	3.20	3.69

Table 5.3: Statistics of the SVM model evaluation.

The boxplot of the relative error of the trained models is shown in Figure 5.15. The model for the medium-disk and large-disk storage is trained based on a sampled dataset to 10% of its original size. Table 5.3 shows the median and standard deviation of the trained models. While the median relative error of the SVM models only slightly decreased in comparison with the CART model, the standard deviation for the large-disk and small-disk models decreased significantly. Thus, the technique does perform significantly better than the CART model.

Neural Networks

The Neural Networks methodology also requires the large workloads to be sampled for the training phase to finish in a reasonable amount of time. Hence the model for the medium-disk

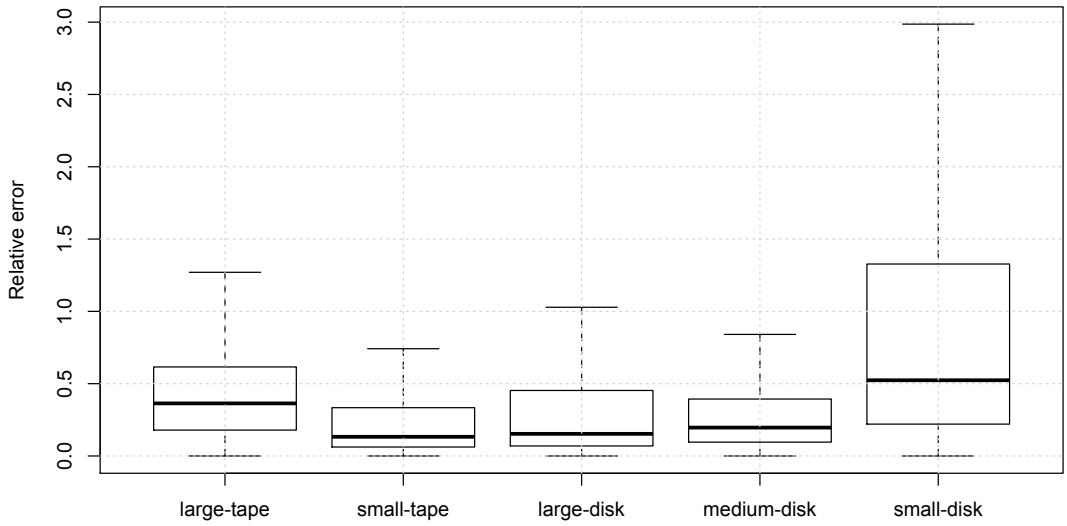

Figure 5.15: Relative error of the SVM model.

and large-disk storage are trained with samples of 10% of their original size.

In comparison with the support vector machines, which do not allow a lot of parameters to be tuned, the neural network approach requires much more fine tuning. There is usually not a very clear strategy of what parameters work best for a certain problem set, so experimentation is required. There is however certain recommendations stemming from literature. For the storage models a single-hidden-layer neural network is used as literature shows that multiple hidden layers do not result in a significant advantage in most cases [Bengio and LeCun, 2007]. The training is done by re-sampling the training workload in multiple iterations and fitting the model to the remainder of the dataset in each iterations. This workflow converges the model to an optimal solution in a number of operations (1000 as an upper bound).

Figure 5.16 shows a boxplot of the relative error of the Neural Network model for the different storage systems. Table 5.4 shows the median relative error as well as the standard deviation of the relative error. In general the model performs very similar to the Support Vector machines. The model for the large-disk shows a higher standard deviation of the relative error while the

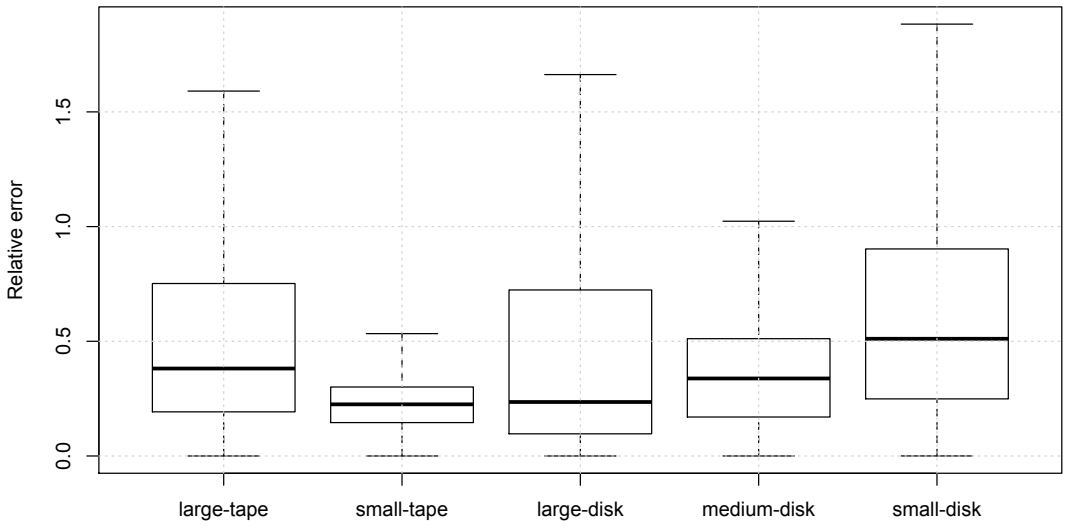

Figure 5.16: Relative error of the Neural Network model.

	large-tape	small-tape	large-disk	medium-disk	small-disk
$\tilde{\delta_{transferrate}}$	0.44	0.22	0.23	0.24	0.36
$\sigma(\delta_{transferrate})$	6.5	2.59	684	11.4	2.5

Table 5.4: Statistics of the Neural Network model evaluation.

model for the small-disk shows a lower median relative error as well as standard deviation.

5.7 Conclusion

This chapter introduced, discussed, and evaluated many different modeling techniques. Although many of the models are widely used, such as very specific analytical models for certain storage systems, they are not applicable in the domain of data-intensive systems. This has been investigated by evaluating the models with the requirements presented in section 4.2. The only acceptable techniques for modeling storage systems in a data-intensive system have been found to be in the domain of machine learning.

Three machine learning techniques, namely Classification and Regression Trees, Support Vector Machines, and Neural Networks were presented and evaluated. For the training and evaluation different storage workloads from the Rucio data-intensive system were used. The Support Vector Machines and Neural Network models showed slightly better, in some cases even significantly better, results than the Classification and Regression Tree models. In general a median relative error of around 20% was demonstrated, however, some storage systems showed a high variation. While these relative errors seem high in comparison with single disk drive models, which usually show relative errors around 1-3%, the error is quite acceptable for such complex systems. The analytical model introduced in section 5.2 showed comparable relative errors of around 15%, however, the variation was much smaller as the error does not exceed 30%. However, the analytical model is able to capture system characteristics, such as caching and internal queues, which the machine learning model, if at all, only captures implicitly. Evidently there has to be a downside for using these much more generic techniques. When comparing the results to predictions made in different domains but for similarly complex systems, such as the prediction of computation times on super computers [Smith et al., 2004], the relative errors shown in this chapter are similar, or better, than the results presented for these systems.

For the inclusion in the hybrid system model, the Support Vector Machine modeling technique is selected. However, it has to be noted that, depending on the storage system, the Neural Networks sometimes performed better than the SVM and vice versa. The challenge is that no statistical characteristics could be identified which a-priori tell which technique will perform superior. Hence, this can only be done by experimentation, which is too time-intensive for a real world data-intensive system. Rucio alone consists of more than 750 storage systems, regularly training and evaluating 1500 models would be infeasible.

CHAPTER 6

Modeling network systems

6.1 Introduction

Next to the storage systems, network systems are the second core resource of a data-intensive system. As the storage resources are rarely concentrated at a single geographical location and as the data is often accessed from different parts of the system, some means of connecting the storage resources are necessary. This is done by shared and dedicated network systems. While there are different computer networking communication standards used throughout the field, such as InfiniBand [Association, 2000], which is mostly used for super computers, Ethernet [Shoch, 1981], and Fibre Channel [Latif et al., 2002] the Internet layer and Transport layer used is predominantly TCP/IP [Fall and Stevens, 2011]. The reason for this is that the internet runs mostly on this protocol, thus it suggests itself to re-use this technology also for the dedicated networks to decrease complexity.

The objective of this chapter is to identify a modeling technique which is able to accurately model networking systems based on the TCP/IP protocol. Specifically, the time to complete or transferrate of a file transfer, independent of it being a *read* or *write* request, has to be estimated by the model. The model has to be able to realistically represent the characteristics of the TCP/IP protocol, such as fairness and congestion effects. Similar to Chapter 5 the modeling technique has to be in accordance with the requirements specified in Section 4.2. The specifics of how

these requirements translate to network systems are presented in Section 6.2.

Network simulation, similar to storage simulation, is a field with a long history. The first network models came up in parallel with the development of network communication technologies. These models were mostly driven by the needs of manufacturers of networking systems and communication services providers. In academia, network simulation became of interest a bit later, in the 1980s. The first widely used network model was the REAL [Keshav and Srinivasan, 1988] simulation package, which was later renamed to ns-1. Many more network models, with different features and use cases in mind followed later. Network models can roughly be categorized into packet-level and flow-based models. Packet-level models represent network transfers as a sequence of events, such as packet departures and arrivals. Flow-based models abstract the discrete packet stream as a single flow and simulate the flow interactions at this abstracted level.

This chapter is structured as follows. In Section 6.2 the model requirements in respect to networks of data-intensive systems are discussed. Also the network topology specification and layout is discussed in this section. Section 6.3 presents the theory behind packet-level models and discusses the most widely used models in literature. In Section 6.4 flow-based models are discussed and presented. Section 6.5 continues with the evaluation of the presented models and Section 6.6 ends the chapter with a conclusion.

6.2 Topology specification

The requirements specified in Section 4.2 apply to networking systems as follows: For requirement 1 a network transfer does not distinguish if the underlying data operation is a *read* or *write* operation, thus the response time being measured and estimated is the end-to-end response time of packet level arrivals associated to a specific file transfer.

The network systems are considered as black-boxes, as required by requirement 2, thus no internal component data is used to construct the model. However, globally available knowledge and data, such as the algorithms and mechanics of the TCP/IP protocol can be used to construct the model, as the data is publicly available. Also the *bandwidth* and *latency* of a link are considered an external/observational data point, thus it can be used during the model generation.

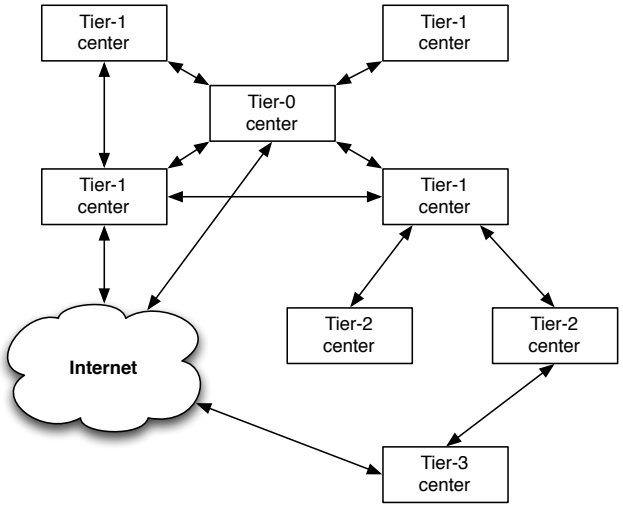

Figure 6.1: Schematic topology of the network used in Rucio.

For example, the outgoing and incoming bandwidth capacities and latencies are available for essentially all data centers in the Rucio data-intensive system.

Requirement 3 states that no controlled experiments can be conducted to measure wide performance characteristics in order to construct a more accurate model.

The network topology of a data intensive system is usually structured as follows: Each data center hosts multiple storage systems which are connected to the internal network infrastructure of the data center. The data centers are either connected to each other by dedicated network links or via the internets shared networks.

For example, the Rucio system categorizes data centers into tiers. Tier-1 data centers are connected to each other via a dedicated network, the LHC optical private network [Simeonidou, Dimitra Nejabati et al., 2004]. Each Tier-1 data center is then connected to its associated Tier-2 data centers via dedicated national networks, typically science networks, or the internet. The categorization of data centers into Tiers is not significant, what is significant though is that data centers are connected to each other via different networks, thus the network topology itself is heterogeneous. A network model thus must support heterogeneous topologies. Figure 6.1 shows an extract of the schematic topology of Rucio. In general, these dedicated resources are faster and potentially easier predictable, as they are under the control of the data-intensive system. The

outgoing and incoming internet links of the data centers are also well described, but it is unknown which routing control and what resources are used at the hops in between the endpoints. It is also unknown how these resources are utilized and if any congestion effects affect the network transfers. The reason for this is that while the network transfer of the data-intensive system are observable and measured, the transfer happening on the internet are not, thus it is unknown if and how they affect the transfers of the system routed via the internet. These effects can only be categorized as noise. The internet transfers are inevitably more difficult to predict than transfers enacted in an isolated network resource. Data centers with multiple links offer different options how to route incoming and outgoing data, this also needs to be represented in the network model.

For the sake of simplicity, the focus of the network model is on external connections and not on the network infrastructure inside the data center. This simplification is done due to two reasons: It is unlikely that the internal network topology has a significant impact on the performance of transfers, as the bottleneck is usually the data center outgoing network link. Thus, ignoring this makes the model less complex. Secondly, the internal topology of the data centers in a data intensive system is mostly unknown. This situation is quite similar to the storage systems as the operation of the data centers is not done centrally. For data center internal operations, the network performance is implicitly included in the storage model.

Any network model supporting the use-case of a data-intensive system should support the representation of the network topology in a simple graph-based notation. The topology is thus defined as an directed graph $G = (V, E)$ where the set of vertices V represents the set of data centers and the set of edges E represents the set of network links. The internet is also represented as a special vertices v_{INT} making the set of vertices $V = \{v_1, v_2, \ldots, v_n, v_{INT}\}$. Each data center has at least one link to the internet $\{v_i, v_{INT}\}$ and the data centers connected to dedicated networks have multiple links. The attributes defining each network link are constrained to the *bandwidth* and *latency*. This information is generally consistently available on a global level. Some models also benefit from additional data, such as average packet loss, but this information is not consistently available. For each combination of vertices v_i, v_j the route is defined as a list of edges $\mathbf{L}_{i,j} = \{\{v_i, v_k\}, \{v_k, v_m\}, \ldots, \{v_n, v_j\}\}$.

The general requirements to a network model thus are:

1. Accurate estimation of TCP/IP transfers and protocol effects.

2. Handling of topology representations in a loosely defined graph.

3. Not requiring more information than bandwidth and latency per link.

6.3 Packet-level models

Packet-level models were among the first network models used in communication network simulation. The idea of using discrete event simulation, which simulates every single packet being part of a TCP/IP data flow, is self-evident as the model can re-enact processes done in real network devices that also manipulate single TCP/IP packets. Instead of enacting these processes in a network device, the process time to complete for each packet is estimated inside the CPU of the computer running the simulation. This approach uses very few abstractions, thus the core model itself is very close to its real world counter parts. There exist numerous simulation models based on this principle, with the most prominent ones being ns-3 [Henderson et al., 2008], GTNetS [Riley, 2003], and SSFNet [Cowie et al., 1999]. A comprehensive overview of network models is presented by Gupta et al. [2013].

These simulators operate, in most parts, very similar. Usually elements of the simulated network, such as links, routers, switches, and computers are represented as queues. A global scheduler analyzes which elements to activate at a certain moment in the simulated timescale, activates the modeled function of the associated elements, which transforms and moves events from one queue into another, and then extends the simulated time for a calculated value. The queues do not hold the actual simulated TCP/IP packets but events associated to a specific packet, such as packet-received, or transmission-complete events. Thus a single TCP/IP packet requires multiple simulation events, which increases the simulation complexity as instead of $O(N)$, $O(j \cdot N)$ events have to be passed. Figure 6.2 shows a simple schematic topology of a network of a packet-level model including two nodes, their queues, and a simplex link in between. A pseudocode for a simple packet transmission is shown in Algorithm 4. When a transmission event, associated to a TCP/IP packet is sent the event simulator calls the *transmit_packet* method, including all the packet information. If the link is currently busy, a transmission event is queued

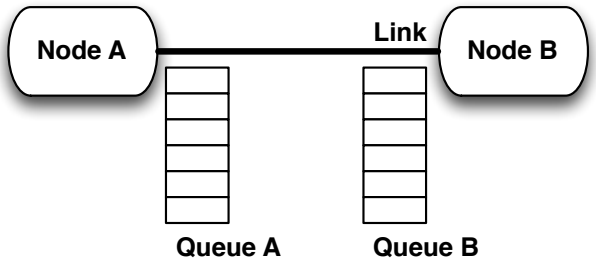

Figure 6.2: Queue architecture example used in packet-level network models.

Algorithm 4 Pseudocode for packet transmission events in a packet-level network model.

```
1:  function TRANSMIT_PACKET(packet)
2:      // Process a packet transmission event
3:      if Link is not busy then
4:          // Calculate the time to transmit
5:          transmit_time ← packet_size / link_bandwidth + propagation_delay
6:          // Schedule receive event at receiver
7:          schedule_packet_receive_event(transmit_time)
8:          // Schedule link free event
9:          schedule_link_free_event(transmit_time)
10:         Set link busy
11:     else
12:         // Link is busy
13:         // Enqueue packet for later transfer
14:         enqueue_packet()
15: function LINK_FREE_EVENT
16:     // Process a link free event
17:     Set link not busy
18:     if Queue not empty then
19:         Remove packet from Queue
20:         transmit_packet(packet)
```

in queue A and the event will be pulled by the scheduler at a later moment. If the link is not busy, the transmission time is calculated based on the packet size and the bandwidth, also the speed-of-light propagation delay is added to the transmission time (Line 5). Then a receive event with this timestamp is put into queue B (Line 7). At the same time a link free event is put into queue A, to signal that the queue is empty now (Line 9). When this link free event is polled at queue A, the link is set to non-busy and a new transmission event can be retrieved from the

queue.

This example only describes a very simple workflow and does not go into details of congestion events, and flow fairness. However, the key characteristic of packet-level models is that a simple packet can generate multiple events, in case of complex congestion issues even a very large amount of events per packet. The model includes queue transformation functions for all type of workflows and situations which always serve two functions: Moving and transforming simulation events, and the expansion of simulation time.

The network models of ns-2/ns-3, GTNetS an SSFNet are widely validated across all commonly used protocols [Henderson et al., 2008]. From an accuracy point of view they are deemed to perform extremely close to their real environments.

Due to this a lot of related work is about finding ways how to reduce the amount of events without losing accuracy due to abstraction [Fujimoto et al., 2003] or to increase the parallelization of the simulator. Simulator performance is measured in the number of packet transmissions that can be processed by a simulator per second of wallclock time (PTS). Typical performance measures on a sequential execution are around 100.000 PTS. With parallel execution with 1500 processors PTS values of up to 100 million have been demonstrated. However, these performances are also very dependent on the problem set and the interaction of the simulated network flows.

6.4 Flow-based models

While the scalability issues of flow-based network models do not matter much for most researchers studying network protocols, the simulation of networks of distributed systems is on a completely different scale, as most problem sets would just take orders of magnitude more simulation time then simulated time.

Flow-based models were first proposed by Anick et al. [1980]. Their objective is to scale much better by using higher levels of abstraction. Specifically they estimate the performance of network flows instead of network packets by having functional abstractions of the bandwidth sharing behavior of TCP flows [Massoulie and Roberts, 1999]. This estimation is done purely

based on the bandwidth and TCP congestion window size. This is intuitively much more scalable, as a single network flow can abstract hundreds of megabytes in transferred packets, which can require millions of events to be processed in a packet-level model. The general idea of flow-based models, such as proposed by Velho and Legrand [2009], is that every link $\mathcal{L}_k$ is defined by a maximum bandwidth $\mathcal{B}_k$ and every flow $\mathcal{F}_i$ has a throughput p_i. The system is thus defined by the constraint that the bandwidth capacity of each link may never be exceeded:

$$\forall \mathcal{L}_k \sum_{i|\mathcal{F}_i \text{ uses } \mathcal{L}_k} p_i \leq \mathcal{B}_k \tag{6.1}$$

The key question for a model is thus how TCP handles bandwidth fairness when more than one flows are active at a certain link. While the initial assumption is that a Max-Min fairness model [Hahne, 1991] is used, this was shown to be wrong by Dah Ming Chiu [2000]. Analytical investigations of the TCP model [Floyd and Fall, 1999] showed that

$$p_i = \frac{c}{RTT \cdot \sqrt{P}} \tag{6.2}$$

RTT is the round trip time, P the fraction of packet loss, and c a constant. Thus when assuming that the packet loss is constant among flows, the formula suggests that the bandwidth is in inverse proportion to the round trip time. The congestion mechanism also relies on the window size W, limiting the bandwidth of a flow to $\frac{W}{RTT}$ as there are always at most W pending packets in a flow. Based on this it has been shown that the TCP sharing mechanism, at equilibrium, is actually equivalent to maximizing:

$$\sum_i \frac{\sqrt{3/2}}{\mathcal{D}_i} \tan^{-1}(\sqrt{3/2}\mathcal{D}_i p_i) \tag{6.3}$$

$\mathcal{D}_i$ is the equilibrium round trip time [Low, 2003]. The issue with this formula is that it is much harder to maximize than a simple max-min model. Thus most flow based models decide to use a simpler equation, still delivering a close result at much lower computational cost.

For example, a popular flow-based model part of the SimGrid [Casanova et al., 2008] framework approximates the equation by using a RTT aware max-min sharing algorithm. The full model and the exact proof is detailed by Casanova and Marchal [2002]. There exist two types of bottlenecks, balanced ones, where the bandwidth is shared fairly, and unbalanced ones. A

bottleneck link is defined as:

$$\sum_{i|\mathcal{F}_i \text{ uses } \mathcal{L}_k} p_i = \mathcal{B}_k \tag{6.4}$$

It can be proven that the links on a route r which are balanced bottleneck have the greatest utilization $U(l)$:

$$U(l) = \frac{1}{\mathcal{B}_l} \sum_{r \in l} \frac{1}{RTT_r} \tag{6.5}$$

The idea of the algorithm [Casanova et al., 2008] is then, to calculate a fair bandwidth allocation at every checkpoint in the event simulation, inversely proportional to the flows RTTs, starting with the balanced bottleneck links.

At any given checkpoint in the discrete event simulation, the model loops over this algorithm until there are no more routes left:

1. Compute the utilization $U(l)$ of every link.

2. Identify the links with the greatest utilization and put them into the set $\mathcal{L}_1$.

3. For each route r going through one of the links in $\mathcal{L}_1$ compute the throughput p. Remove all the routes going through the links in $\mathcal{L}_1$ and lower the capacity of all links used by these routes:

$$\begin{aligned} &\mathcal{R}_1 = \{r \in \mathcal{R} | r \cap \mathcal{L}_1 \neq \emptyset\} \\ &\forall r \in \mathcal{R}_1, p_r = \min_{l \in \mathcal{L}_1 \cap r} \left\{ \frac{\frac{1}{RTT_r}}{\sum_{r' \ni l} \frac{1}{RTT'_r}} \right\} \\ &\forall l \in \mathcal{L}, \mathcal{B}'_l = \mathcal{B}_l - \sum_{r \ni l} p_r \end{aligned} \tag{6.6}$$

Other [Schwefel et al., 2003] flow-based network simulators use a different equation set to mimic the bandwidth sharing/congestion solving behavior of the TCP streams but the general concept of using these abstract interactions at a flow-level save a lot of computation efforts making the simulation much more efficient.

6.5 Evaluation

The question of which model to integrate into the hybrid simulation model for data-intensive systems is based on two key-characteristics: Accuracy and scalability. The GTNetS [Riley, 2003] model is selected as the representation for the packet-level models and the SimGrid [Casanova, 2001] model as representation for the flow-based models. Both models were selected due to their wide adaption and ease of integration. Also both models support the topology specification described in Section 6.2.

Accuracy

Packet-level simulation models have been extensively validated [Henderson et al., 2008]. Their estimations are equivalent, or extremely close, to real-world network communication systems. In most cases, the small differences can be explained by noise or effects which were just not part of the input data of the model. This does not come as a big surprise as the design of the models represent the real systems very closely. However, with flow-based network models the level of abstraction is much higher, thus an estimation error (residual) is, to some degree, expected. The question is how large it is and if it is significant enough to exclude flow-based models for data-intensive systems.

A first comparison between packet-level and flow-based network models was done by Nicol et al. [1999]. The authors used the implementation of a flow-based model and plugged it into the SSF net simulator. This allowed them to directly compare the results of SSF, in packet-level mode, with the flow-based model. They use different test topologies and experimental setups with multiple hundreds of nodes and multiple thousand flows. Their experimental results showed that the relative error of the flow-based model is in the 1% range.

A second comparison was done by Fujiwara and Casanova [2007] for the SimGrid flow-based network model. The authors created and executed multiple experiments with one-link, 5-link (Dumbbell) and multi-link random topologies. They compare the results of the SimGrid model against the packet-level model of the GTNetS simulator. Their findings were that for big flows (Larger than 10MB of data) the result is good (relative error in the 1% range) but

the accuracy decreases with smaller flows. Also in heavily congested networks, the accuracy decreased.

A second approach of these experiments with SimGrid was presented by Velho and Legrand [2009]. The researchers investigated an updated version of their model with a refinement of the Max-Min sharing approximation. They could show excellent results for flows larger than 100KB with relative errors below 0.5%. Also the model worked accurately in heavily congested network situations.

Scalability

While not specifically mentioned in the model requirements, scalability is a relevant characteristic for a model of a data-intensive system, as it has a direct impact on the runtime of the simulation. This has not been specifically required, as the runtime is highly dependent on the hardware the simulation is executed on, the size of the simulated system, and the size of the input workload. But in general terms, it should be possible to simulate a day or multi-day workload of a data-intensive system in a reasonable amount of time, thus in hours, not days.

In Liu et al. [2001] the authors study the efficiency of flow-based simulation vs. packet-level simulation. For the packet-level model they use the SSF simulator and the flow-based model is represented by two FIFO and WFQ based models. They showed a factor 4 performance increase between the flow-based model to the packet-level model.

In [Fujiwara and Casanova, 2007] the authors show that with the highly-performant GTNetS platform the simulation of a 125 sec workload of 200 flows, each transferring 100 MB on a topology with 200 nodes takes about 1500 sec on a 3.2GhZ Intel Xeon processor. Thus it takes an order of magnitude more simulation time than simulated time. A comparison based on the number of flows is shown in Figure 6.3. The plot shows a much smaller incline for the flow-based model. When repeating this experiment with an 8-core AMD Opteron processor with 2.3 GHz the numbers were very similar, an experiment with 2000 flows did not finish after six hours for the packet-level model, while the flow-based model simulation finished after 30 seconds (for a 125 sec workload). While these results are not very surprising due to the millions of events a packet-level simulator has to process, it is quite discouraging in terms of implementation in

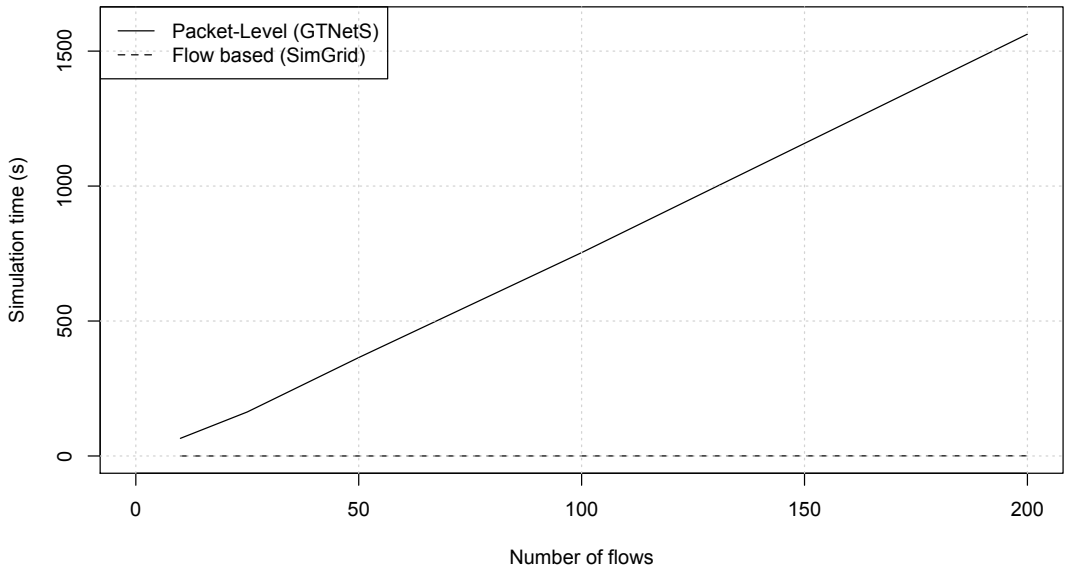

Figure 6.3: Simulation time comparison between packet-level and flow-based simulator by [Fujiwara and Casanova, 2007].

the hybrid data-intensive model. In Rucio, an average days workload includes about a million file transfers with a Petabyte of aggregated volume. Thus, this would require the simulation of a million flows, not including multi-session protocols which would instantiate multiple flows per file, resulting in about 700 billion packets (with an average window size of 1500 Bytes).

6.6 Conclusion

In this chapter network communication models were introduced, discussed and evaluated. At first the network topology specification in respect to data-intensive systems is introduced. Specifically, the requirements laid out in Section 4.2 were examined and adapted to network communication models. This also resulted in a specification for network topologies which any considered network model must support.

Then the two most used paradigms of network communication models were introduced and discussed. Namely packet-level models, which use TCP packets as the unit in the discrete event

simulation and flow-based models which abstract multiple packets into one TCP flow. As the packet-level models work at a very detailed level close to their real-world counterparts, their accuracy is superior. However, it has been shown that the flow-based models deliver very similar accuracy, especially with data flows over a certain size, which is essentially always the case in data-intensive systems. However, the scalability of packet-level models is largely inferior compared to flow-based models. At the size and complexity needed for typical data-intensive systems, a packet-level based simulation would simply not finish in a reasonable amount of time, at least without an impracticable amount of hardware.

Due to this scalability issues and the equivalency in accuracy, flow-based models, specifically the implementation in the SimGrid [Casanova, 2001] framework is selected for inclusion in the hybrid system model for data-intensive systems.

CHAPTER 7

Modeling data integrity validation

7.1 Introduction

Data validation is a common process enacted in most data processing systems and data-intensive systems are no exception. All read and written data has to be validated for integrity by the system. While most underlying network protocols, such as TCP do provide data integrity guarantees failures can still happen during other processes in the workflow, such as failures during reading at the source, and failures during writing in the storage system at the destination. The data-intensive system has to provide functionality to validate the data at the end of the workflow. The Rucio data-intensive system provides four workflows (see Section 4.4) for reading, writing, and transferring data, all of which enact data integrity validation. While writing and transferring do the data validation implicitly via the storage and transferring protocols, which is implicitly included in the storage model, both read workflows enact explicit data validation. The objective of this chapter is to establish a modeling technique which is able to estimate this data integrity validation process.

The remainder of this chapter is structured as follows. In Section 7.2 the general concepts of data integrity errors and their validation methods are presented. This section focuses on checksum algorithms and their performance. Section 7.3 continues by presenting different modeling techniques to estimate the time to completion of checksum algorithms. All techniques are eval-

uated with historic validation data from the ATLAS data grid. The chapter then concludes in Section 7.4.

7.2 Data integrity validation

The purpose of data integrity validation is to detect and recover from corruption errors of the data, so that they are not propagated to the user [Bairavasundaram et al., 2008]. Typical reasons for data corruption are errors in the storage system, such as disk failures, or errors during transferring the data over the network. The general idea of data integrity validation is the same in all data processing systems: Ensure that the data is recorded correctly and upon retrieval ensure that it is the same as when it was originally recorded. In the ATLAS data grid, as in most data-intensive systems, this is done by calculating and storing a checksum when the data gets created. For all subsequent transfer and read operations, the checksum is calculated again for the newly written data and compared against the original checksum. When detecting checksum errors, the system can further investigate if just the newly written data is corrupted, and thus can be re-read, or if there is a data corruption problem at the source, with data which once already passed validation.

There are different types of checksum generating functions, however, the most commonly used ones are hash functions. A hash function is a mathematical function which maps data of arbitrary size to data of fixed size. Hash functions can have different properties, but in the context of data integrity validation, the hash function should support: *Determinism*, thus a given input value always results in the same hash value; *Uniformity*, thus every value in the output range should be generated with the same probability. This is required to decrease the amount of hash collisions. *Non-Continuity*, thus two slightly different inputs should result in very different hash values. Due to their popularity mostly cryptographic hash functions are used, which also have the property of *Non-Invertibility*, thus it is very hard to reconstruct the input of the function based on the hash value.

In the ATLAS data grid two different checksum algorithms are used: MD5 by Rivest [1992] and Adler-32 by Adler [1996]. MD5 processes inputs of any length and produces fixed-length

outputs of 128 bits. This is done by breaking the input into blocks of 512 bit and executing 4 rounds of 16 operations on each block. Each operation is based on a non-linear function F. MD5 consists of four different functions F and uses a different one in each of the four rounds. Adler-32 is based on the Fletcher-32 [Fletcher, 1982] algorithm and also takes inputs of any size to produce a 32 bit output. The aim of Adler-32 is to trade reliability for speed. The algorithm generates two 16-bit checksums A and B and concatenates them to a 32-bit integer. A is obtained by summing all bytes in the stream, plus one, and B is the sum of all individual values of A. The exact algorithm is shown in Algorithm 5. The modulo value used in line 6 and

Algorithm 5 Pseudocode of the Adler-32 [Adler, 1996] algorithm.

```
1: function ADLER32(input)
2:     A ← 1
3:     B ← 0
4:     i ← 0
5:     for i < len(input) do
6:         A ← (A + input[i]) mod 65521
7:         B ← (B + A) mod 65521
       return B × 2^16 + A
```

7 is the largest prime number fitting in a 16 bit integer. The general advantage of the algorithm is that the values are aggregated continuously, thus it can be used in parallel for multiple data streams, such as when writing to tape libraries.

However, the actual used hash function is not essential in terms of modeling the performance of the function. The model should just be able to capture the performance of any kind of hash function, as required by Requirement 2 in Section 4.2.

7.3 Modeling & Evaluation

When it comes to modeling the performance of hash functions, one key characteristic is apparent: the checksum is calculated on the user's computer. This creates an additional difficulty in modeling the process, as while there is some knowledge about the environment of processes running within the data-intensive system, very little is known about the user's computer. A naive assumption would be that the performance of a hash function is mostly dominated by the file-

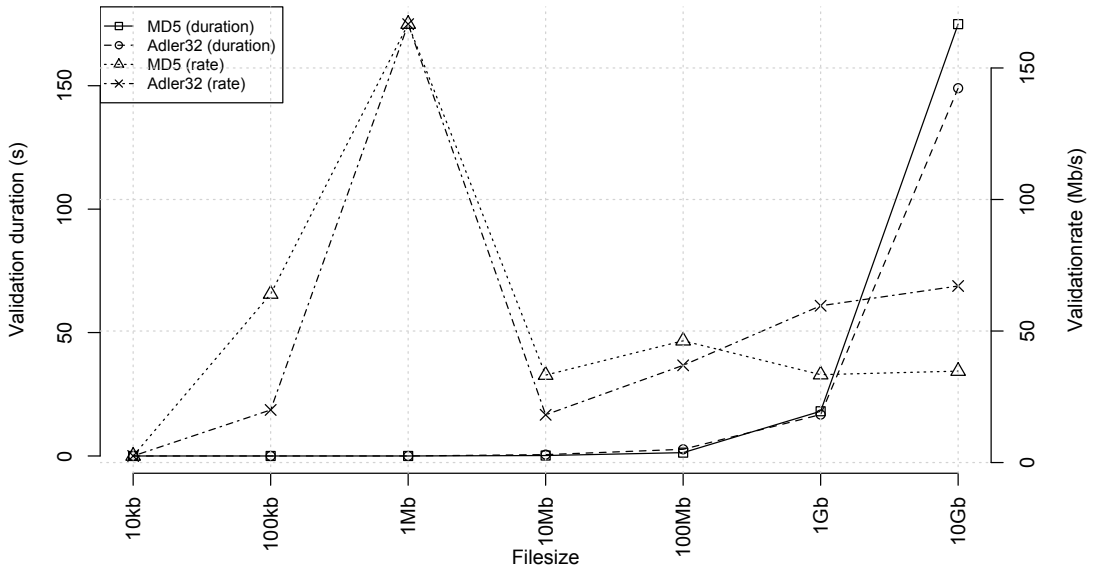

Figure 7.1: Validation duration and validation rate experiment with MD5/Adler-32.

size, the CPU, and the I/O performance of the computer. However, only the file size is known to the model. Figure 7.1 shows a plot of experimental measurements taken on a personal computer by calculating MD5 and Adler-32 checksum for randomly generated files of different sizes. The plot shows both the validation duration as well as the validation rate defined as:

$$validationrate = \frac{filesize}{t_{end} - t_{validation}} \tag{7.1}$$

Both checksum algorithms show very similar durations, with Adler-32 executing faster for large files.

Figure 7.2 shows a distribution of file sizes in the ATLAS data grid. The plot shows that a vast range of filesizes is used in the data-intensive system, thus the question is how the checksum functions perform under such a large variety of inputs.

Figure 7.3 displays the validationrate as boxplot for one month of read and write operations on the ATLAS data grid. The plot shows that the validationrate is very small for small files but asymptotical converges to a value around 190 Mb/s for files bigger than 100Mb. The bad validationrate for smaller files is most likely due to initializiation overheads of the checksum process.

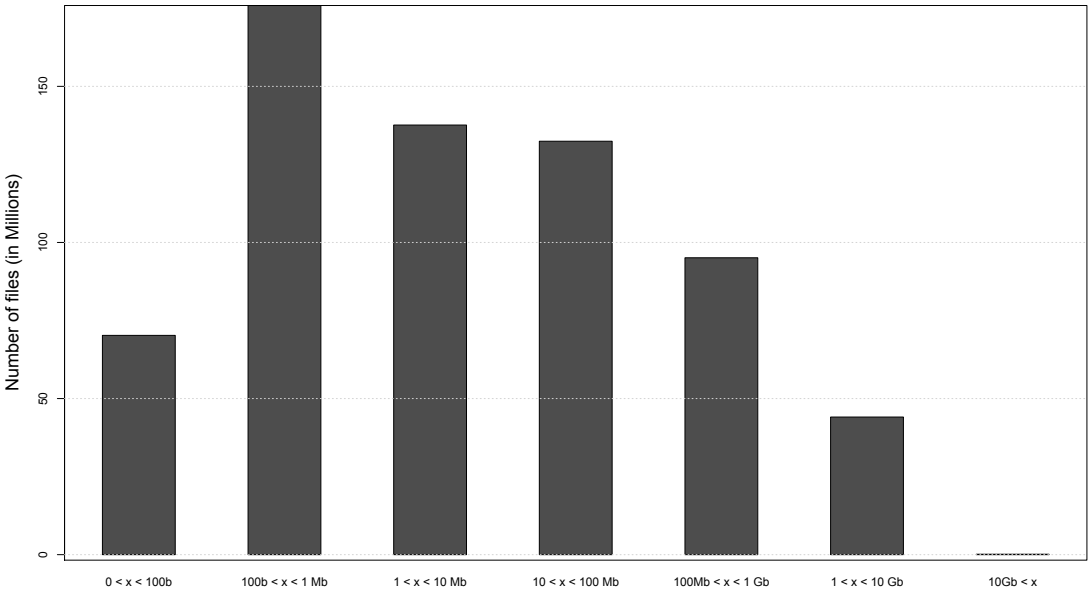

Figure 7.2: Distribution of file sizes in the ATLAS data grid.

However, the plot also shows large variances and deviations from the median. Again, this is most likely due to the fact that the checksum calculation is executed on the user's computer, thus different environmental conditions hugely influence the checksum walltime. The same effect as seen in Figure 7.1 can also be observed in this plot: The validation rate is low for small files and reaches a stable rate for larger files. The outlier for the 1Mb sized file for both methods is most likely explained due to some kind of disk caching effect.

The patterns shown in Figure 7.3 and Figure 7.1 suggest that a model should not be too complex to build, as the behavior of the validation rate is quite predictable. However, as no information is available about the execution environment, the model purely relies on historic data and the averages of the validation rate observed in these datasets. In many cases this is fine, as typically computing nodes from a computing farm are validating the data accessed at a storage element and these computing nodes are relatively homogenous. However if the data of a storage element is accessed from a much bigger set of different computers, the validation predictions are inevitably more erroneous.

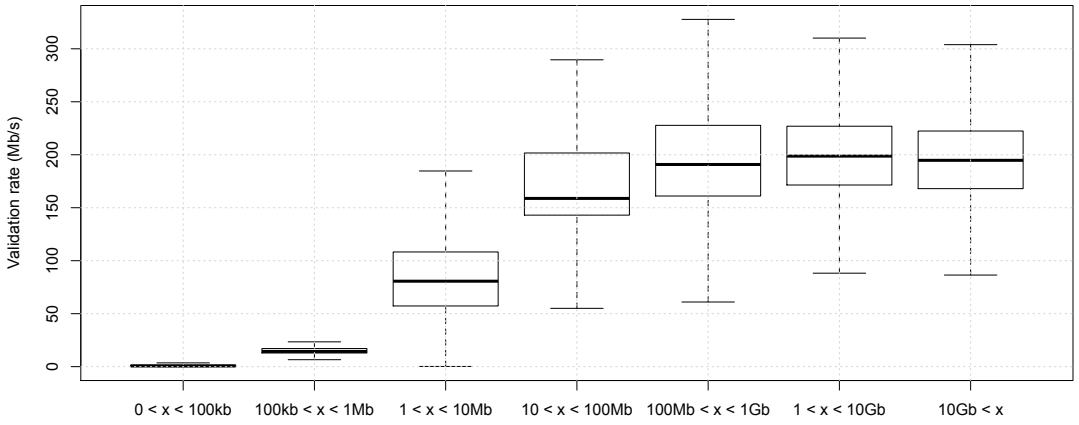

Figure 7.3: Validation rate (in Mb/s) for one month of operations in the ATLAS data grid.

The model's quality is expressed as the relative error of the $validationrate$ defined as:

$$\delta_{validationrate} = \left| \frac{validationrate - \hat{validationrate}}{validationrate} \right| \tag{7.2}$$

Figure 7.4 shows boxplots for the relative error of a linear model, a classification and regression tree, a support vector machine, and a neural network. As already expressed in Section 5.5, all machine learning models comply with the requirements set out in Section 4.2. All models are trained to predict the validation rate based on the $filesize$ of a training workload, as no other information is available about the environment the validation is executed in. The evaluation is done by predicting the validation rate of an evaluation workload. Based on the measured validation rates for different filesizes shown in Figure 7.3, the assumption that a linear model would work well holds, especially if the distribution of filesizes is not too large. In general all models perform quite similar with the neural network model performing best with a median relative error of 17%.

7.4 Conclusion

In this chapter validation models were introduced, discussed and evaluated. The chapter introduced the most commonly used cryptographic hash functions which are used for file integrity

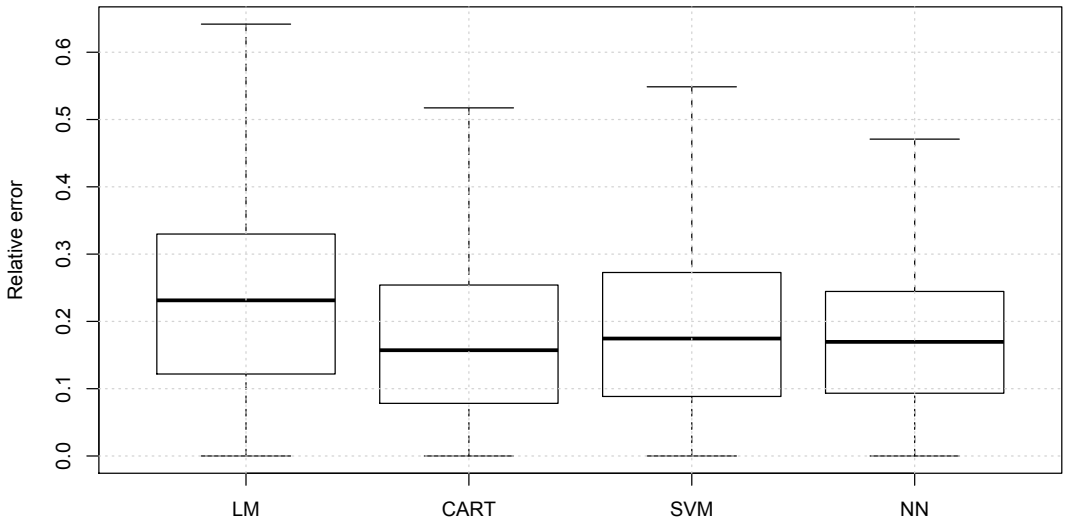

Figure 7.4: Validation of different models for the validationrate prediction.

validation. An investigation was conducted to analyze the correlation between the filesize and the validation duration. It was shown that for files larger than 100Mb the validationrate of cryptographic hash functions stays essentially constant. This was shown based on historical workloads as well as in experiments conducted on a personal computer.

A linear model as well as three machine learning techniques were evaluated to predict the validationrate. All models performed quite similar, with the neural network model performing best, thus it is selected for inclusion in the hybrid system model for data-intensive systems.

CHAPTER 8

Modeling services

8.1 Introduction

While the other components, such as storage, network, or validation are essentially similar in any data-intensive system, the service workflows emphasize its peculiarities. This poses a challenge in creating an accurate, but still abstract, model which is able to predict the performance of any service workflow. Next to the performance specific prediction the component model must also include a functional part in order to correctly represent the decision process of the real-world counterpart. This chapter focuses on the performance predictive part of the model, the functional part will be covered in Chapter 9.

The objective of this chapter is thus to establish one, or multiple, modeling techniques which are able to build an accurate component model for the different service components used in data-intensive systems. The model predicts the components response time and should be abstract enough to be able to cover unspecified service workflows, just based on the observational data specified in Section 4.2.

For the ATLAS data grid four different service components were identified in Section 4.4. The *metadata service* is used to query the system for a list of files based on a set of metadata. The *replica management service* keeps track of the physical replicas in the data-intensive system and resolves a set of files to the addresses of the physical replicas. The *replica selection service*

directs the client which replicas to use, based on the current system state, as well as other inputs controlling the selection algorithm. The *transfer service* is a scheduling component for site to site transfer workflows. It is a queuing system used to protect the system from overload. While the first three components are dominated by the performance of the database, the transfer service is a classical queuing system.

The remainder of this chapter is structured as follows. In Section 8.2 the related work in modeling the performance for service components is presented and discussed. Section 8.3 continues with the introduction, creation, and evaluation of a predictive model for service components which purely relies on historic observations. In Section 8.4 a predictive model specifically for queue based service components is presented and evaluated. The chapter then finishes in Section 8.5 with a conclusion.

8.2 Related work

There is no specific related work on modeling service components. For a majority of service components the response time is mostly dominated by the performance and complexity of a relational database management system (DBMS).

There is some related work in the domain of creating performance models for the execution time of queries for database management systems, such as SQL queries for relational database systems. The internal scheduler of a DBMS benefits from performance models [Wentao Wu et al., 2013] to optimize the execution time of incoming queries. The related work can roughly be divided into two approaches, one based on analytical modeling and the other on machine learning. This section continues with discussing one popular article for each of the two approaches. For the analytical model the work of Wu et al. [2013] and for the machine learning approach the work of Gupta et al. [2008] is presented.

In Wu et al. [2013] the authors follow an analytical approach to make an execution time prediction for an incoming query under load. The query is not predicted in isolation but under load of the database, which is the most common case in a real-world scenario where the database executes multiple queries concurrently. The approach consists of two steps: First, for each

incoming query the optimizer decomposes the query and estimates the costs of I/O and CPU, for each pipeline of the query. These abstract costs are then converted to time by using some system-specific parameters. Second, the acquired execution times for each pipeline are then fed into two queueing networks, one for I/O and one for CPU which represents the concurrency of all active queries in the system. The residence time in the queue essentially represents the execution time of each pipeline. A set of equations is used to transform the queuing times if multiple servers are available to execute the queries or if data is partially available in a buffer. The model is evaluated for a large set of workloads and offers relative errors of the execution time prediction between 20% and 80%. This analytical approach covers very well the problem of infinite number of unknown queries. However, it requires a very large set of parameters specific to the DBMS hardware and configuration.

In Gupta et al. [2008] the authors use a machine learning approach to minimize the amount of information needed about the DBMS. The general idea is that each incoming query is decomposed by the query optimizer into several pipelines and their associated cost. Both values are input to the prediction model. Whenever a new query is executed a load monitor also reads the current system state and transforms it into a load vector which is also fed into the prediction model. The prediction model then estimates the execution time based on the query, its cost and the current load vector. Once executed, the actual execution time is fed back into the predictive model. The authors use a classification tree to train the association between load vectors and queries and thus improve predictions over time. The model performs better the more queries it observes under several load conditions. The evaluation of the model is done by comparing it against real-world executions on an enterprise level DBMS. They achieve an overall average accuracy of 87%.

The applicability to the hybrid model for data-intensive systems is problematic in both cases. The analytical model needs extensive parameterization to express the performance of the underlying DBMS, which strongly conflicts with requirement 2, specified in Section 4.2. However, both models have one issue in common. It is unclear how the abstract, top-level operations map to multiple, specific, structured and, decomposable queries required by both models. From a data-intensive model point of view, the queries are already considered component specific, and

thus outside the scope per requirement 2. Also the current load of the DBMS, required by both models, is not available. Hence both models cannot be considered as component models for services.

As the related work is only very limitedly applicable to data-intensive system, the idea is to build two kinds of machine learning models. The first model is presented in Section 8.3 and is purely trained on cost indicators to predict the response times of operations. The second one, presented in Section 8.4, integrates queuing behavior to the model.

8.3 Cost indicator based response time prediction

The models presented in the related work section use the cost prediction of the query optimizer, to make an estimation about its execution time. Due to lack of information and the requirements described out in Section 4.2, this cannot be done for the hybrid model but there are other indicators which could indicate operational cost and thus correlate an operation with the response time. For example, the replica management- and replica selection service resolves a set of files to its replicas and makes a decision about which ones to use. One could assume that a request with a larger number of files takes longer than a request with a lower number of files, as more data needs to be processed. The number of files per data grid operation is expressed as the number of traces with the same $uuid$, see Table 4.1. Per data grid operation only one request is sent to the component stack. The historic component response time is given as $\Delta t = t_{transfer} - t_{start}$. Equation 8.1 is used to calculate the Pearson product-moment correlation coefficient between the number of files n in an operation and Δt and thus describes the linear relation between the two variables.

$$\rho_{n,\Delta t} = \frac{COV(n, \Delta t)}{\sigma_n \sigma_{\Delta t}} \tag{8.1}$$

The result for a one month dataset resulted in a correlation coefficient of 0.68 with a 95% confidence interval. This indicates a strong linear uphill relationship between the number of files requested in an operation and the response time. The assumption that the number of files is a good cost indicator for the performance of the query holds.

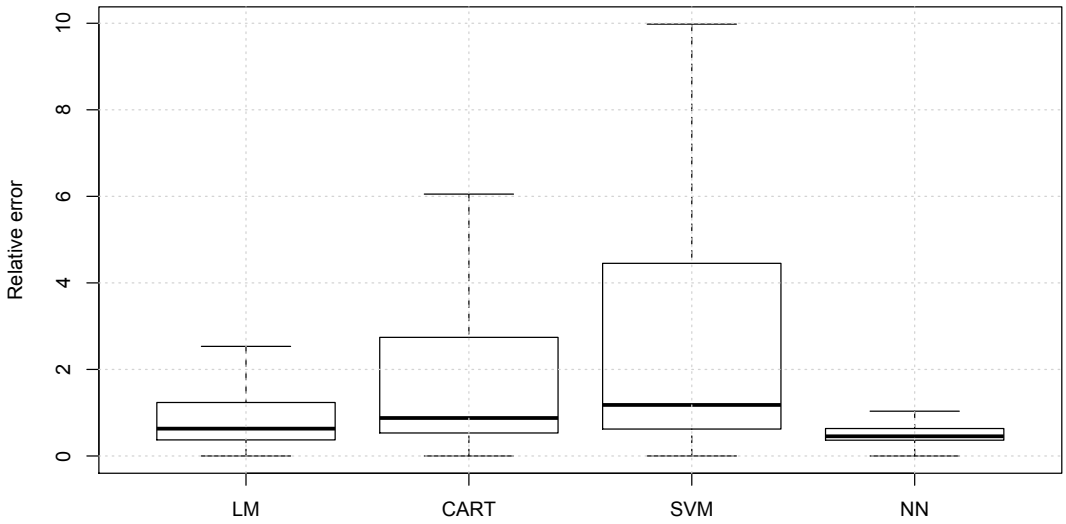

Figure 8.1: Validation of different models for the cost indicator based prediction of service component response time.

Figure 8.1 shows a boxplot of the relative error of models predicting the component response time based on the number of files of the operation. The relative error is defined as:

$$\delta_{\Delta t} = \left| \frac{\Delta t - \hat{\Delta t}}{\Delta t} \right| \tag{8.2}$$

The models are trained and evaluated with two different workloads. A linear model, Classification and Regression tree, Support Vector Machine and Neural Network are trained. The linear model performs comparably well with a median relative error of 60%. The neural network model however performs best with a median relative error of 36% and a much smaller standard deviation. Interestingly the CART and SVM model were not able to represent the linear relation of the two variables at all and showed large relative errors. The models accuracy, both in median relative error and standard deviation, are similar to the specific query execution time models presented in [Wu et al., 2013] and [Gupta et al., 2008]. Hence a very specific query execution time model would most likely only show little to no benefit.

8.4 Queue based response time prediction

The *transfer service* component is used to queue transfer requests and execute them at the right moment in time. The exact implementation of this depends on the data-intensive system. While some systems may execute transfers immediately, others can use queues to protect the system from overload. In the ATLAS data grid, an optimizer is used to analyze how much load a certain link can absorb and issues only a certain number of concurrent transfers, while holding the other requests back. This dynamic behavior makes the service component difficult to model, as no static rules are used to schedule the transfers. For a given link, the number of requests in the queue at the moment a request is submitted, gives some indication about the duration this request will be in the queue. Thus for each operation i the number of in_queue operations is defined as:

$$in\_queue(i) = \sum_{j=1}^{i-1} queued(i, j) \tag{8.3}$$

with $queued(x, y)$ defined as:

$$queued(x, y) = \begin{cases} t_{submitted\_at}(x) \leq t_{started\_at}(y) \leq t_{started\_at}(x) & 1 \\ \text{else} & 0 \end{cases} \tag{8.4}$$

Equation 8.3 only counts operations being in the queue at the moment the operation is added to the queue. The reasoning for this is that the model, at the time of estimation, also has no knowledge about the operations potentially being added to the queue later on.

The model is trained on the in_queue attribute to predict the duration Δt a request resides in the queue.

$$\Delta t = t_{started\_at} - t_{submitted\_at} \tag{8.5}$$

Figure 8.2 shows a boxplot of the relative error of models predicting the component response time, which is equal to the queue time, based on the queue behavior of the operation. All models are trained with a historic training workload and evaluated against a historic evaluation workload. The relative error is rather high for all models, even the best performing SVM shows a median relative error of 71%. Thus, modeling the transfer service queue time based on the amount of files in the queue at a given point in time is not optimal. Intuitively, this is expected

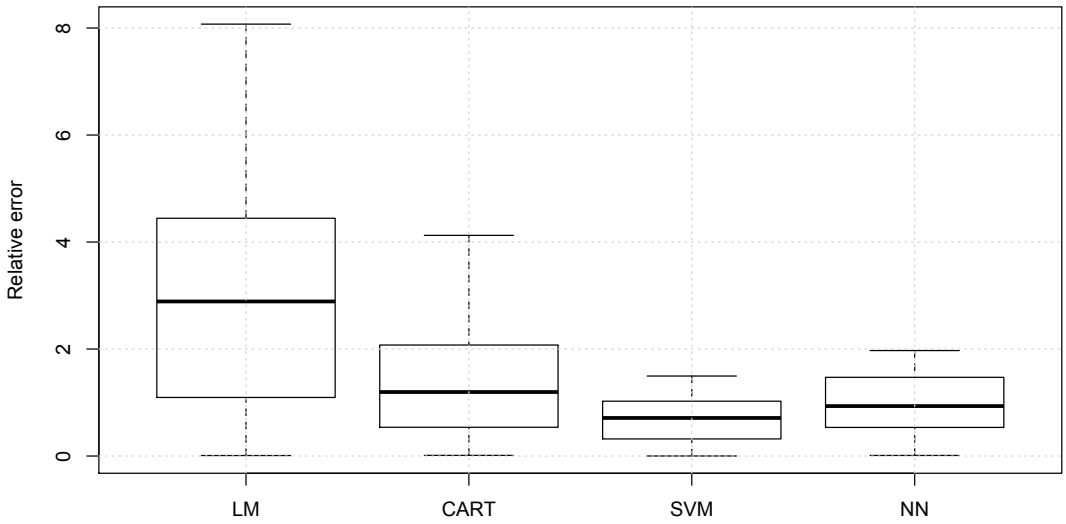

Figure 8.2: Validation of different models for the queue based prediction of service component response time.

behavior because the optimizer schedules transfers based on the number of transfers in flight, and some other performance characteristics. But this is difficult behavior to model without assuming some functionality of the optimizer, which would impact the models generality. However, the models generality is fundamental and the primary reason why the requirements in Section 4.2 were specified. Hence generality is favored here over accuracy. However, the median queue time of an operation is around 7 seconds, thus a median relative error of 70% does not impact the overall operation time too much.

8.5 Conclusion

This chapter presented two approaches how the different service components of a data-intensive system can be modeled. The chapter first introduced two models which predict the execution time of database queries. One based on machine learning, and one based on an analytical model. However, due to the requirements laid out in Section 4.2 and a lack of information of these

system details, other modeling approaches had to be considered. As the different service components are roughly based on two different paradigms, two models were introduced: one based on the number of files as a cost indicator and one based on the queue behavior. For both models, several machine learning techniques were evaluated against historic workloads. For the cost indicator based model the neural network approach performed best, while for the queue model the Support Vector Machine gave the best result.

The best performing underlying machine learning techniques are selected for each model for the inclusion in the hybrid system model for data-intensive systems.

Part III

Evaluation and conclusion

CHAPTER 9

Event simulator and evaluation

9.1 Introduction

The previous chapters proposed a modeling technique for components of a data-intensive system. This chapter focuses on bringing these component models together, unifying them in a full hybrid system model, integrating said model in an event simulator which is able to accurately simulate date-intensive systems, and finally evaluating the model against historic workloads of the ATLAS data grid.

The remainder of this chapter is structured as follows. In Section 9.2 the general workflows, input and output specifications and transformations, model implementations, and the general architecture and workflows of the event simulator are presented. Section 9.3 continues with the evaluation of the event simulator. The simulator is trained based on a historic workload of the ATLAS data-intensive system and evaluated against a different workload of the system. The same workloads are used to train and evaluate a model based on the GloBeM [Montes et al., 2011] methodology. The Chapter finishes in Section 9.4 with a conclusion.

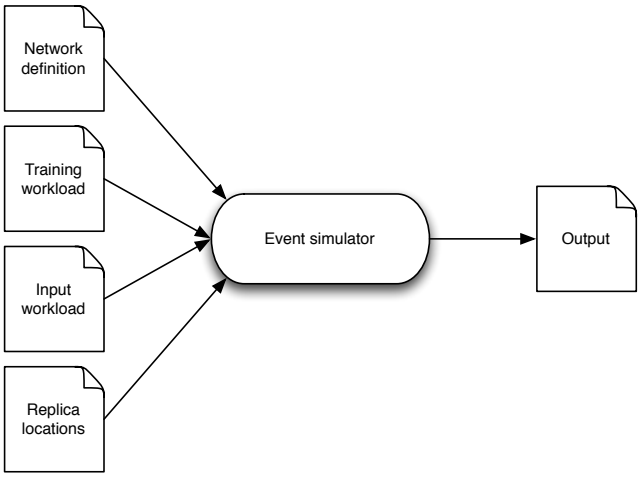

Figure 9.1: Data workflow of the event simulator.

9.2 Event simulator

The general principle of the hybrid simulation model and its implementation in an event simulator has been outlined in Section 4.3. This section picks up on this general outline and focuses on the architecture and implementation of the system simulator. The principle of a discrete event simulation is that the input of the system is a discrete sequence of events and each event changes the state of the system at a specific point in time. The natural event in terms of a data-intensive system is a read or write operation, consisting of several file accesses.

The data workflow of the event simulator is detailed in Figure 9.1. It works in two serial steps, an initialization phase, and a simulation phase. The initialization phase is used to construct the component models. The *network definition* is used to initialize the network model and the *training workload* is used to initialize all machine learning based component models. The *replica locations* define the mapping of files to available replicas. In some use-cases this can be derived from the input workload, in others it is necessary to have this definition. Once all component models are fully initialized the simulation phase is started. The *input workload* is iterated in a chronological order and for each operation the system model predicts the operations response time, which is stored in the *output*.

Attribute	Description
type	The type of the operation (download, upload, transfer).
file	The file identifier, typically the unique filename.
dataset	The dataset identifier the file belongs to.
source	Identifier of the source storage system.
destination	Identifier of the destination storage system.
size	Filesize in bytes.
protocol	Identifier of the protocol being used.
uuid	The universal unique identifier of this operation. In case of multiple transfers being executed in one system operation, all the operation will have the same uuid.
t_{submit}	Submission timestamp of the event.
t_{start}	Transfer initialization timestamp.
$t_{transfer}$	Timestamp of the start of the transfer execution.
$t_{validation}$	Timestamp of the end of the transfer and start of the validation phase.
t_{end}	Timestamp of the end time of the operation.

Table 9.1: Input workload attributes of the event simulator.

Input and output specification

The key requirement to the input specification is a clear and general definition which minimizes the effort to transform a system specific workload into something the event simulator can interpret.

The input workload is a comma separated file where each line represents one file transfer in the system. Table 9.1 shows the minimal set of attributes needed for the event simulation. The first section of attributes are the mandatory values needed for the simulation input workload. For the training workload additional attributes are needed for the model training, which are shown in the second section in Table 9.1. Usually, one system operation reads or writes multiple files. The relation of multiple transfers belonging to the same system operation is expressed by the $uuid$, hence file interactions of the same operation have the same uuid. Operations of the *transfer* type are always storage system to storage system transfers, thus the *source* and *destination* always specifies a storage system. *Upload* and *download* operations are performed from/to a storage system to/from a computer or worker node in some campus network or data center. For these events, the source or destination of the computing node is the specific network identifier of

Attribute	Description
file	The file identifier, typically the unique filename.
location	Identifier of the storage system the file has a replica at.

Table 9.2: Replica location attributes of the event simulator.

the campus or data center, as no mass storage system is used as either source or destination. The output workload is a the same event from the input workload, however, the estimated t_{end} timestamp is added.

The replica location listing is also a comma separated file, with the attributes defined in Table 9.2. The replica locations are necessary for simulation uses cases where the user does not replay a very specific workload but lets the data-intensive system model make some decisions. For example, for a download operation the source storage system is not defined in the input workload, but instead the system model must decide which source replica to use. In cases like this the event simulator must be aware of the initial distribution of replicas.

As SimGrid [Casanova et al., 2008] is used as the underlying model for the network component model, their definition of network topology and routing files are used as network definition. The network topology definition is an XML based file which defines all hosts, such as storage systems, routers, network links, and their interconnection. The routing file is also an XML based file which defines for each host pair a sequence of links. A more detailed topology definition is outlined in Section 6.2.

The input specifications are kept as general as possible, but any data-intensive system will require the workloads it produces to be transformed into the general event simulation inputs. For the ATLAS data-intensive system a small tool-stack developed in Python [Python Software Foundation, 2015] is used for these transformations.

Architecture & Workflow

A component overview of the simulator architecture is shown in Figure 9.2. The event simulator is composed of six components. The simulation stack is implemented in the C programming

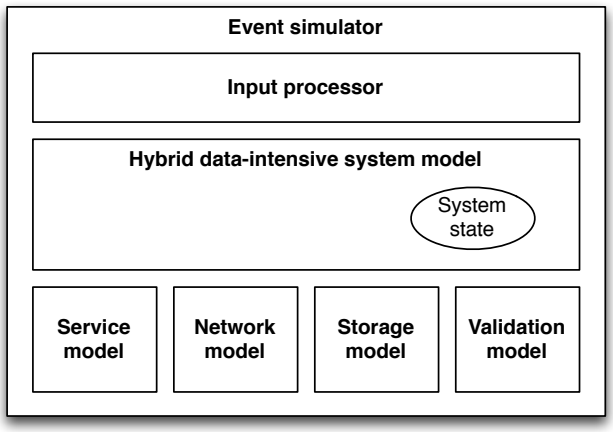

Figure 9.2: Overview of the architecture of the event simulator.

language. In some cases external libraries or Python programs are called via an embedded Python interpreter.

The *input processor* is responsible for processing the workload and handling the various simulation parameters. In simulation mode, it iterates the input workload and relays operations to the system model for response time estimation. The returned t_{end} value of the system model is then, together with the attributes of the input workload, written to the output file. In training mode, it only relays the locations of the training workload, the network definition, and the replica locations to the hybrid system model, as it is up to the component models to process the inputs in the most optimal way for training.

The *hybrid data-intensive system model* is the core component of the simulation. For each operation event coming from the input processor, it calls the component models for an estimator of the operation. The system model also holds a structure of the current system state in memory. A reference to the system state structure is given as pointer to each component model, thus when estimating an operation, the component model has access to the full simulated system state at that moment in time and can also modify the system state. The implementation of the hybrid system model is general enough that it does not have to be adapted to be able to simulate a specific data-intensive system. During initialization phase it uses the information from the replica locations input to update the system state to the respective replica locations. If no replica

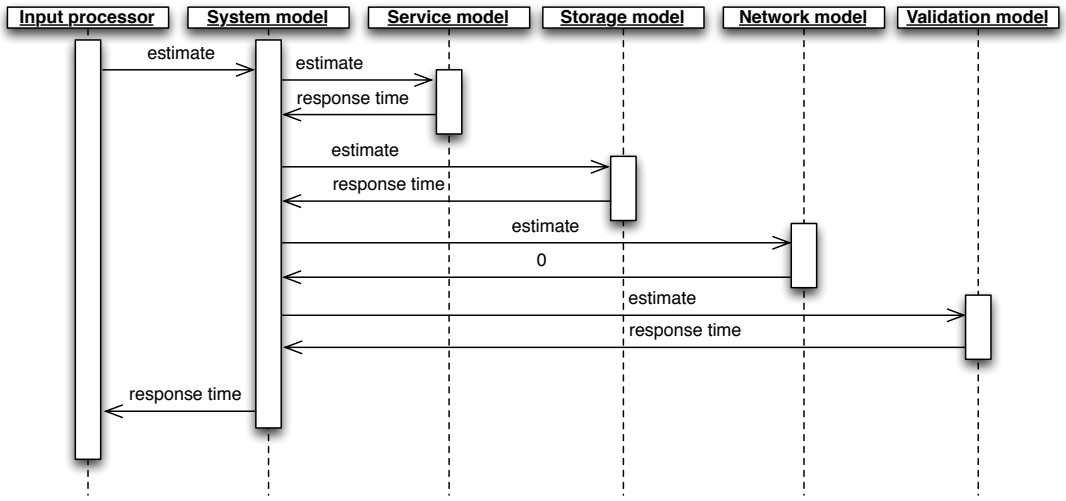

Figure 9.3: Simulation workflow for the estimation of a local read operation.

locations are given, it parses the input workload and derives the replica locations from it. For the component models, each component model's train method is called with a reference to the simulators input. The actual training is then done by the component model itself.

Figure 9.3 shows a sequence diagram for the simulation of one local read operation. The hybrid system model queries all the relevant component models for an estimation of the response time. In the example shown, a local-read operation is shown, thus the network model returns 0 as no WAN transfer is executed. Once all component estimates are retrieved, they are aggregated by the system model:

$$t_{end} = t_{submit} + \sum_{i=1}^{n} response\_time_i \tag{9.1}$$

In simulation mode the sequence shown in Figure 9.3 is only one iteration in the estimation loop of the input processor. At the end of each iteration the estimated time is written to the output file and the loop starts over with the next estimation.

All *component models* are derived from a class definition (defined as a struct in C) implementing two methods. See Figure 9.4 for a class diagram. All models have to implement these two methods, as they act as the interface to the hybrid system model. The complexity of both methods strongly depends on the complexity of the machine learning or analytical model approach being used, but next to these also some data-intensive system specific functionality might

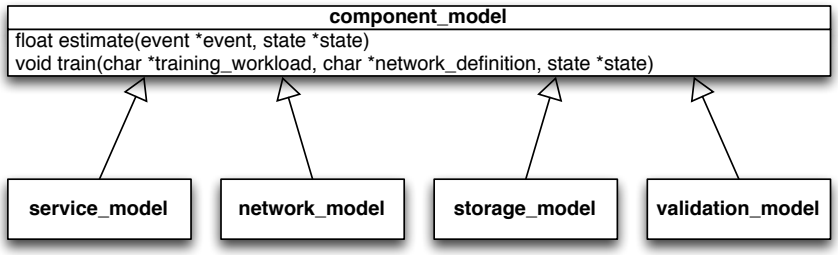

Figure 9.4: Class diagram of the component model.

be necessary to be implemented. The following generic component models are implemented:

- The *service_model* is split into two models. As mentioned in Chapter 8, the cost indicator based model (Neural network based) is used for the replica selection, replica management, and metadata service. The queue based model (Support Vector Machine based) is used for the transfer service. When calling the *estimate* method, the service model decides based on the operation type, source, and destination which of the specific models to call. See Table 4.11 for a mapping. In the case of the ATLAS data-intensive system, the service model also has some system specific implementations for the replica selection service. In cases where the input workload is only partially filled, thus the system model has to decide which source replicas to read, the replica selection service makes this decision. Hence simplified policy rule sets from the actual data-intensive system have to be implemented in the model. In the replica selection case this is a shortest path query in a graph weighted by the link latencies, which is generated during the training phase based on the network topology. Thus the replica closest to the source, measured by link latency, is selected.

- The *network_model* is based on the network model of the SimGrid [Casanova et al., 2008] framework. Specifically the API of the MSG stack is used. The train method is rather simple and just initializes the SimGrid core with the supplied XML platform files. The estimate method requires a more elaborate implementation, as the MSG stack only offers a very basic abstraction for message passing, thus all file transfer specific workflows are implemented in the network model on top of that. For local storage system accesses the

network model returns a response time of 0, as no wide area network interactions are executed in this case.

- The *storage_model* is based on the machine learning approach with Support Vector Machines presented in Chapter 5. In the training phase, the workload is partitioned into storage elements. If necessary, each partition is sampled using the stratum sampling approach. A SVM model is then trained for each of the partitions, thus creating one model per storage system. In the simulation phase the respective model where data is read from or written to is used for the response time estimation.

- The *validation_model* is on the basis of the Neural Network model presented in Chapter 7. In the training phase, the validation model also partitions the workload similarly to the storage model. While this approach does not appear to be very intuitive, it is guided by a very simple motive: The validation performance is mostly driven by the hardware (CPU, disk) of the computer executing the validation. As this information is not available, the partitioning by campus network/data center makes the assumption that computers in this cluster are usually similar. This assumption often holds, as worker nodes in a data center are very often homogeneous. Thus, by having these data center specific validation models instead of one global validation model, the prediction performance is improved. In the simulation phase the implementation of the estimate function only calculates a response time for the local-read and remote-read operation. For other operations 0 is returned, as the validation is done implicitly by the transfer system.

9.3 Evaluation

To evaluate the event simulator and thus the full hybrid simulation model for date-intensive systems, historical traces from real-world systems are simulated and compared. The idea is to inject multiple workloads, not only from the ATLAS data-intensive system, but also from other data-intensive systems. However, no other data-intensive system workloads and system descriptions could be obtained. Even the workloads from the Grid Workload Archive [Iosup et al., 2008] are not applicable to data-intensive systems as they are workloads of computing

grids and thus do not include any data handling. Unfortunately all operators of data-intensive systems which have been contacted in order to evaluate the model had no interest in sharing their data.

For the evaluation a four day workload of the ATLAS data-intensive system is selected and split into a three day training workload and a one day testing workload. The training workload includes about 8 million file transfers while the testing workload almost reaches 3 million transfers. These are comparably average utilized days for the ATLAS data-intensive system. While all file transfers are estimated by the data-intensive system model the evaluation is done on system operation level, as defined in the requirements in Section 4.2. Multiple files are read or written as part of one system operation. The estimation error is based on the difference in response time of the whole system operation. As the workload includes file-level operation, the response time rt of an operation x is defined as:

$$rt(x) = max(t_{end}(i)) - min(t_{submit}(i)), \quad \forall i | uuid(i) = uuid(x) \tag{9.2}$$

Similar to the system analysis the operations are partitioned into four partitions of total operation size between $x < 1$ Gb, $1 < x < 10$ Gb, $10 < x < 100$ Gb, and $100 < x$ Gb. The evaluation error used is the relative error of the response time defined as:

$$error = \frac{|\hat{rt} - rt|}{rt} \tag{9.3}$$

The general objective of the evaluation is not only to evaluate the full system model but also to compare the model against other models. However, as already discussed in Chapter 3, there are no models complying to the requirements laid out in Section 4.2. Most models do not support the full data workflows of a data-intensive system, as they are focused on computational jobs. The GridSim [Buyya and Murshed, 2002] and SimGrid [Casanova, 2001] simulators do have extensions for data workflows, but both the GridSim and the SimGrid model require storage system characteristics and parameters which are not available. Some experiments were conducted by essentially guessing and tuning these parameters, but the simulation errors were extremely high and not representable. However, the authors of both publications show comparable results in their articles when a detailed full system description, such as in a testbed, is available. Yet, this

is not the case for large-scale production systems like the ATLAS data-intensive system. The scope and environment for both models are just too different to accurately compare them and would not give the models proper credit.

The model which is most applicable to data-intensive systems and complying to the requirements is the GloBeM [Montes et al., 2011] model. While originally developed to be a model for grid computational job behavior prediction the approach can also be extended to data workflows. However, this was never demonstrated by the authors. Furthermore the source code of the model has never been published by the authors, thus a simple adaption is not possible. Hence a GloBeM model for data-intensive system has to be implemented in order to evaluate it in this thesis. As the implementation is not necessarily what the original authors would have in mind, the model will be referred to as *GloBeM-like* model. The general principle of GloBeM is to abstract the system as one entity and monitor state changes in the system. These state changes are classified and used to train a model, which later on can be used to predict state changes in a time series. The general advantage is that the model is very general and does not require knowledge about the inner mechanics of the system. The set of states and a workload containing the state changes over time is sufficient. The idea of the GloBeM-like model is to instead of predicting state changes of a computing job to predict state changes of a read or write operation. The individual steps in the file transfer are also split up into state changes, thus one read or write operation goes through several transfer states for each file. For the evaluation the estimated response time, similar to Equation 9.2, is calculated for each operation. The same three day training workload and one day testing workload is used to evaluate the GloBeM-like model. For the underlying machine learning method Neural Networks are used as they generally perform well.

Figure 9.5 shows the evaluation of the hybrid simulation model for data-intensive systems. The median relative simulation error of the entire model is at 22%, however, this is mostly due to the fact that most operations fall into the first partition of operations smaller than 1 Gb. The error gets larger the larger the operations get. One interesting fact is that the variance of the relative error gets smaller the larger the operations get, while usually the inverse effect would be expected.

Figure 9.6 shows the evaluation of the GloBeM-like model. This model uses less knowledge

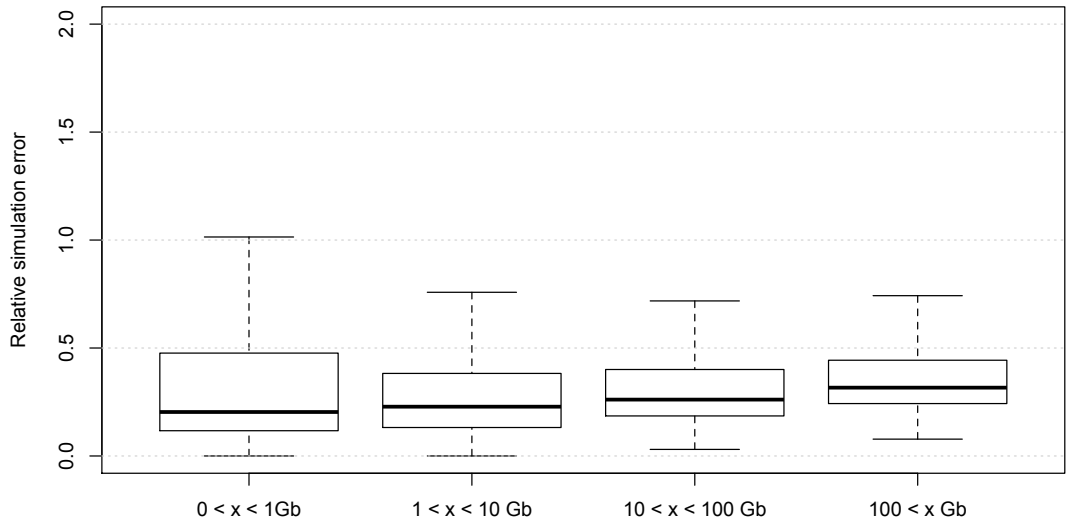

Figure 9.5: Relative simulation error of the hybrid simulation model for data-intensive systems.

of the internals of the system than the hybrid simulation model and relies more on machine learning techniques to capture the state-changing behavior of the system. The median relative error is at 72%. In general the variances are also much higher and the method seems to capture the behavior less well, as the accuracy seems to be quite erratic in each partition. However, the result, especially for some partitions, is still striking, considering that the methodology does not have any knowledge about the internals of the system. Also, the time spent in optimizing the GloBeM-like model is quite limited, thus the results should be taken as a reference, not necessarily as a in-depth analysis of the methodology.

9.4 Conclusion

This chapter presents the evaluation of the hybrid simulation model for data-intensive systems and compares the model against the only available model in literature. After the introduction, Section 9.2 proposes an event simulation framework in which the hybrid simulation model gets executed. The input and output specification of the simulator is presented followed by the archi-

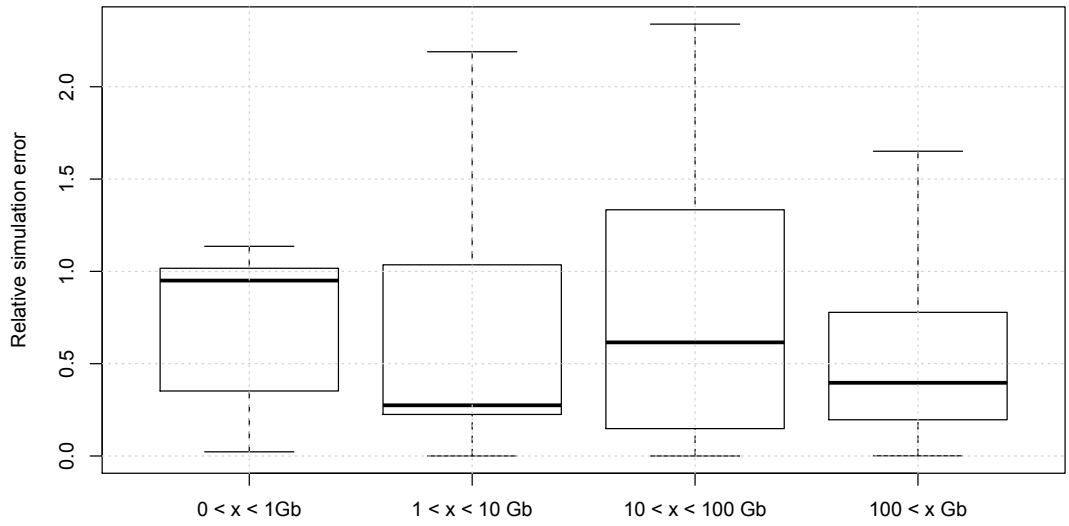

Figure 9.6: Relative simulation error of the GloBeM-like model.

tecture and workflow of the simulator. The workflows and specifics of the component models, and how they are integrated into the simulation framework, are presented as well. The evaluation section first introduces the response time metric being used to evaluate the estimators. The environment of the evaluation, describing the training and simulation workloads is clearly specified. The section continues by introducing a data-intensive system focused model, based on the GloBeM methodology, which was implemented in order to compare the hybrid simulation model. As there are no other applicable models in literature, only the GloBeM-like model could be used as a reference for comparison.

The evaluation showed a median relative error of the hybrid simulation model of 22% while the GloBeM-like model performed at 72%. The hybrid simulation model also showed a smaller spread of the error.

CHAPTER 10

Summary and conclusions

This thesis investigates performance models for data-intensive systems. The research question is stated as "*can a model be constructed that estimates the response time of user operations in a data-intensive system?*". At first, the ATLAS data-intensive system Rucio [Garonne et al., 2012, 2014], which acts as a running example throughout this thesis, is introduced and outlined. By analyzing existing full system simulation models in literature, the challenges, weaknesses, and limits of existing approaches are identified. These findings are compiled into three requirements for a model of an operational data-intensive system. Most models in literature are either not applicable to data-intensive workflows, or make assumptions which do not hold for operational systems, such as the level of specific system knowledge required. Steered by these requirements the novel hybrid simulation model approach for data-intensive systems is proposed. The hybrid modeling approach partitions the system into discrete components, creates models for these components, and combines them to one concise system model which is able to predict the performance of user operations. The identification of these system components results from a quantitative system analysis of the Rucio data-intensive system. The storage, network, data integrity validation, and services components were identified. For the storage model existing models from three different paradigms were evaluated but identified as non-applicable to the domain of data-intensive systems. Three machine learning techniques, namely Classification and Regression Trees, Support Vector Machines, and Neural Networks were presented as well.

Models are built based on these techniques and evaluated by extracting workloads, for different storage systems, from the Rucio system. The models are trained and evaluated with different workloads from the operational system. For the network model two well established modeling paradigms were evaluated. Packet-level models represent the networking system with almost no abstraction and simulate single TCP/IP packets, whereas flow-based [Anick et al., 1980] models abstract multiple packets as a single flow. Both models have been shown to perform similarly accurate for network flows over 100kB in size. However, the packet-level model does not scale with the volume of data moved in data-intensive systems, resulting in simulation times which are orders of magnitude larger than the simulated time. The data integrity validation model captures the performance of data integrity validation workflows by training a Neural Network model with historic workloads. The service component model is more complex, as several different service workflows have to be mapped by the model. Two approaches, one based on cost indicators and one based on queue behavior, are proposed and evaluated. All component models are implemented in an event simulation framework and evaluated against a historic workload of the ATLAS data-intensive system. The hybrid simulation model performs with an median relative error of 22%. A current state-of-the-art GloBeM [Montes et al., 2011] based modeling approach, which had to be adapted to conform to data-intensive systems, performed with an relative error of 72%.

10.1 Contributions

Modeling data-intensive workflows

The vast majority of literature focuses on computational uses cases. The question of what system components influence the performance of a read or write workflow arises. This is addressed in Chapter 4 by a quantitative analysis of historic workloads of the running example Rucio. The system collects extensive traces since the beginning of 2009. Each trace describes a file access, such as a read or write, and is described by more than 30 different attributes. Among these are also several timestamps during certain milestones of the operation. For each of the different user access workflows, such as remote and local read, remote and local write, and site to site transfers,

different system components are accessed during the workflows. For the purpose of the analysis the entire workload of the year 2015 is analyzed, including about 300 million local uploads, 70 thousand remote uploads, 9 million local reads, 3 million remote reads, and 25 million site to site transfers. However, each operation includes multiple file interactions, in total about two billion transfers. The analysis assigns the exercised components to their generic components in the reference architecture [Allcock et al., 2002]. Then the measured periods for each workflow are mapped to the components and the data is analyzed for statistical significance. A quite trivial finding is that the network component is insignificant for local workflows, but other more surprising findings are presented. Thus for small remote read operations the querying of the replica selection services takes as much time as the transfer itself, thus making this component highly significant for a performance model. For the write workflows the registration of replicas and locations at the metadata components takes a significant amount of time, even for larger operations. For the site to site transfers a significant amount of time is spent in queueing the transfers. Some of these findings point to deficits in the data management software, however, the performance issues have to be represented by the model nevertheless, otherwise no accurate estimation can be made by the model. Hence, a common mistake in literature is to make assumptions about the underlying system being modeled, such as that certain parts of the workflow are fast and insignificant. However, a model must be guided by the observed data, not by the assumption of how a system should work. Of the seven components defined by the data grid reference architecture only two to four components are statistically significant for each workflow. Due to the reduction of the component workflows to the performance significant parts, it also decreases the complexity of the hybrid model.

Requirements for modeling operational data-intensive systems

Creating a model for a data-intensive system is inherently different from creating a model for a conventional data processing system. These differences can be divided into two categories: access and scale. For a conventional data processing system it is unquestionable that the researcher has full access to all hardware and software components of the system. Thus creating a model for the system can be done under the assumption of full knowledge. Data-intensive

systems are large-scale distributed systems and operate under entirely different idiosyncrasies. There is no detailed knowledge about many parts and specifics of the system. But even if this access was possible, collecting instrumentation data is on a much bigger scale than for a conventional data processing system and likely infeasible. Further more, there are no test-beds for data-intensive systems. While a conventional data processing system can be exercised with tests in a contained environment, for example to create more extensive workloads or test the system under distinct conditions, the scale prohibits the creation of a representative test environment for multi-continental, large-scale data-intensive systems. Thus modeling the data-intensive system means interacting with an operational system, hence running the risk of operational impact or even jeopardizing operational data. These observations lead to the formulation of three requirements in modeling data-intensive systems which are presented in detail in Chapter 4: *1. The model predicts the response time of individual read and write operations; 2. The data grid components are black boxes. Component models must be constructed based on global observations, rather than internal component data or knowledge; 3. Only passive observations must be used instead of observations from controlled system experiments.* These requirements are driven by operational conditions and thus limit the approach compared to systems modeled under laboratory conditions. However, the formulation of such requirements is essential to not only define the constraints of the model, but also to give other researcher the possibility to compare and reflect on the requirements and assumptions they make about their models.

Modeling data-intensive system components

The storage model, presented in Chapter 5, relies on machine learning techniques. Namely models based on Classification and Regression tress, epsilon regression Support Vector Machines, and feed-forward Neural Networks models were implemented and evaluated. To evaluate the accuracy of the models workloads from five different storage systems of different sizes and underlying technology were extracted from the system. The workloads were enriched with an attribute representing the number of concurrent accesses at a certain time in the system. Each model estimates the transferrate of a single file access, thus the data is normalized by filesize. The Classification and Regression tree model shows an relative median error of around 30%.

For two of the five storage systems it performs with a high standard deviation, with the 9th percentile of the relative error over 400%. The Support Vector Machine model shows a slightly better median performance around 25%, but a generally lower standard deviation with only one storage model at 300% for the 9th percentile of the relative error. The Neural Network model showed a slightly larger median relative error of 28% but a more consistent result with smaller standard deviations of the error for all storage systems. The neural network model shows no 9th percentile of the relative error larger than 150%.

For the network component, presented in Chapter 6, an already established model used in literature is utilized.

The data integrity validation model presented in Chapter 7 aims to accurately represent the data integrity validation workflow of a data-intensive system. The methodology is quite simple, as the validation process is mostly dependent on environmental conditions, such as the local CPU or harddisk, which the model has no knowledge about. Thus the model is built on the filesize of each specific file access. The assumption that there is a linear relation between the filesize and the validation time is confirmed by experiment. A Pearson product-moment test confirms a strong uphill relationship of the data, indicating that a linear model should work well for the component. The model creation is based on the previously used machine learning techniques. Also a linear model is added to the lineup. The evaluation shows a median relative error of 17% for the neural network model. However, even the linear model performs at an median relative error of 23%, confirming the strong linear relationship of the independet variables.

The services models presented in Chapter 8 are the most challenging ones. The model must be general enough to apply to any service component of a data-intensive system. It cannot mock the internal behavior of a specific component. But while being generic, it must also deliver accurate predictions. Two modeling approaches were presented. A cost indicator based response time prediction where a variable in the input is used as a cost indicator, describing the complexity of the operation. For example, the number of files covered by an operation is a good cost indicator for many queries. For three of the four service components the number of files can be used as a cost indicator. A strong uphill relationship of 0.68 with a 95% confidence interval in a Pearson product-moment correlation test can be shown between the service time

and the number of files for those operations. The cost indicator based model is utilizing a neural network which is evaluated with a median relative error of 36%. The second model is based on queuing behavior and aims to model transfer queues in front of a transfer system. This is done by training on the historic behavior of previous transfers, by correlating the number of transfers in the queue with the time they spent in the queue. The Support Vector Machine model performed best with a median relative error of 71%.

The presented models accurately predict the performance of their modeled components. Some of the machine learning techniques are also not ideal to answer certain questions, such as *What happens if a better performing storage system is used at this site?* It is difficult to adapt certain models, such as a neural network or support vector machine, to new parameters. The representation of the trained model is very difficult to interpret for humans. For other techniques, such as Classification and Regression trees the adaption of the model is easier. These issues will be addressed in future work, but are outside of the scope of this thesis.

Modeling and predicting a high number of events

Operational workloads of data-intensive systems process, in a short time-frame, a very high number of events. This raises two questions in the context of modeling the system. Can a model be trained based on such high-volume workloads? But also, once the model is created, can it estimate input workloads of such cardinality? The first problem is addressed in Section 5.6. All tested machine learning techniques can not cope with workloads larger than 100.000 events in a reasonable amount of time. This can be improved by using better hardware but the sizes of data-intensive workloads are orders of magnitude bigger, thus this quickly gets into an unrealistic area. Sampling methods have to be used to reduce high-volume workloads to a digestible size. However, the risk when sampling data is that the intrinsic characteristic of the workload gets lost. A model trained with such a workload would thus be less accurate. This effect is shown in Section 5.6 when a model, trained with an entire workload is compared with a model trained with a 10% sized random sample of the workload. The model trained with the sampled workload performs far less accurate and less stable. However, sampling methods exist which decrease the volume of the dataset while trying to preserve its intrinsic characteristic.

One of such methods is called stratified sampling [Neyman, 1934] where the dataset is divided into stratums which are then sampled systematically. As the defining stratum characteristic the number of concurrent operations is used. The analysis shows that a model trained with a stratum sampled workload with 10% of the size of its original still performs similar to the model trained with the full workload. When estimating a workload all used models show no issues with predicting large workloads, as each operation is predicted in isolation, thus when estimating, the model is independent from the size of the workload. This part of the work shows that data-intensive system models are prone to a high number of events in a workload and cannot cope with them. When applying special systematical sampling techniques to the workload, the characteristics of the workload can be preserved and modeling made possible.

Unifying models in a hybrid system model and integration in an event simulator

The proposed modeling techniques for system components are unified in one full-scale, hybrid system model for data-intensive systems, which is described in Chapter 9. The architecture unifies the C implementations of the models in an event simulator. The actual system specific information is supplied by multiple inputs: a network definition, describing the network graph, a training workload, which is used by the different component models to extract specific component information from the system, a replica location catalog, defining an initial replica distribution, and the actual input workload which is being estimated. After all component models are trained or initialized, the event simulator iterates the input workload and queries each component model for an estimator and interpretation of each single event. The estimators are aggregated by the hybrid model and written to an output file. The evaluation is done by training the hybrid model with workloads from the Rucio system. The event simulator is then exercised with a separate evaluation workload from the system. The median evaluation error of the model is at 22%, with an relative interquartile range error of around 30%. However, the error increases with operation size, resulting in a median relative error of 33% for operations larger than 100Gb. Surprisingly, the spread of the error decreased with growing operation size, while the inverse behavior would be expected. As no related work was applicable to the use cases of operational data-intensive systems, the concept of the GloBeM [Montes et al., 2011] methodology was ex-

tended to data-intensive systems. The GloBeM-like model performed with an median relative error of 72%, for the same training and evaluation workload. Thus, this thesis improves the state of the art by 40%.

10.2 Publications

Significant parts of this thesis were published as articles in peer-reviewed proceedings of international conferences or as peer-reviewed articles for international journals. All articles were presented at their respective conferences as public talks or posters. While the core contributions of this thesis are my own work, some articles, specifically where the model is used as a method to investigate the data-intensive system, are shared work with my colleagues at CERN. Most notably, Mario Lassnig, Vincent Garonne, Angelos Molfetas, Thomas Beermann and Cedric Serfon. According to Google Scholar in January 2017, the articles have been cited more than 170 times.

Making cluster applications energy-aware - Vasić, Barisits, Salzgeber, and Kostic [2009]
This paper describes a novel approach in making cluster applications, such as distributed file systems, energy-aware. The architecture actively and dynamically powers done nodes based on the utilization pattern observed in the cluster. The approach is evaluated with a special implementation of the Hadoop MapReduce framework. This article was presented at the workshop for Automated Control for Datacenters and Clouds at the 6th IEEE/ACM International Conference on Autonomic Computing and Communications (ICAC'09) in Barcelona, Spain. While this work has no direct part in this thesis, the need of accurate performance models for operational systems became apparent to me while conducting this research.

A similarity measure for time, frequency, and dependencies in large-scale workloads - Lassnig, Fahringer, Garonne, Molfetas, and Barisits [2011]
This work describes a novel similarity metric for workloads of data-intensive systems. The method is comprised of two parts, one using discrete wavelet transform to investigate the time

and frequency characteristics of the workload. The other evaluates attributes based on association rule learning. The work was presented at the 24th ACM SIGARCH / IEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC'11) in Seattle, USA.

Popularity framework for monitoring user workload - Molfetas, Lassnig, Garonne, Stewart, Barisits, Beermann, and Dimitrov [2012]
This work describes a methodology to infer information from collected tracing data that allows a data-intensive system to be queried for usage patterns of specific data. This article was presented at the 19th International Conference on Computing in High Energy and Nuclear Physics (CHEP'12) in New York, USA.

Simulating the ATLAS Distributed Data Management System - Barisits, Garonne, Lassnig, and Molfetas [2012]
This article presents a full-scale simulation framework for LHC-specific distributed data management systems. The simulator was evaluated against workloads of the ATLAS data management system DQ2 [Branco et al., 2010]. This work was presented at the 19th International Conference on Computing in High Energy and Nuclear Physics (CHEP'12) in New York, USA.

The ATLAS Distributed Data Management project: Past and Future - Garonne, Stewart, Lassnig, Molfetas, Barisits, Beermann, Nairz, Goossens, Barreiro Megino, Serfon, Oleynik, and Petrosyan [2012]
This article describes the state of the ATLAS distributed data management system DQ2 and outlines recent contributions. The work includes an overview of the updated architecture and preliminary benchmarks of novel data placement algorithms. The article was presented as part of the IOP Journal of Physics, 2012.

Advances in service and operations for ATLAS data management - Stewart, Garonne, Lassnig, Molfetas, Barisits, Zhang, Calvet, Beermann, Megino, Tykhonov, Campana, Serfon, Oleynik,

and Petrosyan [2012]
This paper describes the recent advances in the ATLAS data management system. It specifically outlines the workflows of dynamic data placement algorithms which detect and replicate, popular data. The work was presented at the 14th International Workshop On Advanced Computing And Analysis Techniques In Physics Research (ACAT'14) in London, UK.

The ATLAS Distributed Data Management System & Databases - Garonne, Lassnig, Barisits, Beermann, Vigne, and Serfon [2013]
This overview paper outlined the architecture and usage of large-scale databases as part of the ATLAS data-intensive system. The work was presented at the Extremely Large Database Europe workshop (XLDB'13) in Geneva, Switzerland.

ATLAS Replica Management in Rucio - Barisits, Serfon, Garonne, Lassnig, Stewart, Beermann, Vigne, Goossens, Nairz, Molfetas, and on behalf of the ATLAS Collaboration [2014]
This article describes a novel expressional language which allows operators and users of a data management system to dynamically allocate and replicate data. The semantics of the language enable the definition of data management policies via a simple and robust syntax. The article is part of the IOP Journal of Physics, 2014.

Popularity Prediction Tool for ATLAS Distributed Data Management - Beermann, Maettig, Stewart, Lassnig, Garonne, Barisits, Vigne, Serfon, Goossens, Nairz, and Molfetas [2014]
This work describes a novel machine learning approach in predicting the popularity of newly registered datasets in the ATLAS data grid. This contribution then enables the system to replicate the data based on its predicted popularity. The article was presented at the 11th International Symposium on Grids and Clouds (ISGC'14) in Taipei, Taiwan.

Rucio, the next-generation Data Management system in ATLAS - Serfon, Barisits, Beermann, Garonne, Goossens, Lassnig, Nairz, and Vigne [2014]
This contribution outlines the design and architecture of a novel data management system used

by the ATLAS collaboration. The system is designed to manage hundreds of Petabytes of data and cope with frontend interaction rates of hundreds of Hertz. The article was presented at the 37th International Conference on High Energy Physics (ICHEP'14) in Valencia, Spain.

Resource control in ATLAS distributed data management - Barisits, Serfon, Garonne, Lassnig, Beermann, and Vigne [2015]
This article describe a novel approach of defining resource allocations and dynamically managing resources in the distributed data management system. The work was presented at the 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP'15) in Okinawa, Japan.

Scalable and fail-safe deployment of the ATLAS Distributed Data Management system Rucio - Lassnig, Vigne, Beermann, Barisits, Garonne, and Serfon [2015]
This contribution details the automatic deployment management of the ATLAS distributed data-intensive system. The design of the deployment workflow and monitoring is outlined as well. This article was presented at the 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP'15) in Okinawa, Japan.

A hybrid simulation model for data grids - Barisits, Kühn, and Lassnig [2016a]
This article describes a novel simulation model for data grids. The approach partitions data grid workflows into different components and constructs machine learning based models for each component. The components are then combined into a full-scale system model. This article was presented at the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'16) in Cartagena, Colombia.

C3PO - A Dynamic Data Placement Agent for ATLAS Distributed Data Management - Beermann, Lassnig, Barisits, Serfon, and Garonne [2016]
This contribution introduces a new dynamic data placement algorithm for the ATLAS data grid. This method is designed to pre-place potentially popular data to make it more widely available

by predicting the processing utilization at a given point in time in the future. This article was published as part of the IOP Journal of Physics, 2016.

Automatic rebalancing of data in ATLAS distributed data management - Barisits, Serfon, Garonne, Lassnig, and Beermann [2016b]
This article describes a methodology to predict potential resource shortages and pro-actively, and dynamically, rebalance data to other data centers to avoid resource contention. The algorithm always enforces replication policies, such as replication factors, but tries to optimize data access performance by data movement. This article was presented at the 22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP'16) in San Francisco, USA.

10.3 Physics contributions

Besides the contributions this research made in the field of computer science, major parts of the findings of this thesis had direct technical impact on the software and operation of the ATLAS data grid, which directly led to physics results. Thus more than 80 co-authorships for peer-reviewed physics articles were completed within the ATLAS collaboration. Most importantly, I hope that this work played its role in the discovery of a new particle that is consistent with the Standard Model Higgs Boson, a discovery which led to the 2013 Nobel Prize in Physics for Peter Higgs and François Englert.

Bibliography

Georges Aad, T Abajyan, B Abbott, J Abdallah, S Abdel Khalek, AA Abdelalim, O Abdinov, R Aben, B Abi, M Abolins, and Others. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716 (1):1–29, 2012. doi: 10.1016/j.physletb.2012.08.020.

LSST Science Collaborations: Paul A. Abell, Julius Allison, Scott F. Anderson, John R. Andrew, J. Roger P. Angel, Lee Armus, David Arnett, S. J. Asztalos, Tim S. Axelrod, Stephen Bailey, D. R. Ballantyne, Justin R. Bankert, Wayne A. Barkhouse, Jeffrey D. Barr, L. Felipe Barrientos, Aaron J. Barth, James G. Bartlett, Andrew C. Becker, Jacek Becla, Timothy C. Beers, Joseph P. Bernstein, Rahul Biswas, Michael R. Blanton, Joshua S. Bloom, John J. Bochanski, Pat Boeshaar, Kirk D. Borne, Marusa Bradac, W. N. Brandt, Carrie R. Bridge, Michael E. Brown, Robert J. Brunner, James S. Bullock, Adam J. Burgasser, James H. Burge, David L. Burke, Phillip A. Cargile, Srinivasan Chandrasekharan, George Chartas, Steven R. Chesley, You-Hua Chu, David Cinabro, Mark W. Claire, Charles F. Claver, Douglas Clowe, A. J. Connolly, Kem H. Cook, Jeff Cooke, Asantha Cooray, Kevin R. Covey, Christopher S. Culliton, Roelof de Jong, Willem H. de Vries, Victor P. Debattista, Francisco Delgado, Ian P. Dell'Antonio, Saurav Dhital, Rosanne Di Stefano, Mark Dickinson, Benjamin Dilday, S. G. Djorgovski, Gregory Dobler, Ciro Donalek, Gregory Dubois-Felsmann, Josef Durech, Ardis Eliasdottir, Michael Eracleous, Laurent Eyer, Emilio E. Falco, Xiaohui Fan, Christopher D. Fassnacht, Harry C. Ferguson, Yanga R. Fernandez, Brian D. Fields, Douglas Finkbeiner, Eduardo E. Figueroa, Derek B. Fox, Harold Francke, James S. Frank, Josh Frieman, Sebastien Fromenteau, Muhammad Furqan, Gaspar Galaz, A. Gal-Yam, Peter Garnavich, Eric Gawiser,

John Geary, Perry Gee, Robert R. Gibson, Kirk Gilmore, Emily A. Grace, Richard F. Green, William J. Gressler, Carl J. Grillmair, Salman Habib, J. S. Haggerty, Mario Hamuy, Alan W. Harris, Suzanne L. Hawley, Alan F. Heavens, Leslie Hebb, Todd J. Henry, Edward Hileman, Eric J. Hilton, Keri Hoadley, J. B. Holberg, Matt J. Holman, Steve B. Howell, Leopoldo Infante, Zeljko Ivezic, Suzanne H. Jacoby, Bhuvnesh Jain, R, Jedicke, M. James Jee, J. Garrett Jernigan, Saurabh W. Jha, Kathryn V. Johnston, R. Lynne Jones, Mario Juric, Mikko Kaasalainen, Styliani, Kafka, Steven M. Kahn, Nathan A. Kaib, Jason Kalirai, Jeff Kantor, Mansi M. Kasliwal, Charles R. Keeton, Richard Kessler, Zoran Knezevic, Adam Kowalski, Victor L. Krabbendam, K. Simon Krughoff, Shrinivas Kulkarni, Stephen Kuhlman, Mark Lacy, Sebastien Lepine, Ming Liang, Amy Lien, Paulina Lira, Knox S. Long, Suzanne Lorenz, Jennifer M. Lotz, R. H. Lupton, Julie Lutz, Lucas M. Macri, Ashish A. Mahabal, Rachel Mandelbaum, Phil Marshall, Morgan May, Peregrine M. McGehee, Brian T. Meadows, Alan Meert, Andrea Milani, Christopher J. Miller, Michelle Miller, David Mills, Dante Minniti, David Monet, Anjum S. Mukadam, Ehud Nakar, Douglas R. Neill, Jeffrey A. Newman, Sergei Nikolaev, Martin Nordby, Paul O'Connor, Masamune Oguri, John Oliver, Scot S. Olivier, Julia K. Olsen, Knut Olsen, Edward W. Olszewski, Hakeem Oluseyi, Nelson D. Padilla, Alex Parker, Joshua Pepper, John R. Peterson, Catherine Petry, Philip A. Pinto, James L. Pizagno, Bogdan Popescu, Andrej Prsa, Veljko Radcka, M. Jordan Raddick, Andrew Rasmussen, Arne Rau, Jeonghee Rho, James E. Rhoads, Gordon T. Richards, Stephen T. Ridgway, Brant E. Robertson, Rok Roskar, Abhijit Saha, Ata Sarajedini, Evan Scannapieco, Terry Schalk, Rafe Schindler, Samuel Schmidt, Sarah Schmidt, Donald P. Schneider, German Schumacher, Ryan Scranton, Jacques Sebag, Lynn G. Seppala, Ohad Shemmer, Joshua D. Simon, M. Sivertz, Howard A. Smith, J. Allyn Smith, Nathan Smith, Anna H. Spitz, Adam Stanford, Keivan G. Stassun, Jay Strader, Michael A. Strauss, Christopher W. Stubbs, Donald W. Sweeney, Alex Szalay, Paula Szkody, Masahiro Takada, Paul Thorman, David E. Trilling, Virginia Trimble, Anthony Tyson, Richard Van Berg, Daniel Vanden Berk, Jake VanderPlas, Licia Verde, Bojan Vrsnak, Lucianne M. Walkowicz, Benjamin D. Wandelt, Sheng Wang, Yun Wang, Michael Warner, Risa H. Wechsler, Andrew A. West, Oliver Wiecha, Benjamin F. Williams, Beth Willman, David Wittman, Sidney C. Wolff, W. Michael Wood-Vasey, Przemek Wozniak, Patrick

Young, Andrew Zentner, and Hu Zhan. *LSST Science Book, Version 2.0*. eprint arXiv, 2009. URL `http://arxiv.org/abs/0912.0201`.

Mark Adler. Adler-32 is a checksum algorithm. Technical report, RFC 1950, May, 1996.

M. Aguilar, J. Alcaraz, J. Allaby, B. Alpat, G. Ambrosi, H. Anderhub, L. Ao, A. Arefiev, P. Azzarello, E. Babucci, L. Baldini, M. Basile, D. Barancourt, F. Barao, G. Barbier, G. Barreira, R. Battiston, R. Becker, U. Becker, L. Bellagamba, P. Béné, J. Berdugo, P. Berges, B. Bertucci, A. Biland, S. Bizzaglia, S. Blasko, G. Boella, M. Boschini, M. Bourquin, L. Brocco, G. Bruni, M. Buénerd, J.D. Burger, W.J. Burger, X.D. Cai, C. Camps, P. Cannarsa, M. Capell, D. Casadei, J. Casaus, G. Castellini, C. Cecchi, Y.H. Chang, H.F. Chen, H.S. Chen, Z.G. Chen, N.A. Chernoplekov, T.H. Chiueh, K. Cho, M.J. Choi, Y.Y. Choi, Y.L. Chuang, F. Cindolo, V. Commichau, A. Contin, E. Cortina-Gil, M. Cristinziani, J.P. da Cunha, T.S. Dai, C. Delgado, J.D. Deus, N. Dinu, L. Djambazov, I. D'Antone, Z.R. Dong, P. Emonet, J. Engelberg, F.J. Eppling, T. Eronen, G. Esposito, P. Extermann, J. Favier, E. Fiandrini, P.H. Fisher, G. Fluegge, N. Fouque, Yu. Galaktionov, M. Gervasi, P. Giusti, D. Grandi, O. Grimms, W.Q. Gu, K. Hangarter, A. Hasan, V. Hermel, H. Hofer, M.A. Huang, W. Hungerford, M. Ionica, R. Ionica, M. Jongmanns, K. Karlamaa, W. Karpinski, G. Kenney, J. Kenny, D.H. Kim, G.N. Kim, K.S. Kim, M.Y. Kim, A. Klimentov, R. Kossakowski, V. Koutsenko, M. Kraeber, G. Laborie, T. Laitinen, G. Lamanna, E. Lanciotti, G. Laurenti, A. Lebedev, C. Lechanoine-Leluc, M.W. Lee, S.C. Lee, G. Levi, P. Levtchenko, C.L. Liu, H.T. Liu, I. Lopes, G. Lu, Y.S. Lu, K. Lübelsmeyer, D. Luckey, W. Lustermann, C. Maña, A. Margotti, F. Mayet, R.R. McNeil, B. Meillon, M. Menichelli, A. Mihul, A. Mourao, A. Mujunen, F. Palmonari, A. Papi, H.B. Park, W.H. Park, M. Pauluzzi, F. Pauss, E. Perrin, A. Pesci, A. Pevsner, M. Pimenta, V. Plyaskin, V. Pojidaev, M. Pohl, V. Postolache, N. Produit, P.G. Rancoita, D. Rapin, F. Raupach, D. Ren, Z. Ren, M. Ribordy, J.P. Richeux, E. Riihonen, J. Ritakari, S. Ro, U. Roeser, C. Rossin, R. Sagdeev, D. Santos, G. Sartorelli, C. Sbarra, S. Schael, A. Schultz von Dratzig, G. Schwering, G. Scolieri, E.S. Seo, J.W. Shin, V. Shoutko, E. Shoumilov, R. Siedling, D. Son, T. Song, M. Steuer, G.S. Sun, H. Suter, X.W. Tang, Samuel C.C. Ting, S.M. Ting, M. Tornikoski, J. Torsti, J. Trüm-

per, J. Ulbricht, S. Urpo, E. Valtonen, J. Vandenhirtz, F. Velcea, E. Velikhov, B. Verlaat, I. Vetlitsky, F. Vezzu, J.P. Vialle, G. Viertel, D. Vité, H.Von Gunten, S.Waldmeier Wicki, W. Wallraff, B.C. Wang, J.Z. Wang, Y.H. Wang, K. Wiik, C. Williams, S.X. Wu, P.C. Xia, J.L. Yan, L.G. Yan, C.G. Yang, J. Yang, M. Yang, S.W. Ye, P. Yeh, Z.Z. Xu, H.Y. Zhang, Z.P. Zhang, D.X. Zhao, G.Y. Zhu, W.Z. Zhu, H.L. Zhuang, A. Zichichi, B. Zimmermann, and P. Zuccon. The Alpha Magnetic Spectrometer (AMS) on the International Space Station: Part I – results from the test flight on the space shuttle. *Physics Reports*, 366(6):331–405, aug 2002. ISSN 03701573. doi: 10.1016/S0370-1573(02)00013-3. URL http://linkinghub.elsevier.com/retrieve/pii/S0370157302000133.

A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28 (5):749–771, may 2002. ISSN 01678191. doi: 10.1016/S0167-8191(02)00094-7. URL http://linkinghub.elsevier.com/retrieve/pii/S0167819102000947.

Bill Allcock, Ann Chervenak, Ian Foster, Carl Kesselman, and Miron Livny. Data Grid tools: enabling science on big distributed data. *Journal of Physics: Conference Series*, 16:571–575, jan 2005. ISSN 1742-6588. doi: 10.1088/1742-6596/16/1/079. URL http://stacks.iop.org/1742-6596/16/i=1/a=079?key=crossref.db6875f09f99143fca5f76589bd4e155.

N. S. Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175–185, aug 1992. ISSN 0003-1305. doi: 10.1080/00031305.1992.10475879. URL http://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

Eric Anderson. Simple table-based modeling of storage devices. Technical report, HP Laboratories, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.9140{&}rep=rep1{&}type=pdf.

D. Anick, D. Mitra, and MM Sondhi. Stochastic theory of a data handling system with multiple sources. In *ICC'80; International Conference on Communications, Seattle, Wash., June 8-12, 1980, Conference Record. Volume 1.(A81-32276 14-32) New York, Institute of Electrical and Electronics Engineers, Inc., 1980, p. 13.1. 1-13.1. 5.*, 1980.

Apache Software Foundation. Apache HTTP server, 2015.

InfiniBand Trade Association. *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.

ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation*, 3(08):S08003, 2008. doi: 10.1088/1748-0221/3/08/S08003. URL http://dx.doi.org/10.1088/1748-0221/3/08/S08003.

Ed Austin. The anatomy of the Google architecture, 2009. URL http://www.slideshare.net/hasanveldstra/the-anatomy-of-the-google-architecture-fina-lv11.

Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Garth R. Goodson, and Bianca Schroeder. An analysis of data corruption in the storage stack. *ACM Transactions on Storage*, 4(3):1–28, nov 2008. ISSN 15533077. doi: 10.1145/1416944.1416947. URL http://portal.acm.org/citation.cfm?doid=1416944.1416947.

Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Pearson, 2009.

M Barisits, C Serfon, V Garonne, M Lassnig, G Stewart, T Beermann, R Vigne, L Goossens, A Nairz, A Molfetas, and on behalf of the ATLAS Collaboration. ATLAS Replica Manage-

ment in Rucio: Replication Rules and Subscriptions. *Journal of Physics: Conference Series*, 513(4):042003, 2014. ISSN 1742-6588. doi: 10.1088/1742-6596/513/4/042003.

M Barisits, C Serfon, V Garonne, M Lassnig, T Beermann, and R Vigne. Resource control in ATLAS distributed data management: Rucio Accounting and Quotas. *Journal of Physics: Conference Series*, 664(6):062002, dec 2015. ISSN 1742-6588. doi: 10.1088/1742-6596/664/6/062002. URL http://stacks.iop.org/1742-6596/664/i=6/a=062002?key=crossref.fb68f4788ce19b8e88060807982f1ff3.

Martin Barisits, Vincent Garonne, Mario Lassnig, and Angelos Molfetas. Simulating the ATLAS Distributed Data Management System. *Journal of Physics*, 396(5):052009, dec 2012. ISSN 1742-6588. doi: 10.1088/1742-6596/396/5/052009. URL http://stacks.iop.org/1742-6596/396/i=5/a=052009?key=crossref.ac0d9f9ba9e3b229e25af7e255fbd365.

Martin Barisits, Eva Kühn, and Mario Lassnig. A Hybrid Simulation Model for Data Grids. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 255–260. IEEE, may 2016a. ISBN 978-1-5090-2453-7. doi: 10.1109/CCGrid.2016.36. URL http://ieeexplore.ieee.org/document/7515695/.

Martin-Stefan Barisits, Cedric Serfon, Vincent Garonne, Mario Lassnig, and Thomas Beermann. Automatic rebalancing of data in ATLAS distributed data management. *Journal of Physics: Conference Series*, to appear(Sep), 2016b.

Rakesh Barve, Elizabeth Shriver, Phillip B. Gibbons, Bruce K. Hillyer, Yossi Matias, and Jeffrey Scott Vitter. Modeling and optimizing I/O throughput of multiple disks on a bus. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '99*, pages 83–92, New York, New York, USA, 1999. ACM Press. ISBN 158113083X. doi: 10.1145/301453.301482. URL http://dl.acm.org/citation.cfm?id=301482http://portal.acm.org/citation.cfm?doid=301453.301482.

Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *2007 IEEE Global Internet Symposium*, pages 79–84. IEEE, may 2007. ISBN 978-1-4244-1697-4. doi: 10.1109/GI.2007.4301435. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4301435.

T Beermann, P Maettig, G Stewart, M Lassnig, V Garonne, M Barisits, R Vigne, C Serfon, L Goossens, A Nairz, and A Molfetas. Popularity Prediction Tool for ATLAS Distributed Data Management. *Journal of Physics: Conference Series*, 513(4):042004, jun 2014. ISSN 1742-6588. doi: 10.1088/1742-6596/513/4/042004. URL http://stacks.iop.org/1742-6596/513/i=4/a=042004?key=crossref.7a05efa401f11023d121d976039984d0.

Thomas Beermann, Mario Lassnig, Martin-Stefan Barisits, Cedric Serfon, and Vincent Garonne. C3PO - A Dynamic Data Placement Agent for ATLAS Distributed Data Management. *Journal of Physics: Conference Series*, to appear(Sep), 2016.

William H. Bell, David G. Cameron, A. P. Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, nov 2003. ISSN 00000000. doi: 10.1177/10943420030174005. URL http://hpc.sagepub.com/cgi/doi/10.1177/10943420030174005.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5), 2007. doi: 10.1.1.72.4580.

Ian Bird, Kors Bos, N Brook, D Duellmann, C Eck, I Fisk, D Foster, B Gibbard, M Girone, and C Grandi. LHC computing Grid. *EGEE, Technical design report CERN-LHCC-2005-024*, 2008.

Miguel Branco, Ed Zaluska, David De Roure, Mario Lassnig, and Vincent Garonne. Managing very large distributed data sets on a data grid. *Concurrency and Computation: Practice*

and Experience, 22(August 2009):1338–1364, 2010. URL http://onlinelibrary.wiley.com/doi/10.1002/cpe.1489/full.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Martin Dietrich Buhmann. *Radial basis functions*, volume 9. Cambridge Univ Press, 2000.

Rajkumar Buyya and Manzur Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, nov 2002. ISSN 1532-0626. doi: 10.1002/cpe.710. URL http://doi.wiley.com/10.1002/cpe.710.

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41 (1):23–50, jan 2011. ISSN 00380644. doi: 10.1002/spe.995. URL http://doi.wiley.com/10.1002/spe.995.

H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 430–437. IEEE Comput. Soc, 2001. ISBN 0-7695-1010-8. doi: 10.1109/CCGRID.2001.923223. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=923223.

Henri Casanova and Loris Marchal. A network model for simulation of grid application. Technical report, INRIA, 2002.

Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: A generic framework for large-scale distributed experiments. In *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, pages 126–131, 2008. ISBN 0769531148. doi: 10.1109/UKSIM.2008.28.

Henri Casanova, Arnaud Giersch, Arnaud Legrand, and Martin Quinson. Versatile , Scalable , and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

CERN. CERN Advanced Storage Manager, 2016. URL https://castor.web.cern.ch.

Chih-Chung Chang and Chih-Jen Lin. Training v -Support Vector Classifiers: Theory and Algorithms. *Neural Computation*, 13(9):2119–2147, sep 2001. ISSN 0899-7667. doi: 10.1162/089976601750399335. URL http://www.mitpressjournals.org/doi/abs/10.1162/089976601750399335.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable. *ACM Transactions on Computer Systems*, 26(2):1–26, jun 2008. ISSN 07342071. doi: 10.1145/1365815.1365816. URL http://portal.acm.org/citation.cfm?doid=1365815.1365816.

Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and Social Network of YouTube Videos. In *2008 16th Interntional Workshop on Quality of Service*, pages 229–238. IEEE, jun 2008. ISBN 978-1-4244-2084-1. doi: 10.1109/IWQOS.2008.32. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4539688.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, mar 2002. ISSN 10459227. doi: 10.1109/72.991427. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=991427.

Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab. *ACM SIGCOMM Computer Communication Review*, 33(3):3, jul 2003. ISSN 01464833. doi: 10.1145/956993.956995. URL http://portal.acm.org/citation.cfm?doid=956993.956995.

James Cowie, Hongbo Liu, Jason Liu, David Nicol, and Andy Ogielski. Towards Realistic Million-Node Internet Simulations. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

Dah Ming Chiu. Some observations on fairness of bandwidth sharing. In *Proceedings ISCC 2000. Fifth IEEE Symposium on Computers and Communications*, pages 125–131. IEEE Comput. Soc, 2000. ISBN 0-7695-0722-0. doi: 10.1109/ISCC.2000.860626. URL http://ieeexplore.ieee.org/document/860626/.

Li Deng. Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(3-4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL http://nowpublishers.com/articles/foundations-and-trends-in-signal-processing/SIG-039.

John S Denker and Sara A Solla. Optimal brain damage. In *NIPs*, pages 598—-605, 1989.

Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. XROOTD-A Highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(1), 2006. doi: 10.1.1.127.9281.

Dropbox Inc. Dropbox, 2015. URL http://www.dropbox.com.

Facebook Inc. Facbook, 2015. URL http://www.facebook.com.

Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

John Fletcher. An Arithmetic Checksum for Serial Transmissions. *IEEE/ACM Transactions on Networking*, 3(6):640–651, 1982. ISSN 10636692. doi: 10.1109/90.477710. URL http://ieeexplore.ieee.org/document/477710/.

S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999. ISSN 10636692. doi: 10.1109/90.793002. URL http://ieeexplore.ieee.org/document/793002/.

I Foster, C Kesselman, and S Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 2001. URL http://hpc.sagepub.com/content/15/3/200.short.

Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, jul 2006. ISSN 1000-9000. doi: 10.1007/s11390-006-0513-y. URL http://link.springer.com/10.1007/s11390-006-0513-y.

Patrick Fuhrmann and Volker Gülzow. dCache, Storage System for the Future. In *12th International Euro-Par Conference*, pages 1106–1113, 2006. doi: 10.1007/11823285_116. URL http://link.springer.com/10.1007/11823285{_}116.

R.M. Fujimoto, K. Perumalla, A. Park, H. Wu, M.H. Ammar, and G.F. Riley. Large-scale network simulation: how big? how fast? In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.*, pages 116–123. IEEE Comput. Soc, 2003. ISBN 0-7695-2039-1. doi: 10.1109/MASCOT.2003.1240649. URL http://ieeexplore.ieee.org/document/1240649/.

Kayo Fujiwara and Henri Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework Categories and Subject Descriptors. In *Performance evaluation methodologies and tools*, 2007. ISBN 9789639799004.

John S. Bucy Ganger, Jiri Schindler, Steven W. Schlosser, and Gregory R. The DiskSim Simulation Environment Version 4.0 Reference Manual. Technical report, Technical report, Carnegie Mellon University/University of Michigan, 2008.

V Garonne, M Lassnig, M Barisits, T Beermann, R Vigne, and C Serfon. The ATLAS Distributed Data Management System & Databases. In *XLDB Workshop Europe*, 2013.

V Garonne, R Vigne, G Stewart, M Barisits, T B Eermann, M Lassnig, C Serfon, L Goossens, A Nairz, and on behalf of the ATLAS Collaboration. Rucio - The next generation of large scale

distributed system for ATLAS Data Management. *Journal of Physics: Conference Series*, 513:042021, 2014. ISSN 1742-6588. doi: 10.1088/1742-6596/513/4/042021.

Vincent Garonne, Graeme A Stewart, Mario Lassnig, Angelos Molfetas, Martin Barisits, Thomas Beermann, Armin Nairz, Luc Goossens, Fernando Barreiro Megino, Cedric Serfon, Danila Oleynik, and Artem Petrosyan. The ATLAS Distributed Data Management project: Past and Future. *Journal of Physics: Conference Series*, 396(3):032045, dec 2012. ISSN 1742-6588. doi: 10.1088/1742-6596/396/3/032045. URL http://stacks.iop.org/1742-6596/396/i=3/a=032045?key=crossref.e5360c537eeee81fc83e6bd0964775fe.

Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, jun 2003. ISSN 01635808. doi: 10.1145/776985.776986. URL http://portal.acm.org/citation.cfm?doid=776985.776986.

Google Inc. Youtube, 2015. URL http://www.youtube.com.

Graphite Collaboration. Graphite - Scalable Realtime Graphing, 2016.

Chetan Gupta, Abhay Mehta, and Umeshwar Dayal. PQR: Predicting Query Execution Times for Autonomous Workload Management. In *2008 International Conference on Autonomic Computing*, pages 13–22. IEEE, jun 2008. doi: 10.1109/ICAC.2008.12. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4550823.

Suraj G Gupta, Mangesh M Ghonge, Parag D Thakare, and P M Jawandhiya. Open-source network simulation tools: An overview. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 2(4):pp—-1629, 2013.

E.L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in Communications*, 9(7):1024–1039, 1991. ISSN 07338716. doi: 10.1109/49.103550. URL http://ieeexplore.ieee.org/document/103550/.

M.A. Hearst, S.T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, jul 1998. ISSN 1094-7167. doi: 10.1109/5254.708428. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=708428.

Thomas R. Henderson, Mathieu Lacage, and George F. Riley. Network simulations with the ns-3 simulator, 2008.

M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, apr 2011. ISSN 0036-8075. doi: 10.1126/science.1200970. URL http://www.sciencemag.org/cgi/doi/10.1126/science.1200970.

Bruce K. Hillyer and Avi Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):170–179, may 1996. ISSN 01635999. doi: 10.1145/233008.233039. URL http://portal.acm.org/citation.cfm?doid=233008.233039.

Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H.J. Epema. The Grid Workloads Archive. *Future Generation Computer Systems*, 24 (7):672–686, jul 2008. ISSN 0167739X. doi: 10.1016/j.future.2008.02.003. URL http://linkinghub.elsevier.com/retrieve/pii/S0167739X08000125.

D. Kaeli. Execution-driven simulation of network storage systems. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings.*, pages 604–611. IEEE, 2004. ISBN 0-7695-2251-3. doi: 10.1109/MASCOT.2004.1348318. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1348318.

Jeff Kantor, T Axelrod, K-T Lim, M Freemon, J Becla, and M Juric. Large Synoptic Survey Telescope (LSST) Data Management System Design. Technical report, Large Synoptic Survey Telescope, 2013.

Keshav and Srinivasan. *REAL: A network simulator*. University of California, 1988.

Andrey Kiryanov, Alejandro Alvarez Ayllon, and Oliver Keeble. FTS3 / WebFTS – A Powerful File Transfer Service for Scientific Communities. *Procedia Computer Science*, 66:670–678, 2015. ISSN 18770509. doi: 10.1016/j.procs.2015.11.076. URL http://linkinghub.elsevier.com/retrieve/pii/S1877050915034250.

Wolfgang Kreutzer, Jane Hopkins, and Marcel van Mierlo. SimJAVA—a framework for modeling queueing networks in Java. In *Proceedings of the 29th conference on Winter simulation - WSC '97*, pages 483–488, New York, New York, USA, 1997. ACM Press. ISBN 078034278X. doi: 10.1145/268437.268548. URL http://portal.acm.org/citation.cfm?doid=268437.268548.

M Lassnig, R Vigne, T Beermann, M Barisits, V Garonne, and C Serfon. Scalable and fail-safe deployment of the ATLAS Distributed Data Management system Rucio. *Journal of Physics: Conference Series*, 664(6):062027, dec 2015. ISSN 1742-6588. doi: 10.1088/1742-6596/664/6/062027. URL http://stacks.iop.org/1742-6596/664/i=6/a=062027?key=crossref.3fbda6461e5e4f1ea0c5239e77648e1b.

Mario Lassnig. *Dissertation Workload modelling for data-intensive systems*. PhD thesis, University of Innsbruck, 2014.

Mario Lassnig and Thomas Fahringer. Stream monitoring in large-scale distributed concealed environments. *IEEE e-Science*, 2009. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=5380871.

Mario Lassnig, Thomas Fahringer, Vincent Garonne, Angelos Molfetas, and Martin Barisits. A similarity measure for time, frequency, and dependencies in large-scale workloads. *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2011. doi: 10.1145/2063384.2063441.

Latif, Aamer, Mullendore, Rodney N, White, Joseph L, Uchino, and Brian Y. *Method and apparatus for transferring data between IP network devices and SCSI and fibre channel devices over an IP network*. Google Patents, 2002.

A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill New York, 1991.

Adrien Lebre, Arnaud Legrand, Frederic Suter, and Pierre Veyre. Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 251–260. IEEE, may 2015. ISBN 978-1-4799-8006-2. doi: 10.1109/CCGrid.2015.134. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7152491.

Benyuan Liu, DR Figueiredo, Yang Guo, J. Kurose, and D. Towsley. A study of networks simulation efficiency: fluid simulation vs. packet-level simulation. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1244–1253. IEEE, 2001. ISBN 0-7803-7016-3. doi: 10.1109/INFCOM.2001.916619. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=916619http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=916619.

S.H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions on Networking*, 11(4):525–536, aug 2003. ISSN 1063-6692. doi: 10.1109/TNET.2003.815297. URL http://ieeexplore.ieee.org/document/1224453/.

T Maeno. PanDA: distributed production and distributed analysis system for ATLAS. *Journal of Physics: Conference Series*, 119(6):062036, jul 2008. ISSN 1742-6596. doi: 10.1088/1742-6596/119/6/062036. URL http://stacks.iop.org/1742-6596/119/i=6/a=062036?key=crossref.6ac104c4fcd9216de8543e3f489eabca.

L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, pages 1395–1403 vol.3. IEEE, 1999. ISBN 0-7803-5417-6. doi:

10.1109/INFCOM.1999.752159. URL http://ieeexplore.ieee.org/document/752159/.

M Mezard and Jean-P Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of Physics A: Mathematical and General*, 22(12):2191–2203, jun 1989. ISSN 0305-4470. doi: 10.1088/0305-4470/22/12/019. URL http://stacks.iop.org/0305-4470/22/i=12/a=019?key=crossref.f77094e01bd3275078237195c8d35396.

Angelos Molfetas, Mario Lassnig, Vincent Garonne, Graeme Stewart, Martin Barisits, Thomas Beermann, and Gancho Dimitrov. Popularity framework for monitoring user workload. *Journal of Physics: Conference Series*, 396(5):052055, dec 2012. ISSN 1742-6588. doi: 10.1088/1742-6596/396/5/052055. URL http://stacks.iop.org/1742-6596/396/i=5/a=052055?key=crossref.3cc39540b47c1fb2fbb4f41a37f172f3.

Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. The Grid as a Single Entity: Towards a Behavior Model of the Whole Grid. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 886–897. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-88871-0_62. URL http://link.springer.com/10.1007/978-3-540-88871-0{_}62.

Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. Finding order in chaos: a behavior model of the whole grid. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2009. ISSN 15320626. doi: 10.1002/cpe.1490. URL http://doi.wiley.com/10.1002/cpe.1490.

Jesús Montes, Alberto Sánchez, and María S. Pérez. Grid Global Behavior Prediction. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 124–133. IEEE, may 2011. ISBN 978-1-4577-0129-0. doi: 10.1109/CCGrid.2011.17. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5948603.

Alberto Montresor and Mark Jelasity. PeerSim: A scalable P2P simulator. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 99–100. IEEE, sep 2009. ISBN 978-1-4244-5066-4. doi: 10.1109/P2P.2009.5284506. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5284506.

Randall Munroe. Google's Datacenters on Punch Cards, 2015. URL http://what-if.xkcd.com/63/.

Jerzy Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558—-625, 1934.

David Nicol, Michael Goldsby, and Michael Johnson. Fluid-based Simulation of Communication Networks using SSF Fluid Simulation in SSF. In *European Simulation Symposium*, 1999.

Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic Cloud provisioning for scientific Grid workflows. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 97–104. IEEE, oct 2010. ISBN 978-1-4244-9347-0. doi: 10.1109/GRID.2010.5697953. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5697953.

Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. *Euro-Par 2010 Parallel Processing Workshops*, 6586:305–313, 2011. doi: 10.1007/978-3-642-21878-1_38. URL http://link.springer.com/10.1007/978-3-642-21878-1{_}38.

Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon S3 for science grids. In *Proceedings of the 2008 international workshop on Data-aware distributed computing - DADC '08*, pages 55–64, New York, New York, USA, 2008. ACM Press. ISBN 9781605581545. doi: 10.1145/1383519.1383526. URL http://portal.acm.org/citation.cfm?doid=1383519.1383526.

Phil Simon. *Too Big to Ignore: The Business Case for Big Data*. John Wiley \& Sons, 2013.

Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, nov 1987. ISSN 0031-9007. doi: 10.1103/PhysRevLett.59.2229. URL http://link.aps.org/doi/10.1103/PhysRevLett.59.2229.

Python Software Foundation. Python 2.6 programming language. *URL https://docs.python.org/2.6/ Accessed on the 23 of April 2015*, 2015.

J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 08856125. doi: 10.1023/A:1022643204877. URL http://link.springer.com/10.1023/A:1022643204877.

Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

George F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research - MoMeTools '03*, number August in MoMeTools '03, page 5, New York, New York, USA, 2003. ACM Press. ISBN 1581137488. doi: 10.1145/944773.944775. URL http://dl.acm.org/citation.cfm?id=944775http://portal.acm.org/citation.cfm?doid=944773.944775.

R. Rivest. The MD5 Message-Digest Algorithm. Technical report, apr 1992. URL https://www.rfc-editor.org/info/rfc1321.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 0033-295X. doi: 10.1037/h0042519. URL http://content.apa.org/journals/rev/65/6/386.

Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 1994. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=268881.

Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, 2003.

Erich Schikuta, Thomas Fuerle, and Helmut Wanek. ViPIOS: The Vienna Parallel Input/Output System. In *Lecture Notes in Computer Science*, pages 953–958. Springer Berlin Heidelberg, 1998. doi: 10.1007/BFb0057953. URL http://link.springer.com/10.1007/BFb0057953.

Hans Schwefel, Manfred Jobmann, Daniel Höllisch, and Daniel Heyman. On the accuracy of TCP performance models. Technical report, International Society for Optics and Photonics, 2003.

G. A. F. Seber and C. J. Wild. *Nonlinear Regression*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, feb 1989. ISBN 9780471725312. doi: 10.1002/0471725315. URL http://doi.wiley.com/10.1002/0471725315.

C. Serfon, M. Barisits, T. Beermann, V. Garonne, L. Goossens, M. Lassnig, A. Nairz, and R. Vigne. Rucio, the next-generation Data Management system in ATLAS. *Nuclear and Particle Physics Proceedings*, 273-275:969–975, apr 2014. ISSN 24056014. doi: 10.1016/j.nuclphysbps.2015.09.151. URL http://linkinghub.elsevier.com/retrieve/pii/S2405601415006409.

John F. Shoch. An introduction to the ethernet specification. *ACM SIGCOMM Computer Communication Review*, 11(3):17–19, jul 1981. ISSN 01464833. doi: 10.1145/1015591.1015593. URL http://portal.acm.org/citation.cfm?doid=1015591.1015593.

Reza Simeonidou, Dimitra Nejabati, B St Arnaud, M Beck, P Clarke, DB Hoang, D Hutchison, T Karmous-Edwards, G Lavian, and J Leigh. Optical network infrastructure for grid. *Grid Forum Draft, GFD-I*, 2004.

Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016, sep 2004. ISSN 07437315. doi: 10.1016/j.jpdc.2004.06.008. URL http://linkinghub.elsevier.com/retrieve/pii/S0743731504000991.

Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155—-161, 1997. doi: 10.1.1.10.4845.

Thomas Lawrence Sterling. *Beowulf cluster computing with Linux*. MIT press, 2002.

Graeme A Stewart, Vincent Garonne, Mario Lassnig, Angelos Molfetas, Martin Barisits, Donal Zhang, Ivan Calvet, Thomas Beermann, Fernando Barreiro Megino, Andrii Tykhonov, Simone Campana, Cedric Serfon, Danila Oleynik, and Artem Petrosyan. Advances in service and operations for ATLAS data management. *Journal of Physics: Conference Series*, 368:012005, jun 2012. ISSN 1742-6596. doi: 10.1088/1742-6596/368/1/012005. URL http://stacks.iop.org/1742-6596/368/i=1/a=012005?key=crossref.79d6eed29f1b8e41f3a56555864474e8.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '01*, pages 149–160, New York, New York, USA, 2001. ACM Press. ISBN 1581134118. doi: 10.1145/383059.383071. URL http://portal.acm.org/citation.cfm?doid=383059.383071.

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*, 20:1591—-1609, 2008.

Willy Tarreau. HAProxy, 2015.

Twitter Inc. Twitter, 2015. URL http://www.twitter.com.

United States Government. National Centers for Enviornmental Information, 2015. URL https://www.ncei.noaa.gov/about.

M. Uysal, G.a. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 183–192. IEEE Comput. Soc, 2001. ISBN 0-7695-1315-8. doi: 10.1109/MASCOT.2001.948868. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=948868.

Andras Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European simulation multiconference (ESM'2001)*, page 65, 2001.

Nedeljko Vasić, Martin Barisits, Vincent Salzgeber, and Dejan Kostic. Making cluster applications energy-aware. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09*, page 37, New York, New York, USA, 2009. ACM Press. ISBN 9781605585857. doi: 10.1145/1555271.1555281. URL http://portal.acm.org/citation.cfm?doid=1555271.1555281.

Pedro Velho and Arnaud Legrand. Accuracy study and improvement of network simulation in the SimGrid framework. In *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*. ICST, 2009. ISBN 978-963-9799-45-5. doi: 10.4108/ICST.SIMUTOOLS2009.5592. URL http://eudl.eu/doi/10.4108/ICST.SIMUTOOLS2009.5592.

R Vigne, E Schikuta, V Garonne, G Stewart, M Barisits, T Beermann, M Lassnig, C Serfon, L Goossens, and A Nairz. DDM Workload Emulation. *Journal of Physics: Conference Series*, 513(4):042048, jun 2013. ISSN 1742-6588. doi: 10.1088/1742-6596/513/4/042048. URL http://stacks.iop.org/1742-6596/513/i=4/a=042048?key=crossref.1f7bc644bf462701ba20d101dbaddf18.

Elisabeth Vinek, Peter Paul Beran, and Erich Schikuta. Classification and Composition of QoS Attributes in Distributed, Heterogeneous Systems. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 424–433. IEEE, may 2011a. ISBN 978-1-4577-0129-0. doi: 10.1109/CCGrid.2011.53. URL http://ieeexplore.ieee.org/document/5948633/.

Elisabeth Vinek, Peter Paul Beran, and Erich Schikuta. A Dynamic Multi-Objective Optimization Framework for Selecting Distributed Deployments in a Heterogeneous Environment. *Procedia Computer Science*, 4:166–175, 2011b. ISSN 18770509. doi: 10.1016/j.procs.2011.04.018. URL http://linkinghub.elsevier.com/retrieve/pii/S1877050911000767.

Vital Wave consulting. Big Data , Big Impact : New Possibilities for International Development. Technical report, World Economic Forum, 2012.

Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with CART models. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):412, jun 2004. ISSN 01635999. doi: 10.1145/1012888.1005743. URL http://portal.acm.org/citation.cfm?doid=1012888.1005743.

Wentao Wu, Yun Chi, Shenghuo Zhu, J. Tatemura, H. Hacigumus, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1081–1092. IEEE, apr 2013. ISBN 978-1-4673-4910-9. doi: 10.1109/ICDE.2013.6544899. URL http://ieeexplore.ieee.org/document/6544899/.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. ISSN 00189219. doi: 10.1109/5.58337. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58337.

Wentao Wu, Yun Chi, Hakan Hacígümüş, and Jeffrey F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *Proceedings of the VLDB Endowment*, 6(10):925–936, aug 2013. ISSN 21508097. doi: 10.14778/2536206.2536219. URL http://dl.acm.org/citation.cfm?doid=2536206.2536219.

Martin-Stefan Barisits

Contact

Färbermühlgasse 13/4/11
1230 Vienna
Austria

martin@barisits.at

Education

2010–2017	*Doctor* of Technical Sciences Expected date of defense: April, 2017 Vienna University of Technology, Austria
2007–2010	*Diplom-Ingenieur* in Computational Intelligence Vienna University of Technology, Austria
2003–2007	*Bachelor of Science* in Medical Computer Science Vienna University of Technology, Austria

Employment

07/2014–now	Senior Fellow CERN, Switzerland
08/2013–06/2014	Project Associate CERN, Switzerland
08/2010–07/2013	Doctoral Student CERN, Switzerland

Grants, Awards & Honors

2014	CERN COFUND Fellowship Marie Skłodowska-Curie action under the European Commission's Horizon 2020 Programme
2011	Travel Grant, IFIP Performance International Federation for Information Processing (IFIP)
2010	Full 3-Year Ph.D. Grant for the CERN Doctoral Program Austrian Ministry of Science

Editorial experience

2016	14th International conference on Software engineering and Formal methods (SEFM) Member of the organization team
2015	21st International conference on Computing in High Energy and Nuclear Physics (CHEP) Review of publications in the Computing Activities track

Teaching Experience

2011–now	Supervising CERN summer students Planning and supervision of projects of the CERN summer student program. Students are staying for 6-12 weeks at CERN, working full-time on their designated projects CERN, Switzerland
2009	Teaching Assistant *Distributed Programming with Space Based Computing* Planning, supervision and correction of laboratory part Institute of Computer Languages Vienna University of Technology, Austria

Languages

German	Native language
English	Fluent
French	Basic understanding

Peer-reviewed Publications

1. **Automatic rebalancing of data in ATLAS distributed data management**, 2016
   Martin Barisits, Cedric Serfon, Vincent Garonne, Mario Lassnig, Thomas Beermann
   22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP), San Francisco, USA

2. **C3PO - A Dynamic Data Placement Agent for ATLAS Distributed Data Management**, 2016
   Thomas Beermann, Mario Lassnig, Martin Barisits, Cedric Serfon, Vincent Garonne
   Journal of Physics, IOP Publishing Ltd

3. **A hybrid simulation model for data grids**, 2016
   M Barisits, E Kuehn, M Lassnig
   16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia

4. **Monitoring and controlling ATLAS data management: The Rucio web user interface**, 2015
   M Lassnig, T Beermann, R Vigne, M Barisits, V Garonne and C Serfon
   21st International Conference on Computing in High Energy and Nuclear Physics (CHEP), Okinawa, Japan

5. **Scalable and fail-safe deployment of the ATLAS Distributed Data Management system Rucio**, 2015
   M Lassnig, R Vigne, T Beermann, M Barisits, V Garonne and C Serfon
   21st International Conference on Computing in High Energy and Nuclear Physics (CHEP), Okinawa, Japan

6. **Resource control in ATLAS distributed data management**, 2015
   M Barisits, C Serfon, V Garonne, M Lassnig, T Beermann, R Vigne
   21st International Conference on Computing in High Energy and Nuclear Physics (CHEP), Okinawa, Japan

7. **Rucio, the next-generation Data Management system in ATLAS**, 2014
   V Garonne, R Vigne, G Stewart, M Barisits, T B eermann, M Lassnig, C Serfon, L Goossens, A Nairz
   37th International Conference on High Energy Physics (ICHEP), Valencia, Spain

8. **Popularity Prediction Tool for ATLAS Distributed Data Management**, 2014
   T Beermann, P Maettig, G Stewart, M Lassnig, V Garonne, M Barisits, R Vigne, C Serfon, L Goossens, A Nairz
   11th International Symposium on Grids and Clouds (ISGC), Taipei, Taiwan

9. **ATLAS Replica Management in Rucio: Replication Rules and Subscriptions**, 2014
   Martin Barisits, Vincent Garonne, Mario Lassnig, Graeme Stewart, Thomas Beermann, Ralph Vigne, Cedric Serfon, Luc Goossens, Armin Nairz, Angelos Molfetas
   Journal of Physics, IOP Publishing

10. **The ATLAS data management software engineering process**, 2014
    M Lassnig, V Garonne, G A Stewart, M Barisits, T Beermann, R Vigne, C Serfon, L Goossens, A Nairz, A Molfetas
    20th International Conference on Computing in High Energy and Nuclear Physics (CHEP), Amsterdam, Netherlands

11. **ATLAS DQ2 to Rucio renaming infrastructure**, 2014
    C Serfon, M Barisits, T Beermann, V Garonne, L Goossens, M Lassnig, A Molfetas, A Nairz, G Stewart, R Vigne

20th International Conference on Computing in High Energy and Nuclear Physics (CHEP), Amsterdam, Netherlands

12. **Advances in service and operations for ATLAS data management**, 2014
    G A Stewart, V Garonne, M Lassnig, A Molfetas, M Barisits, D Zang, I Calvet, T Beermann, F Barreiro Megino, A Tykhonov, S Campana, C Serfon, D Oleynik, A Petrosyan
    14th International Workshop On Advanced Computing And Analysis Techniques In Physics Research (ACAT), London, UK

13. **The ATLAS Distributed Data Management System & Databases**, 2013
    V Garonne, M Lassnig, M Barisits, T Beermann, R Vigne, C Serfon
    Extremely Large Database Europe workshop (XLDB), Geneva, Switzerland

14. **The ATLAS Distributed Data Management project: Past and Future**, 2012
    Vincent Garonne, Graeme A. Stewart, Mario Lassnig, Angelos Molfetas, Martin Barisits, Thomas Beermann, Armin Nairz, Luc Goossens, Fernando Barreiro Megino, Cedric Serfon, Danila Oleynik, Artem Petrosyan
    Journal of Physics, IOP Publishing Ltd

15. **Simulating the ATLAS Distributed Data Management System**, 2012
    Martin Barisits, Vincent Garonne, Mario Lassnig, Angelos Molfetas
    19th International conference on Computing in High Energy and Nuclear Physics (CHEP), New York, USA

16. **The ATLAS DDM Tracer monitoring framework**, 2012
    Donal Zang, Vincent Garonne, Martin Barisits, Mario Lassnig, Graeme A. Stewart, Angelos Molfetas, Thomas Beermann
    19th International Conference on Computing in High Energy and Nuclear Physics (CHEP), New York, USA

17. **Popularity framework for monitoring user workload**, 2012
    A Molfetas, M Lassnig, V Garonne, G A Stewart, M Barisits, T Beermann, G Dimitrov
    19th International Conference on Computing in High Energy and Nuclear Physics (CHEP), New York, USA

18. **A Similarity Measure for Time, Frequency, and Dependencies in Large-Scale Workloads**, 2011
    Mario Lassnig, Thomas Fahringer, Vincent Garonne, Angelos Molfetas, Martin Barisits
    24th ACM SIGARCH / IEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC), Seattle, USA

19. **Popularity framework to process dataset tracers and its application on dynamic replica reduction in the ATLAS experiment**, 2010
    Molfetas A., Megino F., Tykhonov A., Garonne V., Campana S., Lassnig

M., Barisits M., Dimitrov G., Viegas F.
Journal of Physics - Conference Series, IOP Publishing Ltd

20. **Making Cluster Applications Energy-Aware**, 2009
Nedeljko Vasić, Martin Barisits, Vincent Salzgeber and Dejan Kostić
Workshop on Automated Control for Datacenters and Clouds, 6th IEEE/ACM International Conference on Autonomic Computing and Communications (ICAC), Barcelona, Spain

I also have co-authorship of 83 physics publications as a member of the ATLAS collaboration.

Geneva, March 6, 2017