High-Performance Persistent Identification for Research Data Management

Dissertation

zur Erlangung des Doktorgrades
Dr. rer. nat.
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Fatih Berber
aus Emmendingen

Göttingen, 2018

Betreuungsausschuss:

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung
mbH Göttingen (GWDG),
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski
Institut für Informatik
Georg-August-Universität Göttingen

Prüfungskommission:

Referent:

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung
mbH Göttingen (GWDG),
Institut für Informatik
Georg-August-Universität Göttingen

Korreferenten:

Prof. Dr. Dieter Hogrefe
Institut für Informatik
Georg-August-Universität Göttingen

Weitere Mitglieder
der Prüfungskommission:

Prof. Dr. Jens Grabowski
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Caroline Sporleder
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Xiaoming Fu
Institut für Informatik
Georg-August-Universität Göttingen

Jun.-Prof. Dr. Marcus Baum
Institut für Informatik
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 7. September 2018

Abstract

Durable identification and access to datasets, especially to research datasets, become increasingly important. This is mainly driven by the explosive dataset growth in the current age. Although the Internet was originally founded as a large-scale end-to-end communication platform, in the current era, it has developed to an information consumption medium with an overwhelming large spreading.

However, the conception of the Internet against its original purpose aggravates an efficient data consumption. This is particularly based on the address-based data access mechanism, in which data is only consumable through a specific locator. Since, data mobility therefore leads to changing locators, the concept of persistent identification has been developed to track these changes. Instead of addressing data directly through its current valid locator, *Persistent Identifier*s (PIDs) enable data retrieval by globally unique and durable identifiers. This in turn has led research datasets to be increasingly assigned with PIDs. With the advent of massive research dataset generation, also the load on PID systems has dramatically increased, which causes PID record management to constitute a considerable performance problem.

Therefore, this thesis focuses on the performance aspects behind PIDs. The goal is to provide solutions for high-performance PID management and resolution. Based on the established Handle System, we provide approaches which enable an accelerated usage of PIDs for research datasets, which are stored in sophisticated research data repositories.

Moreover, this thesis also provides contributions for the area of performance analysis based on the queuing networks. The basic approach is to model a PID system as a multi-tier transactional Internet system and to mathematically investigate improvements of the response time.

Acknowledgements

First, I would like to sincerely thank Prof. Dr. Ramin Yahyapour for his constant support and guidance throughout the course of this thesis. I am especially grateful for his fruitful impulses, which were an important source of inspiration.
I am also very thankful for the pleasant research atmosphere provided by the eScience group of GWDG. In particular my team lead Phillip Wieder, for establishing a sympathetic working environment. Also for my other colleagues Vanessa End, Sven Bingert, Tibor Kalman and Ulrich Schwardmann I would like to express my gratitude for their advice and support during the thesis.
Finally, I would like to express my extraordinary thanks to my family and friends for their moral and understanding.
A special thanks is due to my wife Yasemin for her understanding, her patience and especially her love.
Ultimately, I wish to thank my son Muhammed Ali and daughter Zeyneb who were the main inspiration and energy source behind this thesis. These were particularly evoked by our late night tours with the stroller.

Contents

List of Figures

List of Tables

Acronyms

RDM	*Research Data Management*
PID	*Persistent Identifier*
URL	*Uniform Resource Locator*
URI	*Uniform Resource Identifier*
URN	*Uniform Resource Name*
NDN	*Named Data Networking*
DNS	*Domain Name System*
ICN	*Information Centric Networking*
DOI	*Digital Object Identifier*
ARK	*Archival Resource Key*
THUMP	*HTTP URL Mapping Protocol*
EZID	*Easy-Eye-Dee*
URN	*Uniform Resource Name*
NBN	*National Bibliography Number*
DSID	*Research Data Silo Identifier*
GPR	*Global Proxy Resolver*
GHR	*Global Handle Registry*
LHS	*Local Handle Service*
MVA	*Mean Value Analysis*
CDN	*content distribution network*
TLD	*Top Level Domain*
ISP	*Internet Service Provider*
IANA	*Internet Assigned Numbers Authority*
IoT	*Internet of Things*

DDDS	*Dynamic Delegation Discovery System*
iRODS	*Integrated Rule-Oriented Data System*
RAP	*Repository Access Protocol*
RDF	*Resource Description Framework*
ARK	*Archival Resource Key*
IKB	*Interoperability Knowledge Base*
LQN	*Layered Queuing Network*

Chapter 1

Introduction

The digitalization of the world has heralded a new era, the *Digital Era*. This new era is characterized by unimaginably huge amounts of digitalized datasets which are generated and put into circulation. Likewise new technologies have emerged to ease the consumption of these massive amounts of information. While at the beginnings of the Internet it was used as a medium to exchange textual information, in the *Digital Era*, the Internet has become the central information distribution medium of our current civilization in which data of any shape and size is available. These developments have led to a tremendous increase in value of data. The World Economic Forum even compared data with oil [1], which is especially true for personal data.
As oil is controlled by a few companies, such a monopoly situation is also existing for personal data. Companies like LinkedIn, Twitter, Google and Facebook own incredible large amounts of personal data, which are sold to various other data broker companies. This is exactly why the services offered by these companies are free of charge for individual users. Hence, individual users make use of free services in exchange for personal data and often without the knowledge of the sheer dimensions of personal data. On the other side, data brokers like Acxiom or Cambridge Analytica are specialized in analyzing all these different datasets with the goal of understanding human behavior. The analysis is for example then used to subject specific user groups with advertisements and offers which are tailored for their specific user profiles. This also overs offers containing political content.

The new *Digital Era* has also begun to innovate the scientific area, it has even led to the establishment of a new scientific paradigm: The Fourth Paradigm [2]. As scientific instruments become more sophisticated and accurate, the data they produce, is growing torrential. Experimental, theoretical and computational science do not provide sufficient enough techniques for handling such huge amounts of datasets originating from various different sources. Therefore, a fourth scientific paradigm, also called data-driven science, has been introduced to fill this gap of techniques.
In contrast to the commercial area, the focus with scientific data is not to maximize the profits. Instead, it is to ensure a sustainable exchange between diverse scientific disciplines. Therefore, in order to enable data-driven science, the discipline of research data management is becoming increasingly important and visible.
Research Data Management (RDM) covers different aspects for handling research datasets with the aim to enable the exploitation of the maximum informational content in diverse research datasets. Hence, a sustainable access to research datasets is one of the key aspects in RDM. Research data repositories increasingly develop to highly complex systems which autonomously exchange their content with other information systems. Another key aspect in RDM is to establish global standards to enable global data exchange.

The current Internet is a network of nodes which store content of diverse types. To consume a

specific content, it is first of all necessary to access it by the locator of its current node. Usually, in addition to the locator, additional node-dependent parameters, like the name of the content, are required to distinguish an individual content among others on the respective node.
While current Internet applications, such as research data repositories, are solely based on the content itself, the current Internet protocol is based on the nodes rather than on the content. The fundamental obstacle with the locator-based content retrieval methodology is that it is not sustainable. Even a minor modification in the node or a transfer of the content itself to another node, could easily lead to the inaccessibility of the content, which is especially highly inappropriate for research datasets, in particular for the purpose of reproducibility. To solve this issue caused by volatile locators, for RDM the concept of *Persistent Identification* has been proposed [3].
*Persistent Identifier*s (PIDs) are globally unique and immutable identifiers, which are assigned to research datasets to abstract away their current locators. As a consequence, research datasets are increasingly retrieved by their globally unique PIDs instead of their actual locators. This in turn, has led the concept of persistent identification to play a fundamental role in RDM. Although PIDs are in use for more than twenty years now, the explosive research dataset growth has also a direct impact on PID systems, which is particularly true for the performance of these systems.

In this thesis, we investigate the increasing importance of PIDs for RDM with emphasis on the performance. Therefore, we provide contributions and solutions to meet the increased performance requirements for administering and resolution of PIDs to finally enable a stable and efficient global research data exchange.

1.1 Motivation

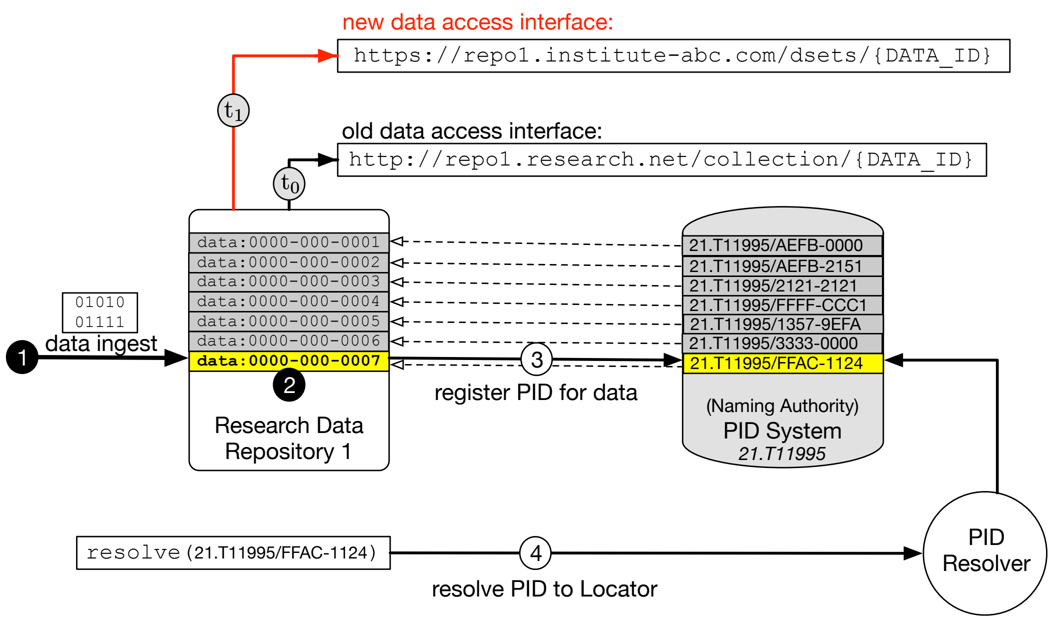


Figure 1.1: Research data repository full of research datasets. The contained research datasets are assigned with a PID at a specific PID system. PID systems are also denoted as *Naming Authorities*, since they are often only responsible for a certain set of PIDs. In this example, the PID system is the naming authority for (`21.T11995`)-PIDs.

An important solution for RDM is store research data in sophisticated research data repositories.

At present there are various different research data repositories in the Internet, each with a distinct architecture, functionality and internal data structure. To ingest or access datasets, each of these repositories offers an individual set of interfaces. Furthermore, the datasets, which are stored in research data repositories, become increasingly interlinked with each other. Hence, to be able to fully process an individual dataset, often requires the retrieval of other datasets.
However, the interlinking or referencing of datasets requires stable links, which is not provided by locators such as *Uniform Resource Locator*s (URLs).

In Figure 1.1 we can see a fictive research data repository, which offers a particular data ingest and access interface. Based on this example, the following explanation demonstrate the importance of PIDs:

Suppose at time t_0, the access interface of the repository to be:

`http://repo1.research.net/collection/{DATA_ID}`.

Further, suppose that at time t_1, this repository is subjected to modifications in its architecture resulting in a new data access interface:

`https://repo1.institute-abc.com/dsets/{DATA_ID}`.

Assume, that between t_0 and t_1, the repository is filled with a huge amount of datasets, where the ingest workflow for each dataset is composed of three steps: ① input, ② internal processing and storing and ③ PID assignment.

Further, assume that between t_0 and t_1, the datasets have been referenced from other datasets, web sites, research papers etc. If the datasets would have been referenced through their current locators, upon t_1, all these links would become invalid resulting in the inaccessibility of all these datasets, which is also known as the ”decay of web references” [4]. In contrast to that, the linkage through the corresponding PIDs would still enable their accessibility.
Therefore, the datasets stored in research data repositories are increasingly referenced through their corresponding PIDs, which provide stable links. However, a locator change always requires also an update of the corresponding PID-to-locator binding at the PID system.
The access via the PID requires its resolution through a particular PID resolver (step ④), which yields the current valid locator of the corresponding dataset.
Note that the problem of invalid links can also be caused by the migration of individual datasets to different repositories.

However, the registration or update ($\equiv$ administration ③) of a PID at a PID system usually also constitutes an expensive operation within the overall ingest workflow. Thus, for a huge amount of research datasets, step ③ often causes a significant performance degradation. On the other side, since PID resolvers have only a little global widespread in the Internet, also step ④ often causes a significant overhead in the access of research datasets.

As an example, the response time for the registration or update of a PID at the ePIC[1] PID system, offered by GWDG, is around $250ms$. For a research data repository, which has to update 2,000,0000 PIDs associated with its individual research datasets, we have the following extrapolation:

[1]http://www.pidconsortium.eu

$$\begin{aligned} 2,000,000 \text{ PIDs} \times 250ms &\widehat{=} 500,000,000ms \\ &\widehat{=} 500,000s \\ &\widehat{=} 139h. \end{aligned}$$

Therefore, the objective of this thesis is to provide solutions for research data repositories to:

(a) enable a **high performant administration** of PIDs (③), and
(b) enable a **globally fast resolution** of PIDs (④).

Note that in the reminder of this thesis, a single PID administration operation in step ③ is represented by an **OP**, such as depicted in Figure 1.2:

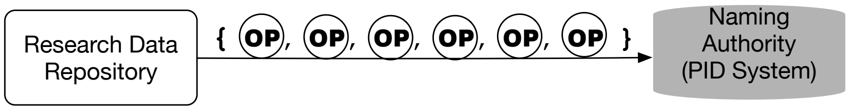


Figure 1.2: Research data repository administering PIDs at a PID system.

1.2 Scope of Thesis

Hence, our major hypothesis is that the performance of PID administration and resolution is highly important for research data repositories. To achieve a performance improvement, in this thesis, the respective research questions can be grouped into four approaches.
The first approach is to investigate the fundamental principles behind the concept of persistent identification. The corresponding research questions are:

- **RQ 1:** Why is persistent identification performance relevant?

- **RQ 1.1:** What is the goal behind the concept of persistent identification?
- **RQ 1.2:** What is the origin of performance problem caused by the concept of persistent identification?
- **RQ 1.3:** How to achieve a reduction of the overhead caused by PID administration?

In the second approach, a PID system is considered as an ordinary multi-tier transactional Internet system. As these systems require continuous maintenance and improvement, appropriate research questions to be investigated are as follows:

- **RQ 2:** How to support an efficient advancement of a multi-tier Internet system such as a PID system?

- **RQ 2.1:** What is an appropriate performance model for a multi-tier system?
- **RQ 2.2:** What is the response time behavior of the MVA algorithm?
- **RQ 2.3:** What is the impact of an improvement at an individual tier onto the overall system's response time?

- **RQ 2.4:** How the effect of an improvement endeavor can be described and estimated?
- **RQ 2.5:** Which effect have to be particularly taken care of after an improvement effort?

The common methodology of the following two approaches is a contrasting of a PID system against the well-known DNS system.
The third approach emphasizes on a fast PID record management by the following research questions:

- ■ **RQ 3:** How to achieve a high-performant PID record management protocol?

- **RQ 3.1:** What is the fundamental performance problem of the current Handle protocol in conjunction with the specific workload Handle servers are subjected to?
- **RQ 3.2:** How to enable a high-performant PID record management with the Handle protocol?

Finally, in the fourth approach, the focus is on the resolution performance of PIDs. The relevant research questions are the following:

- ■ **RQ 4:** How to achieve a high-performant PID resolution?

- **RQ 4.1:** How is the PID resolution time composed?
- **RQ 4.2:** What is the cause of the current unsatisfactory performance of PID resolution?
- **RQ 4.3:** How to improve the current PID resolution?

1.3 Goals and Contributions

For the area of persistent identification, this thesis provides advancement of the state-of-the-art through the following contributions:

Contribution 1: A **high-performance persistent identification concept** for identifying and accessing research datasets stored in a research data repository, which is equipped with an internal naming component for assigning internal identifiers for its research datasets (Chapter 3). This concept enables these internal identifiers to be globally resolvable without the need to individually register them at a PID system.

Where the contribution includes the following sub-contributions:

- The ***Research Data Silo Identifier*** **(DSID)** (instead of a PID), which directly identifies a research data repository to include the respective access interface for retrieving research datasets stored within the research data repository (Section 3.2.4).
- The new `HS_RDS_URL` **type**, which is necessary to realize this DSID concept within the Handle System (Section 3.3.2.2). The data part of this type is used to hold a particular locator composition rule to enable a dynamic locator composition.

Contribution 2: A **high-performance persistent identifier management protocol**, which is suitable for registering and managing large amounts of PID records (Chapter 5).

The overall contribution is composed of the following sub-contributions:

- Extension of the Handle protocol with the new `OC_CREATE_HANDLES_BULK` **operation**, used to administer multiple Handle-PIDs records by a single request (Section 5.4.4). Whereas the current Handle protocol only offers the `OC_CREATE_HANDLE` **operation**, which is used to administer an individual Handle-PID record.

- The new `HandleRecordsBulk` **data model** for including multiple Handle-PID records containing individual descriptive information about corresponding research datasets. This new data container is then transmitted with the new `OC_CREATE_HANDLES_BULK` operation as the message body (Section 5.4.4.1). In contrast to that, the current data model (Handle Record) only supports the inclusion of descriptive information for a single Handle-PID.
- An **algorithm implemented into Handle servers** to process the new `OC_CREATE_HANDLES_BULK` operation and its message body (Section 5.4.4.3). This algorithm defines the internal processing workflow for the new operation within a receiving Handle server.
- A **comprehensive performance evaluation** of the Handle servers, which reveals the performance of the current and extended Handle server implementation (Section 5.5).

Contribution 3: A **high-performance persistent identifier resolution concept**, which enables an accelerated resolution of PIDs through the well-established DNS system (Chapter 6).

This contribution in turn, is composed of the following sub-contributions:

- A **mapping algorithm**, which maps the Handle System data model (Handle Record) into a DNS Resource Record (Section 6.4.3). Since our idea is to resolve Handle-PIDs through DNS traffic, it is required to transform Handle Records into Resource Records.
- A **transformation algorithm**, which transforms Handle-PIDs into DNS domain names (Section 6.4.4). This is necessary, because the DNS system only supports the resolution of domain names, wherefore Handle-PIDs have to be transformed into DNS domain names at the resolution through DNS resolvers.
- An appropriate **DNS request processing algorithm** implemented into Handle servers for embedding them into the global DNS system. This algorithm is used to map incoming DNS resolution requests into corresponding internal native Handle protocol resolution operations to yield corresponding Handle Records (Section 6.4.6).

This thesis also advances the state-of-the-art of multi-tier Internet system performance analysis by the following contributions:

Contribution 4: An approach to investigate the effects of a **response time speedup of multi-tier Internet systems** (Chapter 4).

The following sub-contributions compose the above mentioned overall contribution:

- An **investigation of the response time decomposition** for increasing concurrency level calculated by the MVA algorithm (Section 4.2.2.1: Theorem 1). This theorem provides an understanding of the general composition structure of the overall response time calculated by means of the MVA algorithm.
- An **analysis of the response time behavior** for increasing concurrency level of the MVA algorithm (Section 4.2.3: Theorem 2). This theorem constitutes the core key for the derivation of the remaining mathematical results. More specifically, this theorem provides a description of the internal load distribution within a multi-tier system for increasing request population.
- **Response time boundaries** of the MVA algorithm (Section 4.2.3: Theorem 3). This theorem enables to quickly estimate the boundaries of the overall response time for increasing

concurrency level.

- A **mathematical description of the response time speedup** (Section 4.3.2: Lemma 1). This lemma provides insight into the impact of an individual speedup factor achieved at an individual tier onto overall response time speedup factor.
- A **mathematical description of the load redistribution effect** occurring after an improvement at an individual tier of the overall multi-tier system (Section 4.3.3: Theorem 4, Corollary 1). Theorem 4 basically reveals how an improvement at an individual tier mathematically impacts the remaining tiers.
- **Boundaries to estimate the overall response time speedup** after improving an individual tier (Section 4.3.4.1: Lemma 2). This lemma provides a tool, to quickly estimate the limits for the expected overall response time speedup factor, after improving an individual tier.
- **Boundaries to estimate the load at individual tiers** after an improvement effort in a specific tier (Section 4.3.4.2: Corollary 2, Corollary 3). These corollaries basically incorporate all aforementioned mathematical results. In principle, they provide insight into the expected load at the tiers after an improvement of a particular tier. Whereas Theorem 4 provides an exact mathematical description of the load redistribution effect consisting of several complex mathematical terms, these two corollaries provide approximations for these complex terms for different cases. Hence, these corollaries are applicable in practical situations, when only quantitative measures are available.

1.4 Impact

This thesis is based on intermediate results, which have been published in the following peer-reviewed conference proceedings:

- **F. Berber**, P. Wieder, R. Yahyapour, ”A High-Performance Pesistent Identification Concept”, Proceedings of the 11th IEEE International Conference on Network, Architecture and Storage (NAS2016), Long Beach, USA, Aug. 2016. DOI: 10.1109/NAS.2016.7549387
  URL: http://ieeexplore.ieee.org/document/7549387/
- **F. Berber** and R. Yahyapour, ”A High-Performance Persistent Identifier Management Protocol”, Proceedings of the 12th IEEE International Conference on Network, Architecture and Storage (NAS2017), Shenzhen, China, Aug. 2017. DOI: 10.1109/NAS.2017.8026839
  URL: http://ieeexplore.ieee.org/document/8026839/
- **F. Berber** and R. Yahyapour, ”DNS as Resolution Infrastructure for Persistent Identifiers”, Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS2017), pp. 1085-1094, Prague, Czech Republic, Sep. 2017. DOI: 10.15439/2017F114
  URL: http://ieeexplore.ieee.org/document/8104688/
- **F. Berber** and R. Yahyapour, ”Response Time Speedup of Multi-tier Internet Systems”, Proceedings of the 36th IEEE International Conference on Performance Computing and Communications (IPCCC), San Diego, USA, Dec. 2017. DOI: 10.1109/PCCC.2017.8280469
  URL: http://ieeexplore.ieee.org/document/8280469/

Furthermore, the following result is currently under review at the peer-reviewed journal:

- **SUBMITTED**:
  - **F. Berber** and R. Yahyapour, ”High-Performance Persistent Identifier Resolution”, in IEEE/ACM Transactions on Networking.

1.5 Thesis Overview

This thesis is organized as follows:

In Chapter 2, by reviewing the related research efforts, we provide the context and background for our contributions.
In Chapter 3, we investigate the realization of the concept of persistent identification and reveal its relevance for performance. In addition, we provide solutions, which reduce the administration overhead entailed with the usage of PIDs.
Chapter 4 emphasizes on the improvement of the response time of a PID system, which is usually composed as a multi-tier transactional Internet system. For this, we derive mathematical estimation formulas, which can support an improvement endeavor.
Chapter 5 investigates the specific workload a PID system is typically subjected to. This is followed by an appropriate approach to optimize the processing performance for such a workload.
In contrast to the previous chapters, Chapter 6 particularly captures the acceleration of the resolution time of PIDs.
In Chapter 7, we provide a discussion, which evaluates our research questions against the achievements of this thesis. Moreover, we also discuss the limitations of our deduced approaches.
Finally, with Chapter 8 we conclude this thesis and for the outlook, we provide indications for future research endeavors.

Chapter 2

Related Work

2.1 Research Data Management

In principle, research data management is originated in the discipline of digital preservation. The focus here is to preserve digital information over long time periods. Moore [5] therefore even designates digital preservation as the communication with the future. Furthermore, Moore proposes the *Integrated Rule-Oriented Data System* (iRODS) as a preservation environment, which enables digital datasets to be preserved for the future. iRODS can be thought as a software middleware, which abstracts the underlying storage systems, which in turn are subjected to technological evolution. However, the distinctive feature of iRODS is provided by the so-called micro-services. A micro-service defines a set of operations, which are applied to the incoming datasets. Each micro-service is executed whenever the corresponding rule applies to an individual dataset. As an example for such an micro-service is the computation of the checksum of a dataset to ensure its integrity. Another example is the registration of a dataset at a particular PID system. In principle, iRODS constitutes one of the early sophisticated research data repositories.
Staples, Wayland and Payette [6] introduce another research data repository system: Fedora. To some extend, it is similar to iRODS. However, in contrast to iRODS, Fedora focuses more on the management of the datasets, while in iRODS the primary focus is on the abstraction of the underlying storage medium and data replication. In addition, Fedora defines its own XML based data model, which provide the possibility to specify the relations between various data objects.
In this context, Zhu et al. [7] propose a repository architecture, which is based on iRODS and Fedora with the emphasis on combining both technologies to realize an overall digital preservation environment.
Marketakis and Tzitzikas [8] provide an approach to also preserve the *intelligibility* of digital objects. With *intelligibility* they mean the interpretability of the preserved bits composing a digital object. Their approach starts with a formalization of the problem based on the metadata, which basically results in a dependency graph. By processing the dependency graph, the goal is determine an optimal set of metadata, which is required to preserve the *intelligibility* of a digital object.
Koutsomitropoulos et al. [9] focus on digital repositories with a metadata model, which is "usually semi-structured from a semantics point of view". Targeted for such repositories, they propose a mechanism to enable Semantic Web techniques to be applicable on them. Their basic approach is to extend the repository "to provide inference-based knowledge discovery, retrieval and navigation" without modifying the original repository and its metadata model.
Jantz and Giarlo [10] emphasize on specific technologies that enable a digital repository to become a *trusted digital repository*. To achieve that, they propose three technologies: digital signatures, PIDs and audit trails. By means of digital signatures, they address the problem of the detection of unauthorized modifications to the datasets. However, the essential challenge with digital sig-

natures is to prevent the signature itself to become corrupted. To ensure permanent access to the datasets, they propose to use PIDs, which is also emphasized in this thesis. With audit trails, it is possible to trace back all the transformation steps an individual dataset has undergone. This in turn provides another mechanism to ensure data integrity.
Thibodeau [11] approaches the issue of digital preservation quite theoretically by partitioning a digital object into three classes. He concludes that ”every digital object is a physical object, a logical object, and a conceptual object.” Furthermore, he also concludes that the ”properties at each of those levels can be significantly different”. ”A physical object is simply an inscription of signs on some physical medium. A logical object is an object that is recognized and processed by software. The conceptual object is the object as it is recognized and understood by a person, or in some cases recognized and processed by a computer application capable of executing business transactions.” Hence, a conceptual object, can have multiple logical representations. Another complicating fact is that digital objects are increasingly interconnected with each other. Therefore, Thibodeau states that to preserve a digital object for the future, it is required to ”preserve its physical and logical components and their interrelationship.” He also concludes that preservation of digital content paradoxically requires modifications to be processable with future data formats, applications, operation systems and hardware systems.

2.2 Persistent Identifier Introductory

The core intention behind the concept of persistent identification is based on the volatile access methodology originated from the current Internet network setup, which is also inherited by URLs. In the particular case of URLs, over time, this network setup leads to the occurrence of more and more invalid URLs, which is also known as *link rot* or *URL decay* phenomenon. Several works, [12], [13], [4], [14] and [15] are addressing this problem, where PIDs can be considered as a consequence of this problem.
One of the first works about PIDs is provided by Kahn and Wilensky [3]. They envision the Handle System as a means of providing ”universal basic access to registered digital objects”, rather than emphasizing it in the context of persistent access. Instead, the whole work focuses on ”network-based aspects” of a digital repository infrastructure, without going into aspects related with the content itself. Thus, they abstractly propose digital repositories to support a simple and common protocol for accessing and manipulating digital objects: the *Repository Access Protocol* (RAP). In this protocol, the `ACCESS_DO` operation is used to access digital objects, while the `DEPOSIT_DO` is intended to perform manipulation on the digital object.
Also the Handle System is only quite abstractly introduced, whereby a Handle server constitutes an integral part of digital repository in their proposed architecture of a digital repository. The core function of such a Handle server is to provide a component for ”naming, identifying and/or invoking digital objects”. In addition, they discuss to avoid semantic expressions in the Handle-PIDs identifiers. The complete specification of the Handle System itself is given by [16].
Paskin [17] states that the ”management of intellectual content” in the Internet ”requires the existence” of PIDs. He also argues that URLs and the *Resource Description Framework* (RDF), ”provide an infrastructure for managing and resource discovery and distribution, but not a sufficient framework in which to manage intellectual content and the rights which accompany that content, such as access rights and copyright.” Hence in contrast to [3], this work of Paskin focuses more on the content itself. Moreover, Paskin proposes the *Digital Object Identifier* (DOI) system, as a mechanism to ”enable intellectual content management to be integrated with Internet technologies”.
However, the actual architecture of the DOI concept is addressed in another work of Paskin [18]. According to this work, the DOI concept is made of three components: the DOI identifier, the DOI

resolver and the DOI metadata. The resolution infrastructure of the DOI system is provided by the underlying Handle System, therefore a DOI identifier is always also an ordinary Handle-PID with a prefix of the form `10.X`. The main additional or differentiating component is constituted by the DOI metadata, which is decoupled from the underlying Handle System. In order to assign a digital object with a DOI, it is necessary to comply with the relative strict metadata set [19].
Besides the Handle System, another important PID system is proposed by Kunze [20], which is the *Archival Resource Key* (ARK). In contrast to the Handle System, the ARK concept is limited on URLs as data locators. Another aspect is that Kunze envisions an ARK identifier to be associated with three "things": "the object, its metadata, and the current provider's commitment statement". The selection of these "things" is accomplished via additional parameters appended to the ARK identifier. He proposes an ARK followed by the "?" character to deliver the metadata, and by appending "??" to yield the commitment statement. The actual digital object is then provided by the sole ARK identifier without any appended character.
Hilse and Kothe [14] provide a comprehensive introductory work about PIDs and discuss and compare several existing PID systems. In this work, the authors also provide the related background information, such as history, functionality, implementation, etc., about each of the considered PID systems. The essential intention of this work is to enable an appropriate PID system choice for a specific use case, therefore the authors provide a checklist of relevant questions which aim to distill an appropriate PID system.
Nicholas, Ward and Kerry Blinco present an abstract model of identifiers and identifier schemes [21] with the goal to provide a mechanism to compare different identifier schemes. By means of this abstract model, which is formal and layered, they determine that also *Uniform Resource Identifier*s (URIs) "qualify as identifiers, provided that they are appropriately managed".
In the context of URIs, Kuhn and Dumontier [22] propose trusty URIs for providing trust and reliability for research datasets. The focus is on using cryptographic hash values in URIs corresponding to identified research datasets, so called trusty URIs. A trusty URI can then be used to determine whether the identified datasets has been subjected to manipulations. The essential difference to the approach in [10] is that Kuhn and Dumontier impose the signature into the identifier itself instead to store it as an additional attribute. However, the authors do not consider the use of PIDs for such a manipulation check. The advantage with PIDs is that they do not change even when the research datasets move to another storage location. Another issue with trusty URIs is related with the digital preservation. As indicated by Thibodeau [11], a digital object can have multiple logical representations, and therefore also several different resulting hash values.
Hakala [23] provides another overview work about PIDs, whereby a specific set of PID systems is compared with *cool URIs* [24]. A similar approach represents the work of Tonkin [25].
Richards, White, Nicolson and Pyle [26] provide a further introductory work about PIDs. As [14], the authors in this work provide a comparison of some PID systems and a checklist for implementing of PIDs on specific datasets. In contrast to [14], this work is more targeted on individual researchers or research groups, therefore, it also covers the granularity issue of research datasets. Since research datasets usually originate from a specific context, the authors provide guidelines of how to apply PIDs for the degree of fineness of such contexts.
Also the use of internal identifiers is discussed, which is also in the scope of this thesis. However, the authors do not consider to enable these internal identifiers to be globally resolvable.
Another comparison work is presented by Duerr et al. [27]. They address various PID technologies and evaluate their usage for earth science data. The focus here is on data, which is stored at multiple storage locations. In their view, an unique identifier should identify all copies of an individual dataset. Moreover, they define an ideal identifier as one which is generated at the dataset's creation time, as one which is incorporated into the dataset itself and as one which is additionally associated with descriptive information about the dataset. Furthermore, they state that such

an identifier can be used within a ”verification process” since the identifier is embedded in the dataset, which finally serves the purpose of ”reproducibility and verifiability”. In principle, the Handle System enables to associate a single Handle-PID with multiple URLs, however, it lacks in an efficient selection algorithm, which results the best locator upon a resolution request.
A work, which discusses PIDs in a more sophisticated context is provided by Van de Sompel, Sanderson, Shankar and Klein [28]. However, it also represents a work, which considers the concept of persistent identification only as a simple redirection mechanism. Therefore, their focus is to devise a ”machine-actionable bridge” from PIDs to URIs, which are associated with semantic information or metadata about the identified research datasets. This is required because of the increasing number of machine agents depending on machine readable information stored in the Internet. In contrast to Kunze [20], the approach in this work is based on special HTTP linking headers defined in [29]. A machine agent can then follow specific links contained in the HTTP response, which in turn result from a PID resolution. By this, the machine agent is directed to a representation containing machine-actionable information. A human user in turn, will be provided with a human-readable representation of the identified research dataset. However, the essential aspect, which is not considered by this work, is the fact that PID records can, in addition to the sole URL, also be equipped with semantic information relevant for machine agents.

2.3 Advanced Usage of Persistent Identifiers

As one of the first works which considers more advanced aspects for the concept of persistent identification is presented by Bellini et al. [30]. The authors provide a hierarchical architecture for a PID system based on the *Uniform Resource Name* (URN) scheme [31]. To overcome a centralized architecture in order to avoid a single point of failure, they propose a hierarchical architecture which reflects the DNS system. For URN-based PID systems this is useful, because URN is first of all purely a scheme for providing namespaces and identifiers. One of the most important naming schemes is the URN:NBN namespace, which is used to uniquely identify resources in the area of national libraries. The main problem the URN concept is that there is no central and globally known resolution system in place, which is capable of resolving all namespaces. Therefore, Bellini et al. basically provide a PID system architecture which implements the URN scheme.
In another work of Bellini et al. [32], the authors address the problem of the fact that there are multiple different PID systems in place. The authors propose an interoperability framework on top of the diverse set of PID systems. This framework can be considered as a technology which provides a common interface, which can be used by individual users to retrieve data associated with individual PIDs and their underlying PID systems. The core principle behind the interoperability framework is an ontological refinement. Relevant services scrutinize the metadata associated with PIDs in order to determine a common ontology. The distilled ontology is then used by individual ”content providers” to implement the actual mapping from the original format into the common ontology format supported in the interoperability framework. Hence, it requires additional processing from the ”content providers” to provision the respective *Interoperability Knowledge Base* (IKB), which holds the data in the corresponding common ontology structure. Thus, the entire set of IKBs form the base data source of the interoperability framework.
The authors Weigel, Kindermann and Lautenschlager [33] present a work, which can be considered as a first attempt towards semantic PID records. The authors emphasize on using common abstract data types [34] incorporated into PID records which then can be autonomously understood and processed by machine actors. In addition, the authors consider such PID for the following use cases: Data object replication, data object provenance, composite data objects, data object versioning.
In the context of data replication, PIDs provide a medium which ”can help to identify all replicas

of a master object". It should be noted that the replication mechanism of the EUDAT project [35] is highly based on PIDs. By means of additional special metadata types imposed into the PID records, it is possible to "keep track of" the replicas and the master object, which have to appropriately specified within the respective PID records.
Also the issue of data provenance becomes increasingly important, since data constitutes an intellectual property, which has be protected against unauthorized modifications. Furthermore, data is often derived from other data. Thus, for this use case, PIDs can provide a mechanism to reveal all the derivations of an individual dataset.
Related to data provenance is the aspect of data versioning. Also in this context, PIDs constitute an appropriate registry to record all modifications an individual dataset has been subjected to.
Composite data objects are data objects, which are composed of various other data objects. Here, PIDs can enable to define the individual relations between the data objects. Thus, the inter-linking of individual data objects, which actually compose an overall data object is then accomplished within the corresponding PIDs records.
Finally, to realize the proposed approach, the authors implement their idea with Handle Records of the Handle System. A Handle Record represents the data model of the Handle System used to store information associated with an individual Handle-PID, such as the URL of a dataset.
The work of C. Liu, Yang and T. Liu [36] is an example for the need of a naming component in the *Internet of Things* (IoT) paradigm. The main idea is in principle to provide a DNS-like system specifically targeted for IoT devices. However, this is de facto already existing: the Handle System, which is also already in productive operation since several years. In addition, the data structure (Handle Records) of the Handle System is generic enough to hold any kind of data including device information. Another major advantage of the Handle System is that there are also commitments and guarantees for a long-term operation.
In the context of IoT, the EU-China IoT Advisory Group [37] discuss PIDs as a concept for identifying "things".
An interesting work is provided by Bolikowski, Nowinski and Sylwestrzak [38]. The starting point of the authors is the assertion that PID registration and management is usually bounded to a single organization which therefore poses a single point of failure. As a solution for their assertion, they propose a decentralized technology called *Peer-Minted Persistent Identifiers*, which is highly inspired by the Bitcoin concept [39]. By means this technology anyone can register PIDs and associate the respective records with a URL and a list of authorized users, which can perform modifications to the respective PID record. As with Bitcoin, the technology is based on cryptographic chains, and hence, registration and updates lead to the extension of the chain. However, the problem is that extending the chain requires a certain amount of CPU computation, which the authors denote as *proof-of-work*. This means, in addition to an expensive transactional storage operation, with this technology, PID registration would also involve a formidable CPU computation overhead. For explosive amounts of research datasets, this is quite inappropriate. Considering their assertion, the Handle System constitutes a highly distributed PID system since it enables a naming authority, which is service for registering and managing PIDs, to be distributed among different network locations and therefore to overcome a single point of failure.
Evrard et al. [40] introduce ORCID, which is a global persistent registry for scientist identification. In their concept, a PID represents an individual scientist. Therefore, the corresponding PID, namely ORCID record, contains the bibliographic information of the scientist.
Wannenwetsch and Majchrzak [41] address a critical point of the concept of persistent identification, which is constituted by the fact that PIDs have to be steadily administered in order to keep them valid. Therefore, the authors assert to propose "maintenance-free" PIDs. However, the core of the work consists of creating PIDs for datasets stored in a decentralized data distribution platform such as BitTorrent [42]. Since, data in the BitTorrent network is addressed by the *Mag-*

net Link Scheme, instead of an ordinary URL, the respective PIDs have to reveal *Magnet URIs* upon their resolution. To implement their idea, they utilize the Handle System. Thus, in principle their essential contribution is to create PIDs, which contain *Magnet URIs* for datasets stored in a decentralized data distribution platform, in which data is retrieved by Magnet Links. And since these Magnet Links are independent of the actual location of the datasets, they will keep intact even when the corresponding datasets moves within the decentralized data distribution network. Hence, PIDs containing Magnet Links become ”maintenance-free”. Therefore, the authors actually do not address the core problem of PIDs itself, which would also include PIDs containing ordinary locators, such as URLs. Such PIDs represent the vast majority among the set of all registered PIDs. Another issue is that, to some extent Magnet Links can be considered as PIDs as well, which means that ordinary PIDs associated with data stored in such networks as BitTorrent represent only alias identifiers.
Schwardmann [43] proposes a registry, the ePIC PID Information Type Registry[2], which enables to define machine processable types, whereby the registered types can be incorporated into Handle-PIDs. Handle Records equipped with these types can then be autonomously processed by machine actors. More specifically, by means of this registry, a machine actor can determine the interpretation rule of a registered type and then apply the rule(s) on the actual data stored in the Handle Record. Initially, the idea of such semantic PID records was introduced by Weigel, Kindermann and Lautenschlager [33]. However, Schwardmann contributes with a registry, which can be publicly used to register standardized types, which in turn enable a broad adoption of the idea of semantic PID records. However, for such a registry, its highly essential to establish reliable commitments and guarantees for a long-term operation. In addition, for an efficient global operation, such a registry has to be established as a distributed architecture, such as the DNS system or the Handle System itself. Otherwise, this could lead to performance problems due to increasing request loads, especially with respect to machine actors.

2.4 Persistent Identifiers in the Context of ICN

Xylomenos et al. [44] present a comprehensive overview work about the ICN paradigm. Although there are various different initiatives for an ICN network, the authors identify the following ”key functionalities”: Naming, name resolution and data routing, caching, mobility and security. Based on these ”key functionalities”, the authors conduct a comparison of all existing ICN initiatives [45], [46], [47], [48], [49], [50] and [51]. Although, the working principle of PIDs is pretty similar to some of the ”key functionalities” provided by the ICN approaches, the authors lack in a comparison between the concept of persistent identification and the ICN paradigm solutions.
In principle, Sollins [52] proposes the URN identifier scheme for different ICN implementations, in particular for the PURSUIT system [47]. The author first points out the importance of a reliable and stable identification concept for the ICN paradigm, which is characterized by the ”what” rather than the ”where”. Therefore, the core of ICN is based on the identifier, which in turn is used to address and retrieve the ”what”. As a solution, the author presents an URN like solution, whereby the identifiers resemble the syntax of the URN identifier scheme. In addition, Sollins describes the resolution procedure to be accomplished by involving a ”global idspace service” (G-PPOID RS) and a ”distinct sub idspace service” (S-PPOID RS), which pretty much resembles the setup of the Handle System or the ARK concept. Hence, instead of considering these existing PIDs systems for ICN, the author basically combines concepts, in particular naming (URN) and resolution (Handle System and/or ARK identifier), of different established PID systems.
One of the first works, which explicitly addresses PIDs in the context of ICN is provided by

[2]http://dtr.pidconsortium.eu

Karakannas and Zhao [53]. Although the authors compare different ICN realizations and PID systems, their main objective is to enable the ”delivery” of datasets which are actually accessed through PIDs via *Named Data Networking* (NDN) [46]. To realize the delivery, the authors introduce a special NDN ”gateway” for PIDs, which holds the mappings between PIDs and their corresponding NDN names. Hence, to retrieve a PID identified dataset, this gateway functions as a registry which responds with the corresponding NDN name in order to consume the dataset stored in the NDN network. However, this NDN gateway basically constitutes another lookup system, in addition to ordinary PID systems, and also, in addition the NDN infrastructure. The authors do not consider the option that PID records enable to hold multiple locators, which would result the special NDN gateway to become unnecessary.
The work of Schmitt, Majchrzak and Bingert [54] can be considered as an extension of the work provided by Karakannas and Zhao [53]. The objective of Schmitt, Majchrzak and Bingert is to enable PID management over the NDN network, while Karakannas and Zhao only consider the dataset retrieval. In principle, Schmitt, Majchrzak and Bingert consider two different overlay networks, the Handle System and the NDN network. Hence, to enable Handles Records to be manageable within the NDN network, they propose to extend the Handle server with a NDN interface, such that the Handle servers are also integrated into the overlay network of NDN. In addition to a NDN interface, the Handle servers are also equipped with an appropriate transformation algorithm which converts NDN requests (at the Handle server) into native Handle protocol requests and vice versa. By this, NDN requests targeted to specific Handle servers are then accepted by Handle servers, which in turn, perform a mapping of the incoming requests into corresponding Handle protocol operations, such as resolution, creation, updating etc. Furthermore, the authors conduct a performance evaluation, in which they compare the performance of the native Handle protocol stack and the NDN enabled interface. It turned out, that with the NDN enabled interface, the response times are approximately five times longer than with the native Handle protocol interface, which is tremendous for high-performance.

2.5 DNS System

In particular the established Handle System, but also the design of other existing PID systems have been more or less influenced by the well-known and heavily used DNS system. Therefore, in this section we discuss DNS-related works, while we focus on performance-relevant contributions.
Jung, Balakrishnan and Morris [55] present a work, which addresses the ”client-perceived” DNS resolution time and the ”effectiveness of caching”. For that, they collected DNS and TCP requests at two different network nodes (USA and South Korea), where these nodes present gateways from an internal network to the Internet. The idea behind to also gather TCP connections was to determine the effect of caching. Since, DNS is mainly used to resolve a domain name into its corresponding IP-address, which is then required to address specific hosts in the Internet, a TCP connection is often preceded by a DNS request for the IP-address of the target node. Hence, by taking to ratio between TCP connections and DNS requests, the authors were able to reveal a DNS cache hit ratio around 85%. Furthermore, during their one month analysis phase, they found out that at the first node (located in the USA), the median resolution time was about $85ms$. In contrast to that, at the second node (located in South Korea), the median resolution time resulted in $42ms$. The difference of the median resolution times at these two locations, is based on the fact that for the node located in South Korea, a caching DNS server was in close proximity of the gateway node.
Cohen and Kaplan [56] present another work for accelerating DNS involved communication. They propose two different techniques. First, the *renewal* technique, in which certain DNS records are queried ”unsolicitedly”. In contrast to *preresolving*, also proposed by Cohen and Kaplan [57], *re-*

newal is incorporated into nameservers, which then automatically provision their caches with the respective DNS records. Instead, *preresolving* ”utilizes predictions made based on per-user access patterns and currently-viewed hyperlinks”. Whereby the relevant information for *preresolving* is ”available at the user’s browser or proxy server” and hence, the technique of *preresolving* is then performed by clients. The second proposed technique is to apply *simultaneous validation*, which is to establish the HTTP connection based on expired domain names containing IP addresses from the local cache. However, during HTTP connection establishment, another request is concurrently (simultaneously) issued to fetch the current DNS name-to-IP binding. Since, a HTTP connection is usually preceded by a DNS query for an IP address, the technique of *simultaneous validation* is addressed to reduce the latency. The results from the HTTP connection and the DNS query are therefore validated in a concurrent manner. The problem of these techniques is that they require the implementation and support of the algorithms at the nameserver and client respectively, which is hindering the broad adoption.

The work of Yu, Wessels, Larson and Zhang [58] investigate the algorithm for nameserver selection of various DNS resolvers. Since a DNS zone often consists of multiple nameservers to ensure load-balancing and high-availability, current popular DNS resolvers employ specific algorithms to distribute the query load among these nameservers. The authors analyze the effectiveness of these distribution algorithms, where most of them are based on the latency (RTT) between the DNS resolver and the nameserver. In order to prevent the query load to be concentrated at the nameserver with the smallest latency, the algorithms cause the latency values of the individual nameservers to be weighted by specific values, which finally lead the queries to be partitioned among the authoritative servers. The essential intention of the authors was to determine if the partitioning is reasonable and effective. However, they found out that the employed algorithms also include unresponsive nameservers, which cause long resolution times. The authors therefore suggest to revise the selection algorithms, since they often cause ”large-latend” servers to be subjected with disproportionate query loads.

Often, nameservers of top-level domain zones (TLDs) are also geographically distributed. Thus, in order to redirect a domain name resolution request to the nearest nameserver, usually the technique of anycast is applied. In anycasting, multiple hosts are reachable through the same IP address. To redirect the request to the nearest host is based on the static hop count, which is used by network switches to determine the port the packets to be sent out. Thus, Sarat, Pappas and Terzis [59] analyze the effectiveness of anycast for DNS server selection, in which they reveal the general usefulness of anycast.

However, in *content distribution network*s (CDNs) anycast alone usually does not suffices for efficient redirecting of user requests to appropriate CDN servers. Since in anycast the routing decision is done by the network switches based on the static hop count between different autonomous systems, dynamic parameters such as the current load at individual servers are not covered by anycasting. Therefore, in CDNs the request routing is based on special DNS servers which are basically tasked with the collection of routing relevant parameters. The collected parameters are then used to determine the most appropriate CDN server.

In this context, Shaikh, Tewari and Agrawal [60] analyze the ”effectiveness of DNS-based server selection”. However, DNS-based server selection is also based on two other aspects, first, the reduction of the TTL values in DNS records and second, the assumption that the proxy DNS resolvers, used by clients/applications, to be representative for the client/application location. The authors found out that the small TTL values result in a significant growth of the resolution overhead within the global DNS infrastructure, which can in future endanger the stability of the global DNS system with the increased usage areas and requests. The authors also show that the assumption, that the client/application is in close proximity of the proxy DNS resolver is often ”violated”. In addition, they propose to develop further mechanisms which take into account their findings.

As a possible solution, they propose to extend the DNS protocol, to "carry additional information to identify the actual client making the request".
Furthermore Pan, Hou and Li [61] provide a detailed investigation of the working principle of request routing in current CDNs. The authors reveal that current CDNs employ different DNS servers with different functions in combination with anycast to achieve a reliable and effective request routing. At the heart of such architectures are these already mentioned DNS servers which collect parameters to serve as a request history memory. The primary function of such request history memories is to constitute a base for request routing decisions.

2.6 Accelerating Database Ingests

As we have seen, the works related to DNS basically only cover to improve the retrieval of resources. For PID systems, however, in addition to retrieval also the administration is highly important. Therefore, in this section we briefly provide an overview about works, which consider the speedup of ingest operations by employing the technique of data bulking. Bulk insert is a well-known technique to insert new records into databases. The works of Wiener and Naugthon [62] and Jagadish et al. [63] provide examples for the use of bulking. In these works, the authors focus on improving bulk insertion into a single database.
However, with the immense digital data deluge, in order to ensure scalability, the architecture of recent data management systems becomes increasingly clustered. Therefore, the work by Silberstein et al. [64] for example, focuses on an efficient bulk insertion into such a clustered architecture.
Common to all these works, is that they employ a specific grouping of datasets, on which then a certain database-relevant operation is applied, instead of processing each individual dataset in a separate manner.
Kepner et al. [65] achieved 100,000,000 database inserts per second with the Apache Accumulo[3] database. The Apache Accumulo database belongs to a newer group of databases, namely to the group of NoSQL databases. It is known that NoSQL databases usually provide enormous performance improvements. However, the underlying principle behind these enormous performance gains is the relaxation of the ACID principle ([66, p. 566]), which ensures the *Atomicity*, *Consistency*, *Isolation* and *Durability* of a transactional database operation. For PIDs, which are meant to provide persistent access and unambiguous global identification, the ACID principle is essential. Therefore, in this thesis, we emphasize on performance improvements complying with the ACID principle.

2.7 Performance Analysis

The performance analysis of computer systems is a mature area. Various techniques and models have been developed and discussed.
The work of Jain [67] provides a comprehensive overview about the topic. Another example for a work, which presents a general overview about performance measuring in computer system is given by Lilja [68].
In this context, the analysis of queuing networks often constitutes one of major aspects, which is also covered in this thesis. This is based on the fact that PID systems are usually composed of several components, such as myriads of other services offered in the Internet. These systems in turn enable to be modeled as queuing networks, which finally enable to analyze their performance behaviors.
Queuing networks in turn, are composed of multiple queuing systems, which are analyzed by

[3]https://accumulo.apache.org

methods provided by the discipline of queuing theory. In principle, a queuing system is characterized by three elements: First, the arrival distribution, secondly, the service time distribution and thirdly, the queue discipline.
The arrival distribution specifies the interval between incoming requests to be processed by the service center.
The service time distribution determines the amount of time individual requests are in the state of processing in the service center.
Finally, the queue discipline characterizes the ordering a queue filled with requests is emptied by the service center.
Depending on the character of these individual elements, the analysis could become rather complex, which is particularly true for the combination of multiple individual queuing systems to form a queuing network. Therefore, simplifying assumptions are applied to enable the analysis of such complex queuing networks. In this context, the so called M/M/1 queue (so called Kendall notation), represents the simplest queuing system, where the first and second "M" denote the arrival process and the service process respectively to be of Poisson nature. The "1" denotes that the queue is served only by one service center. Another assumption is that the queuing system applies the First-Come-First-Served (FCFS) queue discipline. Hence, the overall processing time for an individual request results in a service time superposed with a waiting time. The most important aspect of an exponential distribution, such as the Poisson distribution, is that it is *memoryless*, which is specified by the "M" in the Kendall notation: assume a request in a queuing system was already waiting t_p time units, with an exponentially distributed waiting time, the probability that the request has to wait t_{add} additional time units for initiation of the processing at the service center, is the same for any t_p. Hence, an exponential distribution does not take into account the previously waited t_p time units, wherefore it is denoted as to be *memoryless*.
A fundamental property of M/M/1 queuing systems is that the departure process (requests leaving the system) is the same Poisson process as the arrival process (requests entering the system). Exactly this property enables the analysis of complex queuing networks. The analysis of queuing networks complying with this property, reveal in the product of the results of the individual queuing systems composing the overall queuing network. The overall solution is then composed as if each individual queuing system has been considered independently from the remaining queuing systems and by deducing the overall solution by calculating the product of all the individual results. Therefore, queuing systems equipped with this property forming a queuing network are also known as *product-form queuing networks*.

A contemporary about queuing theory and queuing networks is provided by Bhat [69]. The author briefly introduces the foundations about the topic and finally discusses their aggregations in queuing networks. Another characteristic of this work is that the authors also provide the corresponding historical background behind the individual concepts.
Jain [67] additionally addresses possible target areas of queuing networks and their appropriate analysis methods in great detail. However, the underlying assumption behind that work is that even fine-grained parameters are easily retrievable, which is usually not the case in practice. This is specially true with the new virtualization paradigm.
Another methodology for analyzing queuing systems and queuing networks is provided by operational analysis. The work of Lazowska, Zahorjan, Graham, and Sevcik [70] represents a comprehensive contribution for the area of operational analysis, which is most suitable to investigate the parameters of the actual state of the system. The fundamental tools of operational analysis are the operational laws, where Little's Law constitutes the most important one:

$$L = \lambda W, \tag{2.7.1}$$

where L denotes the queue length, λ, the arrival rate into the system and W, the waiting time.

2.7.1 MVA

Usually, the analysis of queuing networks is based on mapping their performance characteristics into average, mean measures, which is realized by applying the MVA algorithm on the queuing network. The MVA algorithm is applicable when the queuing network has a *product-from solution* and a fixed request count circulating within the overall system. The fundamental principle of the MVA algorithm is to recursively calculate the overall response time R_T^n of the queuing network populated by n requests.
By Reiser and Lavenberg [71] it was shown that for a product-form queuing network, the processing time of the ith queuing system results in

$$R_i^n = S_i(1 + L_i^{n-1}), \tag{2.7.2}$$

where S_i represents the time, the service center requires for processing and hence, the service time. More importantly, L_i^{n-1} represents the request population at the ith queuing system when the overall queuing network is populated by $n-1$ requests. Hence, the nth arriving request at the ith queuing system will see a request population with a mean population count of L_i^{n-1} requests. Thus, as if the overall queuing network is populated by (only) $n-1$ requests.
Another representation of the MVA is given by the following equation:

$$R_i^n = S_i(1 + A_i^n), \tag{2.7.3}$$

where A_i^n denotes the mean request population an arriving request finds at the ith queuing system. From this equation, it immediately follows the recursive nature of the MVA algorithm:

$$A_i^n = L_i^{n-1}, \tag{2.7.4}$$

which basically reveals the dependency of performance measures of the concurrency level n onto performance values of lower concurrency levels $n-1$.

2.7.2 Internet System Performance

There are myriads of different application areas of queuing models in Internet systems. In this section, we provide an overview of the application areas.
Slothouber [72] proposes a queuing network model for HTTP Web servers, where the author models a single Web server by the fundamental resources it utilizes (CPU, Storage, Network). With this model, the author showed that with increasing load on the Web server, the response time is increasing ”gradually up to a point”. A further load increase then leads to a rapid growth of the response time.
The objective of Doyle et al. [73] is to enable an efficient resource provisioning on which clustered Internet services are dependent on. The paradigm of CDNs, which distributes individual services among different hosts, entails a provisioning problem. Thus, to ensure the available underlying resources to be efficiently utilized, the authors propose a ”model-based resource provisioning” (MBRP). The authors emphasize on the performance of the basic resources such as CPU or storage. Their approach is to apply performance prediction models derived from queuing theory onto these basic resources. By means of a resource provisioning algorithm it is then determined how to distribute the load among the available resources. This algorithm is based on the predicted parameters from the basic resources.
The work of Stewart and Shen [74] represents another work in the category of efficient resource provisioning in distributed systems using queuing models. Furthermore, also another work of Stewart and Shen [75] can be counted into that category.

Kamra, Misra and Nahum [76] present ”Yaksha” to be an admission controller for 3-tiered Websites. Yaksha is a proxy system in front of the actual Website, which prevents the overloading of the actual Web service. The working principle of Yaksha is to control various parameters of the Web service, to ensure the response times not increase rapidly, such as observed by Slothouber [72]. The control and decision mechanism of Yaksha is based on a M/G/1 (Kendall notation) Processor Sharing queue modeling of the underlying Web service, where ”G” specifies that the service time has a general distribution.
One of the first works which models a multi-tiered Internet systems as a queuing network is presented by Liu, Heo and Sha [77]. Since the authors also take into account the multi-threaded nature of Web services, to solve the queuing network by means of the MVA algorithm, they adopt an approximation method ([78]). The main idea is to model each tier with two successive queuing systems, whereas the first queuing system is used to represent the size of the thread pool on which the requests are distributed to. The second queuing system in turn, is used to model the actual processing delay.
In this thesis, our contribution related to multi-tier Internet system performance improvement is mostly influenced by Urgaonkar et al. [79]. The work of Urgaonkar et al. represents one of most important works for performance modeling of multi-tier Internet systems. The basic approach of the authors is to model each tier as a single queuing system as it is done in this thesis. Moreover, the authors apply the MVA algorithm on the resulting queuing network to determine the mean response time of the overall system. In addition, they also provide a modeling to simulate the case, when the queuing capacity at each queue is exceeded, which they describe in the section ”handling concurrency limits at tiers”. To capture this behavior, before each individual queuing system of the queuing network, the authors insert a mechanism which forks the incoming requests into two different paths. The first path represents the regular workflow in which each request is traversing the ordinary queuing systems to perform the ordinary tasks to finally retrieve a successful response. The second path in turn is selected with a particular probability to simulate the requests which are discarded by a tier due to an exceeding of the corresponding concurrency limit. This is realized by leading these requests into another queuing network which finally cause the affected requests to start the processing again at the first tier. The intention behind the second path is to capture a certain retry behavior of discarded request, which is often implemented in recent multi-tier Internet systems.
Also the work of Bhulai, Sivasubramanian, van der Mei Maarten and van Steen [80] is basically an extension of the work of Urgaonkar et al. [79]. Since Urgaonkar et al. emphasize on the mean response time, the basic extension of Bhulai, Sivasubramanian, van der Mei Maarten and van Steen consists of additionally providing the variance bounds of the response time.
Stewart, Kelly and Zhang [81] provide a work, which addresses the aspect that often multi-tier Internet systems offer different types of operations with different service times. Hence, such multi-tier Internet systems are usually subjected to a ”mixture” of different requests, where each request type requires a particular utilization of the available resources (CPU, disk, network, etc.). This then finally complicates the modeling and predication of the response time behavior. The authors propose to express the response time as a combination of services times multiplied by the number of the corresponding request types occurring in a specific interval. Hence, their resulting response time decomposition is a parameter of an interval, rather than the load. To accumulate the load, the authors provide an extended response time decomposition, which is superposed with waiting times of M/M/1 queuing systems. In their approach, each tier is associated with three queuing systems for each resource (CPU, disk, network).
The aspect of simultaneous resource possession is dedicatedly considered by Rolia and Sevcik [82]. In principle, this work constitutes the base for the *Layered Queuing Network* (LQN) methodology. The basic approach of this methodology is to also take into account each com-

ponent/resource within a tier. This therefore requires a detailed understanding of the internal architecture and the respective parameters, which is in practice often hard to realize, especially in systems including third-party components.
Another work contributing to the LQN concept is provided by Neilson, Woodside, Petriu and Majumdar [83]. The authors concentrate on improving the throughput of a layered/tiered system. Moreover, they introduce the concept of *software bottlenecking*, which is performance constraint onto a layered/tiered system. A *software bottleneck* is a phenomenon, which is characterized by the effect of load ”push-backing” caused by a fully utilized tier, which then leads its predecessor tiers to be also over-utilized, whereas its successive tiers are under-utilized. Therefore, the load is called to be ”push-backed” onto the previous tiers. In addition, the authors introduce the quantity of the *bottleneck strength*, which is a measure of the potential throughput improvement by using ”clone-sets”. By clone-sets the authors mean to extent the queuing capacities of individual tiers by increasing the thread pool size or by positioning additional instances of individual tiers on which the load is distributed to.
A similar work is presented by Franks, Petriu, Woodside and Xu [84], which can be considered as an extension of [83]. The authors introduce a corrected form of the *bottleneck strength*, which is a factor, which specifies the maximum possible throughput increase after relieving a fully utilized resource/tier. In the old definition constituted in [83], it is determined by the utilization ratio of the saturated resource/tier and its direct successor. The corrected form in turn is defined to determine the utilization ratio between the saturated resource/tier and one of its successive resources/tiers with the highest utilization value. The main difference is that in the corrected form also the indirect successors are taken into account. This is reasonable because the overall system’s throughput will be limited by the second most utilized resource/tier after improving the most utilized resource. Furthermore, the authors propose three strategies to achieve a relieve of the saturated resource. The first strategy is to ”add resources”, which result in a distribution of the load among the additional resources. In principle, this strategy complies with the ”clone-sets” idea already proposed in [83]. The second strategy is to reduce the service time of a resource/tier, which then leads to a faster processing of the queued requests at the affected tier. Finally, with the third strategy, the authors propose to reduce the amount of requests involving a saturated resource/tier.
The notion of potential *speedup* is mainly used in the area of parallelization. M. Kim, Kumar, H. Kim and Brett [85] use the potential speedup to estimate the benefit of a serial code parallelization effort. Donaldson, Berman, and Paturi [86] analyze the speedup in heterogeneous computing network. Since our context is a speedup analysis of homogeneous layered models, their speedup analysis is not adaptable to our context.
A more practical work is provided by Lilja [68], wherein several measures for expressing computer system performance are discussed, including the speedup measure. However, their speedup notion does not consider layered systems, which is addressed in this thesis.

2.7.3 MVA Algorithm Advancement

Applying the MVA algorithm onto large systems which additionally offer a mixture of operations, can easily lead to extensive computations in each step of the MVA algorithm. Therefore, several research efforts focus on deducing approximative and faster versions of the MVA algorithm. As one of the earliest work in this context, Bard [87] introduce several estimation techniques for the MVA algorithm, which can be summarized under the so called Approximate Mean Value Analysis (AMVA).
The core of these expensive computations (in terms of time consuming) is based on the recursive relation represented by equation (2.7.4). Hence, the performance quantities of a concurrency level of interest, require computations of all the previous concurrency levels, which can be formidable

expensive for high concurrency levels. Thus, AMVA techniques basically emphasize on fast to compute approximations for A_i^n. In principle, there are two different approximation approaches, where the first is based on the following:

$$A_i^n \approx L_i^n - \left(\frac{1}{n}\right) L_i^n. \tag{2.7.5}$$

The underlying assumption behind this approximation is that removing a single request does only lead to a slight change in the state of each queuing system of the overall queuing network. By making use of this relation, the L_i^n is iteratively approximated until a specific convergence threshold is achieved. The main advantage of this approximation approach are savings in the storage requirements for the relevant previous values. However, the savings in the storage are at the expense of the accuracy.
Thus, the second approach in turn is based on the following:

$$A_i^n = (n-1)\left(\frac{L_i^n}{n} + \gamma_i^n\right), \tag{2.7.6}$$

with

$$\gamma_i^n = \frac{L_i^{n-1}}{n-1} - \frac{L_i^n}{n}. \tag{2.7.7}$$

In this approach, there are iterative computations to approximate the γ_i^n terms, also until a specific convergence threshold is met. In contrast to the first approach, this algorithm reveals a significantly better accuracy, but also higher storage requirements.
Based on AMVA algorithms, Zahorjan, Eager, Sweillam [88] introduce the Aggregate Queue Length (AQL) algorithm, which is based on the second approach, however, with reduced storage requirements. In addition, the authors provide a comparison of these three approximation algorithms.
Wang and Sevcik [89] provide a software package (IAMVAL), which includes two more iterative approximate MVA algorithms: The Queue Line algorithm (QL) and Fraction Line (FL) algorithm. Both of these proposed algorithms are based on the relation (2.7.5). Moreover, in both algorithms, the emphasis is on improving approximations of specific terms of the algorithms. However, the computational overhead in both algorithms is approximately the same as the original approximative version of the MVA, which is based on relation (2.7.5). In general the QL algorithm is more accurate than the original approximation algorithm. Whereas the FL algorithm is either more accurate than QL and the original approximative version, or even less accurate than both of approximation algorithms.
In this context, Cremonesi, Schweitzer and Serazzi [90] present a framework, which allows a ”unified view of approximate solution techniques” for the MVA, which also includes all the aforementioned approaches. Under this framework, all ”known” approximations approaches for the MVA can be analyzed.
The work of Petriu and Woodside [91] is another comprehensive contribution for the area of approximative MVA algorithms. The goal here is to provide an approximate MVA solution for ”Markov models that represent a composition of components”. The focus of the authors is to derive approximation techniques for systems, which are considered from their processes levels, which requires a deep and detailed insight about the inter-process interactions and their corresponding parameters.

2.8 Summary and Research Delta

First of all, the performance aspect in the context of persistent identification is not covered at all or only addressed as a side effect by all the aforementioned research efforts.
The starting point of Schmitt and Majchrzak [54] is to enable the operation of the Handle System on top of a possible future Internet infrastructure: the NDN infrastructure. Their objective is to ensure the retrieval of legacy research datasets, which have been assigned Handle-PIDs, to be also intact in NDN. Moreover, they also propose a solution, which not only ensures the resolution, but also the management of Handle-PIDs over NDN. Only as a secondary objective, they also compare the performance for resolution and administration of Handle-PIDs within the traditional Internet and the NDN infrastructure. Their comparison reveal that the performance within NDN is significantly slower than within the traditional Internet.
All remaining works, related to PIDs, do only discuss their usage for the area of research data management.

Since in a certain sense, PID systems reflect the DNS system, we have also analyzed research efforts in the area of DNS. However, the primary purpose of DNS is to act as a global data lookup registry, rather than a global data management system. Therefore, the research efforts addressing DNS performance, in principle only capture the performance of the resolution operation. The main methodology here is to employ different caching techniques at various DNS servers. However, for PIDs it is also highly important to ensure a high-performant PID record management, which is addressed in this thesis.
To improve the resolution of PIDs, especially Handle-PIDs, however, we utilize the DNS system and its worldwide spreading. For the PID systems based on the Handle System, this can be done by equipping these PID systems by a DNS interface, which realizes the embedding of PID systems into the global DNS infrastructure. In a sense, the embedding approach of Schmitt and Majchrzak [54] is similar to our approach. They equip the PID systems with a NDN-interface, which enable the operation within the NDN infrastructure. However, with their approach, the resolution time becomes even significantly longer than with the native interface.

PID systems can also be considered as globally distributed databases for holding relevant information about research datasets. Therefore, we also analyzed performance-relevant works in the area of databases. In this context, bulking has proofed to be an efficient technique to accelerate (especially) data writing operations, which is also employed in thesis. However, the focus in this thesis is more on the respective PID system communication protocol used between a client and a PID system, rather than on the PID system database.
Another technique is to relax the ACID principle, which results in tremendous performance improvements in database operations. However, this comes at the expense of data consistency, which is essential for PIDs. Therefore, the relaxation of the ACID principle does usually not represent an overall solution for PIDs systems.

Furthermore, PID systems are meant for long-term operation, which therefore requires their improvement with technological advance. Hence, another view on PID systems reveals that they are basically transactional multi-tiered systems among various others available in the Internet. The literature for performance improvement for such systems is considerable rich. Therefore, in this thesis, we focus on the response time improvement of transactional multi-tier Internet systems. For this thesis, the work of Urgaonkar et al. [79] provides the basic approach for modeling multi-tier Internet systems are a queuing network and to apply appropriate performance analysis techniques. In contrast to Urgaonkar et al., we focus on the improvement of the response time of multi-tier systems and the resulting effects of improvement in respective tiers onto the overall

system's performance behavior. Urgaonkar et al. basically only emphasize on the aspect of appropriate modeling.
Closest to our approach is the work of Franks, Petriu, Woodside and Xu [84]. In contrast to our approach, the authors focus on detecting and relieving bottlenecks in layered systems to improve the throughput of the system. One strategy proposed by the authors is to reduce the service time of the resource/tier which causes the bottleneck. However, the effect of such an improvement in the service time onto to the overall system's response time is not analyzed by Franks, Petriu, Woodside and Xu. This, however, is provided in this thesis.
The core of our approach is to analyze of the response time behavior of the MVA algorithm. This approach enables to deduce certain performance measures which are appropriate to support the long-term improvement of multi-tier systems, such as PID systems.
All in the previous sections investigated research works addressing the MVA algorithm, do only emphasize to deduce faster but approximative versions of the exact algorithm. In contrast to that, we basically compare the application of the MVA algorithm on two systems, the old and improved system, and deduce formulas which capture the resulting load distribution effects caused by improving a specific tier.
Another aspect of the contributions to performance analysis, in particular the works related to the MVA algorithm, is that they require the implementation of a certain simulation algorithm, mostly the MVA algorithm or a variant of it. Although, our approach is fundamentally based on the MVA, it does not require the implementation of the MVA algorithm. Instead, from the behavior analysis of the MVA algorithm, we deduce estimation formulas, which can be used in combination with the tools provided by operational analysis to investigate a multi-tier Internet system.

Chapter 3

High-Performance Persistent Identification

In this chapter, we provide insight into the performance problem caused by persistent identification, which is based on an additional maintenance overhead composed of expensive transactional operations.
The starting point of the discussion is the analysis of the content access mechanism of the Internet. This serves the purpose of providing the underlying context, which initiated the introduction of PIDs. In this regard, we also consider the ideas behind the *Information Centric Networking* (ICN) paradigm, which show similarities with the concept of persistent identification. By this comparison, we reveal that durable links are a matter of continuous maintenance causing a certain overhead.
We finally propose a concept, which enables a significant reduction of the overhead required for maintaining PIDs to keep them valid (cf. Figure 3.1). This concept is especially suitable for a specific group of research data repositories.

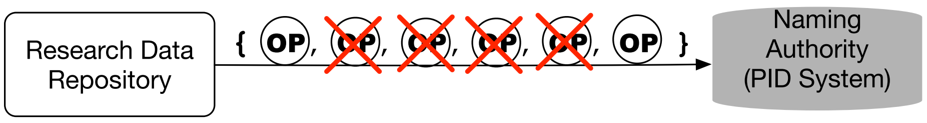


Figure 3.1: Focus in chapter: Reducing the maintenance overhead for research data repositories for administering PID records to keep them valid. A reduction of the maintenance overhead can be achieved by reducing the number of these expensive **OP**s (cf. Section 1.1), which is tackled in this chapter.

The base of our approach has been published in a conference paper at the 11th IEEE Networking, Architecture and Storage (NAS) conference 2016 in Long Beach, CA, USA [92]. In this thesis, however, we provide a revised and extended version.

3.1 Towards Persistent Access

In this section, we briefly describe the increasing importance of digital content. In addition, we provide an investigation of its consumption mechanisms within the Internet. We also provide insight into the fundamental challenge of persistent access within the Internet and explain the existing approaches for a durable content access and enlighten the role of PIDs in this context.

3.1.1 Internet Content Retrieval

The Internet is composed of myriads of interconnected nodes, where each node offers a diverse set of digital content in the form of services and datasets. The original function of the Internet is to establish a communication network among distributed computers. The first exchanged messages were small ASCII messages containing a few bytes of text. However, the amount of transmitted data has increased rapidly which in turn has led to the emergence of various information exchange technologies such as the World Wide Web, or simply the Web. In principle, the Web is an additional layer on top of the Internet, which enables a convenient access to content. Content in the Web is denoted as a *resource* and is addressed by a *Uniform Resource Locator* (URL), which is a locator.
To retrieve an individual Internet content, it is required to address the node on which the requested content is currently stored on, in combination with an additional access parameter serving as a local identifier. A URL, therefore, is a composition of a node identifier and an additional, locally unique content identifier (cf. Figure 3.6).
However, in the Internet, content usually does not reside on a specific node eternally. Instead, the storage node of content is often changing, leading to changing locators such as URLs. These changing URLs are the root cause for the so called link rot phenomenon [12].
Therefore, persistent access to content constitutes one of the major problems of the Internet which is also inherited by the Web, our current standard information exchange platform. Exactly this problem is the main reason for deducing PIDs and also the ICN paradigm.

3.1.2 Basic Idea of the Concept of Persistent Identification

In the previous subsection, we have identified the locator-based content access to pose a serious challenge for durable data retrieval. In this subsection, we analyze the basic idea behind PIDs, which represent a possible solution to tackle the problems induced by the address-based Internet architecture.
The basic idea behind the concept of persistent identification is twofold: first to provide persistent naming for digital objects, and second, to ensure persistent access to digital objects stored in the Internet. The common functionality of the various existing PID systems is to provide an identification layer which abstracts the locators of the digital objects, such that an individual digital object is solely identified by an immutable, opaque and globally unambiguous identifier. However, such a globally unique identifier has also to be *actionable* [20] [93] in the Internet, to enable the access of the identified data. Exactly this (continuous) actionability, makes a globally unique identifier to a real PID.
However, in particular the importance of the continuity of the actionability is often undervalued. Only this continuity results in persistent access. Persistent access to data in turn constitutes a formidable problem requiring elaborated and sustainable solution techniques. Therefore, the following subsection discuss the overall problem of persistent access in a broader context.

3.1.3 Persistent Access with PIDs and Information Centric Networking

The idea of PIDs has been mainly developed in the context of digital preservation of scientific datasets. However, the problem of persistent access has gained an increasing relevance with the recent explosively growing number of digital objects. Digital objects are now omnipresent. Therefore, the ICN paradigm has been proposed to provide efficient solutions for data-oriented communication including persistent access.
The goal of the ICN paradigm is to establish a new Internet architecture, which is specifically ori-

ented on content consumption rather than on node-to-node communication. In ICN, the focus is on the ”what” instead of the ”where”. In addition to content retrieval, the ICN approach also provides solutions for data integrity and security. The general idea is inspired by the publish-subscribe communication model: Data producers notify the network when new data is available for distribution. Data consumers in turn, subscribe to datasets in which they are interested. Furthermore, to solve the location-binding of content, in ICN, data is additionally stored at special caches within the network level. By means of these caches, the load at the originating data locations can become significantly reduced, also preventing Denial of Service (DoS) attacks [94].
Although there are several different approaches for ICN, they all employ similar techniques and components. The consumption of data is done in two steps: first it is localized, then it is routed to the requesting consumer. Therefore, the existing ICN approaches are always comprised of a name resolution and a data routing system. The name resolution system is used to localize a requested dataset, whereas the data routing system is used to efficiently transmit the localized dataset to the interested consumer.
One group of the existing ICN realizations employ *coupled* name resolution and data routing ([45], [46], [49], [50]), whereas another group employ *decoupled* name resolution and data routing ([47], [48], [51]). In the coupled approach, the reverse order of the nodes, which were traversed in the name resolution procedure, are used again to route the data to the consumer. In the decoupled approach in turn, data is routed through a different path, often a more efficient path for data delivery.
Another characteristic of the proposed ICN solutions is given by the composition of the data identifiers. In one part of the ICN solutions, the identifiers are hierarchical ([46], [50]), whereas in the other part, the identifiers are flat or quasi-flat ([45], [49], [47], [48], [51]). Flat means that the identifier does not incorporate any structure. The naming system of the proposed ICN solutions, which are based on flat or quasi-flat identifiers, can be considered as a quasi centralized global data registry, wherein all new datasets and the corresponding dataset movement operations have to be registered to keep track of the data locations. This, however, often causes a significant overhead in the global data registry. With hierarchical identifiers in turn, the global naming system becomes distributed leading the datasets to be registered at specific sub-registries, which are only responsible for a certain group of datasets. The global retrieval of an individual dataset is then accomplished by suffixing the local data identifier by the globally unique sub-registry identifier. However, a hierarchical identifier is basically a grouping mechanism, which again can aggravate the movement of individual datasets.

A major problem with the existing ICN initiatives is that they require a profound replacement of the current Internet architecture. All Layer-3 switches, currently used in the world, would have to be replaced by the new name resolution and routing systems. In contrast to that, existing PID systems function only as overlay networks on top the current Internet, without the need of changes in the underlying network.
On an abstract level, the existing PID systems employ pretty similar techniques as the proposed ICN solutions. The existing PID systems are pretty much inspired by the well-known DNS system. Hence, PIDs are composed of hierarchically ordered labels, but opposed to DNS domain names, without any semantic in the individual labels. In the DNS system and also in the PID systems, the overall architecture is built as a globally distributed database. In a global PID system, an individual database is usually denoted as a *naming authority* (local PID system), which is assigned a globally unique identifier. New datasets are then registered at specific naming authorities to be assigned with PIDs. A PID is then constructed by suffixing the globally unique naming authority identifier with a label, which is only unique in the scope of that specific naming authority. The result is a globally unique identifier assigned to an individual dataset. Note that the term *naming authority* was already introduced in the introduction chapter (Chapter 1) of this thesis.

To consume a dataset assigned with a PID, a preliminary request is always necessary to resolve the PID into its locator, which is then followed by a second request with the obtained locator to retrieve the actual content. This means that PIDs only enable indirect data retrieval. In contrast to that, in ICN, the data identifier enables a direct retrieval of the corresponding the content. However, apart from that, the PID concept is comparable to those ICN approaches, which employ decoupled name resolution and data routing:
On the one side, at the resolution procedure of PIDs, several nodes of the global resolution architecture are traversed until the responsible naming authority is found. The traversal path is determined by the hierarchically ordered labels composing the PID. The retrieval of the actual dataset by means of the obtained locator, is then carried out totally independently from the PID system. In terms of ICN, a PID system can be considered as a name resolution system. This means that there is already a name resolution system in our Internet architecture.
On the other side, data producers have to register all their datasets at a specific naming authority (local PID system) to enable their global localization. Controversially, also this has to be done with the ICN solutions.

Finally, although the ICN initiatives aim to established a data-oriented network, from an abstract perspective, their conceived network elements resemble again the components already used within the current Internet architecture. The most important result of this section is that persistent access is only ensured by a continuous tracking of data movement. More concretely, persistent access requires a persistent maintenance effort to ensure the validity of the name-to-location bindings, this holds true for the proposed ICN solutions as well as for PIDs.

3.2 Performance in Persistent Identification

So far, we have revealed that persistent access requires a persistent maintenance overhead. This, however, usually results in a performance problem, especially for PIDs. In this section, we examine the fundamental cause of the performance problem of existing PID systems. After concretely revealing their individual response times for PID record administration, we finally provide insight into our first proposal of this thesis, which is based on Data Silo Identifiers (*Research Data Silo Identifier*s (DSIDs)). Our proposed concept of DSIDs enable the reduction of the overhead caused by the maintenance of the PID-to-locator bindings.

3.2.1 Fundamental Performance Problem

The fundamental performance problem of existing PID systems from the perspective of a research data repository is based on the following performance issue:

- A persistent **maintenance overhead** for PID records to maintain them valid, which is composed of a set of **expensive transactional** administration operations (cf. Figure 3.1 or Figure 3.2: **OP**) issued against a particular naming authority (local PID system).

Figure 3.2 provides a vivid example for this: We can see a research data repository, which is continuously issuing administration requests against the ePIC PID system. The ePIC system is composed of several components including a Handle server (grey box), which represents a particular naming authority within the global PID system established by the Handle System. In the ePIC system, the response time of a single **OP** includes the processing times of all these components. And since a single **OP** is accomplished transactionally, it often leads to a performance problem for large amounts of PID records. A concrete example of a single **OP** is provided by Listing 3.1.

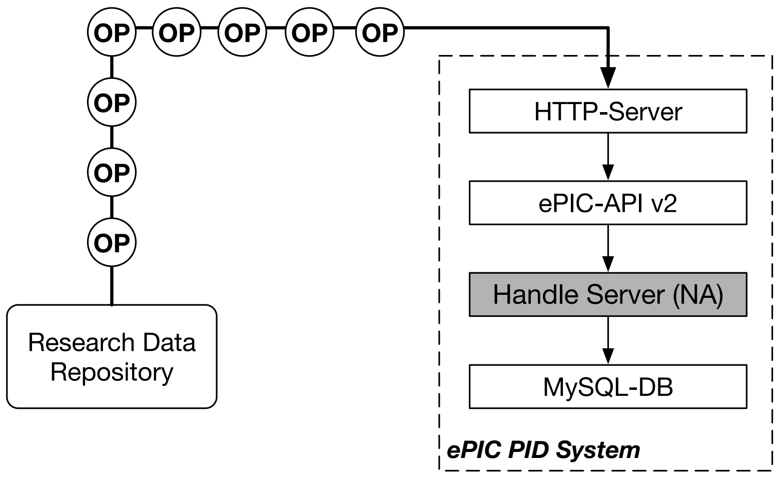


Figure 3.2: Research data repository administering PID records at the ePIC PID system. Again, each **OP** symbolizes an expensive transactional administration operation.

This listing illustrates a HTTP request example for creating a single PID at the ePIC PID system provided by GWDG.

```
POST /handles/21.T11996 HTTP/1.1
Host: pid.gwdg.de
Accept: application/json
Content-type: application/json
Content-length: 51

[{"type":"URL","parsed_data":"https://bitbucket.org/fatihbucket"}]
```

Listing 3.1: Example for a single **OP**: HTTP Request for creating a PID at ePIC system (v2) provided by GWDG. The resulting PID will point to the locator `https://bitbucket.org/fatihbucket`.

Since the existing PID concepts are commonly realized as overlay networks (cf. Figure 3.5), each such an overlay network requires the implementation of a particular protocol and/or data model to enable the internal communication, which is usually not supported by recent research data repositories. This is the reason for the need of naming authorities which comply with the respective protocol and/or data model.
Moreover, these naming authorities usually employ expensive transactional administration procedures to ensure the durability and consistency of the information associated with PID records.

There are two options for the choice of a naming authority, from which research data repository system designers are compelled to make a selection:

(a) Setting up an internal naming authority service.
(b) Making use of an external naming authority service provided by an external PID service provider.

Option (a) usually entails a significant implementation effort to enable the support of the required protocol and/or data model. In addition, this option requires an appropriate maintenance effort for the service itself to ensure its long-term operation.

Since in option (b) the naming service is already offered by an external service provider, there is no significant implementation effort required. In addition, the service operation effort is already covered by the service provider (such as GWDG). However, due to the network latency, the overhead of the administration procedure is usually even more increased with option (b).

The following subsection, provides a brief overview of various existing PID systems including their protocols and data models.

3.2.2 Description of Existing PID Systems

Note that the description of the various PID systems, is followed by a comparison of their response times for PID record administration.

Handle System: The Handle System, requires the implementation of the Handle protocol [95], which enables the integration into the corresponding global resolution system. In fact, this system can be also considered as an imitation of the well-known DNS system. Instead of computer host identification, the Handle System is used for research dataset identification and localization. A resolution request in the DNS system primarily results in a DNS Resource Record containing an IP-address of a computer host. In the Handle System for instance, it results in a Handle Record, which contains the corresponding locator of a research dataset. Originally, the Handle protocol has been designed to operate directly via TCP/IP and UDP/IP. A mature REST interface is only supported beginning with version 8 of the Handle System software.

Digital Object Identifier (DOI) System: The DOI system is the most prominent among the PID systems. It is technologically based on the Handle System. Initially, the DOI system has been developed on top of a previous version (<8.0) of the Handle System, which did not offer a REST interface. Therefore, the DOI system extends the Handle System by a complete REST interface. In addition to a REST interface, there is also a special function for collecting and processing metadata of registered research datasets. Thus, instead of functioning as a sole proxy between a research data repository and the Handle System, the DOI system additionally provides a special index of all registered research datasets. This special index is realized by an additional database. However, to register a research dataset at a DOI registration agency, a research data repository is required to map the research dataset into a specific data model [19]. This mapping is not just a transformation from one data model into another. Instead, it is a procedure to verify whether an individual research dataset is compliant to be included into the overall index. For that, an individual research dataset has to provide a mandatory set of key-value pairs, which are then mapped into the DOI data model. Usually, only research datasets which are already in a finalized state of the research data lifecycle [96], can provide this mandatory set of key-value pairs. Therefore, the special index of the DOI system usually contains research datasets, which have been published.

ePIC PID System v2: The ePIC system (version 2) [97] (cf. Figure 3.2) is another PID system, which is based on the Handle System. In principle, it is very much comparable to the DOI system, as it provides a REST interface and a special function for metadata processing. The essential difference is that the data model of ePIC is much more flexible than the DOI data model. The focus in ePIC is more towards machine readability of the Handle Records of the registered research datasets. Therefore, with the ePIC system, individual researchers can first define their own data structure(s) and then, register new research datasets, which in addition are compliant with the predefined data structure(s). This flexibility in the data model enables to register any type of research datasets at the ePIC system. In contrast to the DOI system, in ePIC there is no additional index database which has to be provisioned. Instead, the ePIC system directly operates with the database attached to the underlying Handle server. By means of this direct connection to the database of the Handle server, the ePIC system also provides a search functionality for searching directly in the registered set of Handle Records.

Archival Resource Key (ARK) PID Concept: Another established concept for persistent identification is realized by the ARK [98] concept. In comparison to the Handle System, the ARK identifier is much more lightweight. This identifier is in principle only a naming scheme, for which a set of URL-based conventions is defined. These conventions form the so called *HTTP URL Mapping Protocol* (THUMP) [99]. Therefore, to be integrated into the global resolution infrastructure of the ARK identifier system, a research data repository is compelled to implement the THUMP protocol.

Easy-Eye-Dee (EZID) PID System: A PID service provider based on the ARK identifier scheme is provided by the EZID system [100]. Similar to the PID systems based on the Handle System, this system provides an additional service on top of the underlying ARK PID system, such as a REST interface for the administration of the PIDs. Also the implementation of the THUMP protocol is covered by the EZID system. However, as with the Handle-based PID systems, the EZID system also requires a specific data model (ANVL[4]) to be provided for the registration of individual research datasets.

Uniform Resource Name (URN) Scheme: Often, the URN [31] scheme is counted to the set of PID concepts, however, it is much more a naming scheme than a real PID system. The common goal of URNs and PIDs is to provide persistent names for digital objects stored in the Internet. In contrast to PID systems, the URN specification does not provide a common globally widespread overall network to make URNs *actionable*. Among the registered URN namespaces, the *National Bibliography Number* (NBN) namespace is the most used namespace which can be listed within the existing concepts for persistent identification. It is particularly used in the area of libraries. However, even for the URN-NBN namespace there is no central resolution service, which is able to resolve all URNs of the NBN namespace. Instead, there are multiple resolution services, each responsible for a specific NBN sub-namespace. In addition, each of the registration agencies of the sub-namespaces define their own requirements for research datasets necessary for their registration.

These descriptions are summarized with Table 3.1.

Furthermore, Figure 3.3 shows the response times of different PID systems for administering a single PID record. The response time R_T represents the overhead of a single administration operation (cf. Figure 3.1 or Figure 3.2: $\textcircled{\textbf{OP}}$) offered by an individual (local) PID system to register or update a single PID record.

We can see that the most known DOI-PID system requires approximately 800*ms* to create a single DOI-PID. After registration, for DOI-PIDs, it then usually also takes about 24 hours for them to be globally resolvable.
Version 1 of the ePIC PID system does not allow an URL which is already assigned a PID to be assigned with another PID within its database. For this, it aims to ensure a bijective relation between

[4]https://confluence.ucop.edu/display/Curation/Anvl

PID System	Protocol	Data Model	Global Resolver	Identifier
Handle	HDL Protocol	Handle Record Handle Value	hdl.handle.net	hdl:PREFIX/SUFFIX
DOI	REST	DOI Metadata	dx.doi.org hdl.handle.net	doi:10.PREFIX/SUFFIX
ePIC	REST	ePIC Json	hdl.handle.net	hdl:21.PREFIX/SUFFIX
ARK	THUMP	-	n2t.net	ark:PREFIX/SUFFIX
EZID	REST	ANVL	n2t.net	ark:PREFIX/SUFFIX
URN	-	-	-	urn:NID:NID-SPEC

Table 3.1: Overview of existing PID systems.

PIDs and URLs. This however requires an additional duplicate check within the database causing a long registration time. In contrast to that, version 2 of the ePIC PID system (cf. Figure 3.2) only ensures a surjective relation between PIDs and URLs, which means that there is no such a duplicate URL check. Therefore, the second version of the ePIC PID system is significantly faster than its previous version. Note that its plotted response time ($250ms$) corresponds to the HTTP request, specified by Listing 3.1.
As already indicated, with version 8 of the Handle server implementation, there is also a built-in REST interface for PID record management. The PID creation times of the new REST interface and the native Handle protocol are labeled as ”Handle-REST” and ”Handle-LIB” respectively. We can also see that these interfaces provide the fastest response times for PID creation.

However, for research data repositories subjected to an immense data growth, such PID maintenance times are simply too slow. For example, the TextGrid[5] repository frequently registers up to 10,000 or more PIDs at once. With the measured response times, this means that the special PID registration module within the TextGrid repository would be blocked for approximately 1700 to 10,000 seconds.

To overcome the problems posed by the current PID systems, we propose a novel persistent identification concept. This novel concept focuses on the reduction of the maintenance overhead by reducing the number of the transactional administration operations, rather than on the acceleration of the processing time of the operations itself.
Since our concept is specifically suitable for a certain group of research data repositories, we first provide an abstract view onto recent research data repositories in order to distill their essential functionality. This is followed by a repository categorization, which is based on the different usages of PIDs in such repositories.

3.2.3 Abstract View on Research Data Repositories

An abstract view on a research data repository is illustrated in Figure 3.4. As can be seen from this figure, a research data repository can be considered as a research data silo, which offers the following five basic operations: *Create*, *Read*, *Update*, *Delete*, and *Search* and for short: CRUDS. Originally, the CRUD paradigm [101] has been established in the area of databases. In addition to the CRUD operations, many research data repositories also offer a search operation, which provides an aggregation of different datasets containing a common information pattern.
Recent research data repositories are usually additionally equipped with a built-in naming component, which generates opaque and immutable identifiers for the incoming datasets. Note that the

[5]https://textgridrep.org.de

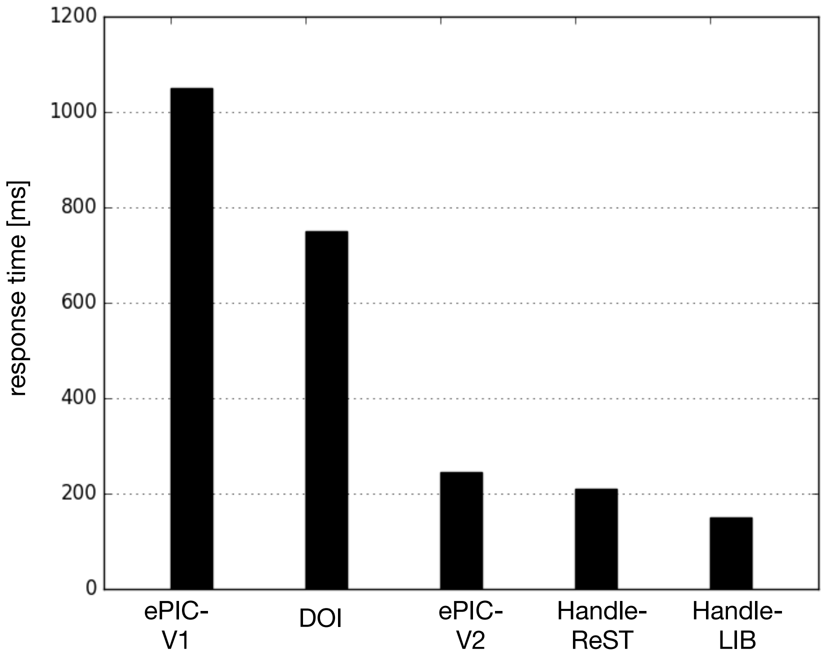


Figure 3.3: PID creation times for various PID systems.

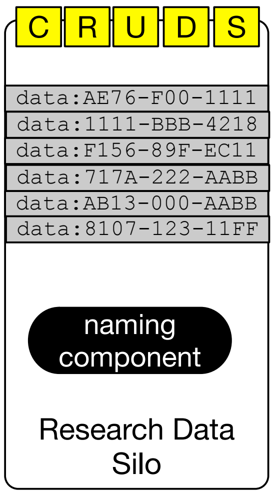


Figure 3.4: Abstract view on a research data repository offering a set of interfaces: **C**reate, **R**ead, **U**pdate, **S**earch (CRUDS). The stored datasets are usually assigned internal identifiers generated by a special built-in naming component.

research datasets are finally stored as regular files on the underlying storage system. The access to individual research datasets is then accomplished by using the corresponding file paths. The internal identifiers in turn, are used as logical names to abstract the file paths. However, these internal identifiers are only unique in the scope of the respective research data repositories. To enable a global retrieval of research datasets, it is necessary to provide the IP-address or the domain name of the research data repository in combination with the internal identifier and often also a further access parameter, which finally results in a volatile locator, such as a URL.

Based on the usage purpose of PIDs, contemporary research data repositories can be categorized into three groups:

- **Group (A):** In this group of research data repositories, PIDs are only used for locator abstraction.
- **Group (B):** In this group, in addition to a sole locator, the corresponding PID records are additionally imposed with semantic information about the research datasets.
- **Group (C):** In this group of research data repositories, the overall set of the corresponding PID records can be split into two subsets: A first subset consisting of PID records, which only contain sole locators. And a second subset of PID records equipped with semantic information in addition to a locator.

Our novel concept is particularity suitable for research data repositories belonging to group (A). Solutions for group (B) and group (C) of research data repositories are particularly addressed in Chapter 4 and Chapter 5.

Figure 3.5 depicts the situation of research data repositories (all groups (A),(B),(C)) registering each of their research datasets at particular naming authorities.

To reduce the management overhead for group (A) research data repositories, it would be much more efficient to directly enable the internal identifiers to be globally resolvable. This would eliminate the whole overhead for individual PID record management.
An abstract discussion of the global resolvability of internal identifiers is provided by the following subsection.

3.2.4 Global Resolvability of Internal Identifiers via DSIDs

In this subsection, we theoretically discuss how internal identifiers can become globally unique and resolvable.
Since existing global PID systems are composed of hierarchically ordered naming authorities, such as the DNS system, PIDs consist of hierarchical labels reflecting the ordering of the corresponding naming authorities. Commonly, a PID is composed of a prefix and a suffix, e.g. `21.11115/0000-000B-C8D9-F`. It is the prefix (e.g. `21.11115`) which uniquely identifies the responsible naming authority within the hierarchy of the overall global PID system. The suffix (e.g. `0000-000B-C8D9-F`) on the other hand, is a locally unique identifier assigned to an individual research dataset. Only by prefixing the locally unique suffix with a globally unique prefix label, the resulting identifier becomes globally unique. Therefore, the prefix plays an integral role in the global resolvability of identifiers.

For a research data repository assigned with a dedicated (prefix) label, by prefixing its internal identifiers with that label, these internal identifiers become globally unambiguous. To additionally enable these unambiguous internal identifiers to be globally additionally resolvable, requires a special locator composition rule associated with the prefix. This rule has to be used by the naming authority to compose the locator for the incoming identifier at a resolution request.

We define such a prefix, which is dedicated to an individual research data repository and associated with a specific locator composition rule to be a:

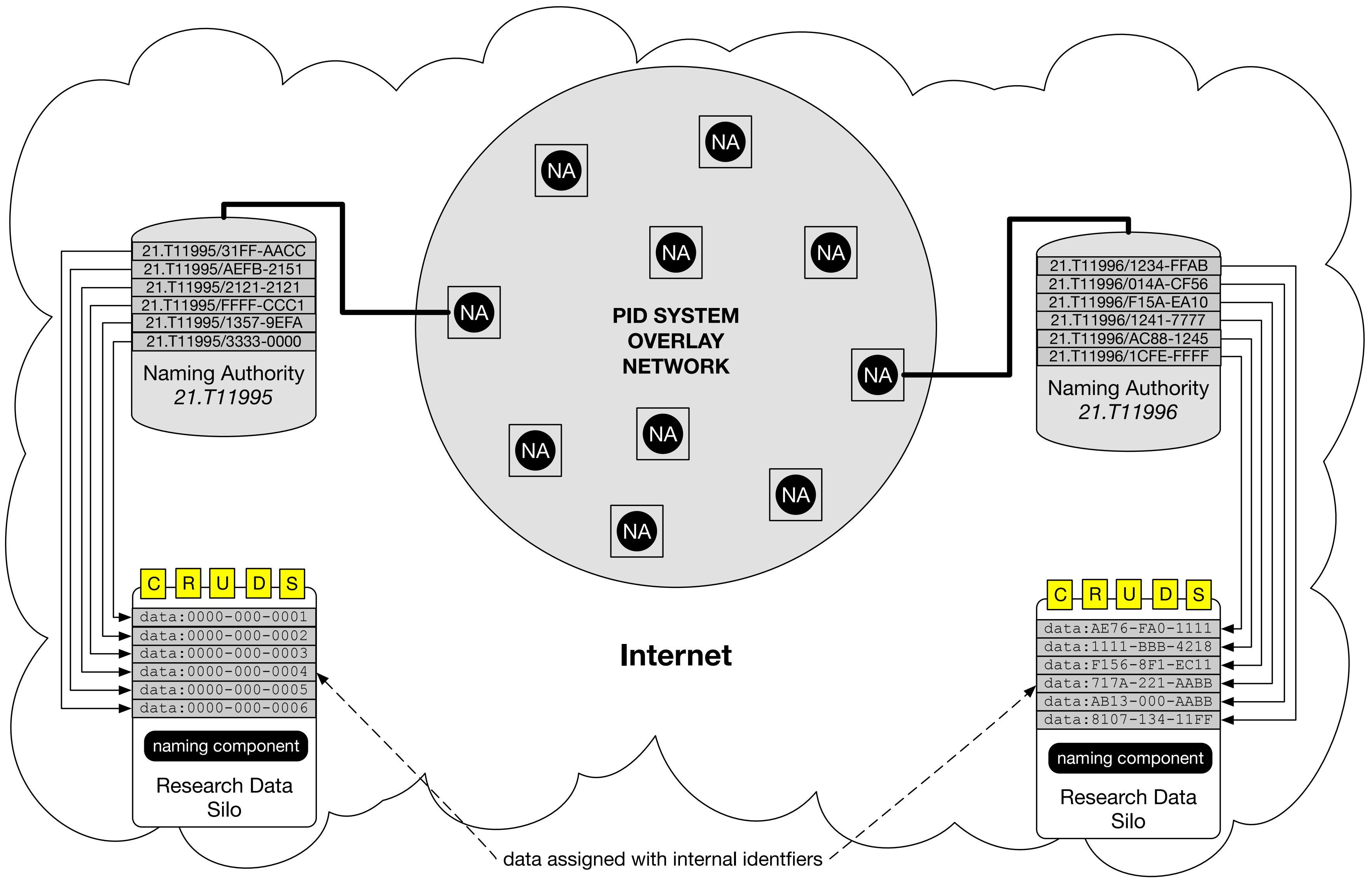


Figure 3.5: A global PID system composed of various naming authorities (NA) as part of the global Internet. These naming authorities (local PID systems) are used by research data silos to register their individual datasets to be assigned a PID. As part of the global Internet, each research data silo as well as naming authority is hosted on an ordinary Internet host addressed by an individual IP-address. The solid graphs represent the PID-to-data bindings, symbolizing a data access through PIDs.

- **D**ata **S**ilo **Id**entifier (DSID).

Hence, a DSID prefixed internal identifier would be global resolvable without being individually registered at a naming authority.

Finally, the technical realization of the DSID approach is discussed in the following subsection.

3.2.5 Data Access Interface Registration

The basic function of a regular naming authority is twofold: on the one hand to enable the administration of PID records and on the other hand, to respond to resolution requests revealing the locators. The resolution operation basically results in a lookup into its database for entries associated with the incoming PID string. For group (A) research data repositories, the most important entry within an individual PID record is the PID-to-locator binding. This entry in turn usually contains the following three information entities:

(a) The current Internet address of the research data repository.
(b) The current access request syntax used for retrieving datasets from that research data repository.
(c) The internal identifier of the research dataset stored in the research data repository.

The entities (a) and (b) compose the base part of the data access interface of a research data repository. This base part is usually included in all PID records associated with an individual research repository, which means that the information entities (a) and (b) are usually redundantly available in a database of a naming authority.
Therefore, for a group (A) research data repository assigned with a dedicated prefix, the DSID approach can be technically realized by associating the prefix with a composition rule based on the information entities (a) and (b). This requires only a single registration/update of the base part of the data access interface of the repository at a naming authority, which is then stored under the record of the DSID. The resolution request of identifiers prefixed by a DSID is then accomplished by dynamically extending the base part with the incoming information entity (c).

An example for a group (A) repository is the ARCHE repository[6]. Its datasets are registered at a naming authority, which is based on the ePIC PID system and therefore also on the Handle System. This naming authority is provided by the GWDG. Figure 3.6 illustrates an excerpt of the important entries (PID-to-locator bindings) from the database of this naming authority. In addition to that, Figure 3.7 depicts the complete entries of an individual PID. These entries form a Handle Record for one of these Handle-PIDs listed in Figure 3.6. The Handle Record is composed of three entities, where the first entry represent the most important one, since it contains the URL (locator). The second and third entries contain only internal information used by the ePIC PID system and the underlying Handle server for administrative purposes, which means that they do not hold any additional information specific to the identified dataset.

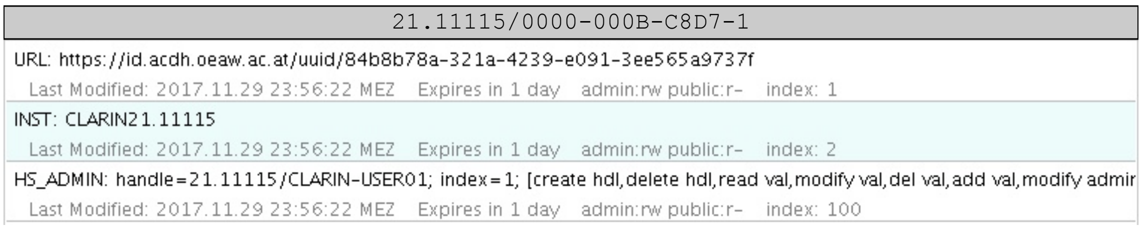
As can be see from Figure 3.6 and also from the first entry in Figure 3.7, the locators are composed of exactly the three entities: (a) `id.acdh.oeaw.ac.at`, which is the address of the repository, (b) `https://{···}/uuid/`, which specifies the current access request syntax and (c) an internal identifier, which is opaque and hence, appropriate to be used as a PID.

Instead of registering each individual research dataset, with the DSID concept, there is only an

[6]https://arche.acdh.oeaw.ac.at

PID	URL
21.11115/0000-000B-C8D5-3	https://id.acdh.oeaw.ac.at/uuid/1f0bbeff-343d-5081-a887-90b632b2a05f
21.11115/0000-000B-C8D6-2	https://id.acdh.oeaw.ac.at/uuid/a80553c0-6be1-6a7f-a63b-16993060302d
21.11115/0000-000B-C8D7-1	https://id.acdh.oeaw.ac.at/uuid/**84b8b78a-321a-4239-e091-3ee565a9737f**
21.11115/0000-000B-C8D8-0	https://id.acdh.oeaw.ac.at/uuid/e1fc9dfe-b8c0-e9f3-9115-ae48421fb2ff
21.11115/0000-000B-C8D9-F	https://id.acdh.oeaw.ac.at/uuid/d62ca450-5693-6c45-c5b0-fe6890e366fc
21.11115/0000-000B-C8DA-E	https://**id.acdh.oeaw.ac.at**/uuid/cca1beff-5c9e-4223-970e-7faf7b5580a3
21.11115/0000-000B-C8DB-D	https://id.acdh.oeaw.ac.at/uuid/69bf3f5f-46d2-6296-dc98-a24d299ce9dc
21.11115/0000-000B-C8DC-C	https://id.acdh.oeaw.ac.at/uuid/c47f12a0-e05f-d2ea-a7f9-e4210e3f8df0
21.11115/0000-000B-C8DD-B	https://id.acdh.oeaw.ac.at/uuid/0e33f785-5ff5-55f2-f6f3-efea0453833d
21.11115/0000-000B-C8DE-A	https://id.acdh.oeaw.ac.at/uuid/72045ebc-b726-f10d-c3a4-4816abaa9d46
21.11115/0000-000B-C8DF-9	**https://**id.acdh.oeaw.ac.at**/uuid/**47841bc7-086d-ec40-978d-bb992e72a03b
21.11115/0000-000B-C8E0-6	https://id.acdh.oeaw.ac.at/uuid/37ff0e18-3197-7bcb-9d59-11aa6bd8a78c
21.11115/0000-000B-C8E1-5	https://id.acdh.oeaw.ac.at/uuid/830106dd-6494-7a4b-d455-432b6d05660e
21.11115/0000-000B-C8E2-4	https://id.acdh.oeaw.ac.at/uuid/53274ca7-51b8-cfc0-dd59-9f284254c5c7
21.11115/0000-000B-C8E3-3	https://id.acdh.oeaw.ac.at/uuid/b929ed98-a354-d82b-e733-d01ba5ae3b29
21.11115/0000-000B-C8E4-2	https://id.acdh.oeaw.ac.at/uuid/60e5d6aa-e48a-e298-c7eb-0f22e2a24745
21.11115/0000-000B-C8E5-1	https://id.acdh.oeaw.ac.at/uuid/f440a719-17c0-ed14-f091-d8c4331c09d7
21.11115/0000-000B-C8E6-0	https://id.acdh.oeaw.ac.at/uuid/07188f15-2948-73c0-8bc4-e3eadb91a072
21.11115/0000-000B-C8E7-F	https://id.acdh.oeaw.ac.at/uuid/bbe76c68-48ff-fe96-e638-ae2001e2de65
21.11115/0000-000B-C8E8-E	https://id.acdh.oeaw.ac.at/uuid/fd428ae8-0ed1-c285-a0bc-96d0bb759874
21.11115/0000-000B-C8E9-D	https://id.acdh.oeaw.ac.at/uuid/b7e43e5f-42a2-619d-d3de-5f04edd88900

Figure 3.6: Excerpt of database entries of a naming authority hosted by GWDG.
(a) address of repository, (b) access request syntax, (c) internal identifier



21.11115/0000-000B-C8D7-1

URL: https://id.acdh.oeaw.ac.at/uuid/84b8b78a-321a-4239-e091-3ee565a9737f
Last Modified: 2017.11.29 23:56:22 MEZ Expires in 1 day admin:rw public:r– index: 1

INST: CLARIN21.11115
Last Modified: 2017.11.29 23:56:22 MEZ Expires in 1 day admin:rw public:r– index: 2

HS_ADMIN: handle=21.11115/CLARIN-USER01; index=1; [create hdl,delete hdl,read val,modify val,del val,add val,modify admir
Last Modified: 2017.11.29 23:56:22 MEZ Expires in 1 day admin:rw public:r– index: 100

Figure 3.7: Handle Record of a dataset stored in the ARCHE repository.

administration of the information entities (a) and (b) within the record of the prefix `21.11115` required.
Often, there are research data repositories, which already contain huge amounts of research datasets without being previously registered at a PID system. For these research data repositories, which also belong to group (A), with this concept, all the contained research datasets get assigned with a PID by a single registration. Therefore, the DSID approach can lead to an enormous reduction of the maintenance overhead for PID administration.
Note that a concrete technical implementation of the DSID approach is only provided in the implementation section, Section 3.3. In contrast that, the following subsections discuss limitations and side effects of this approach.

3.2.5.1 Tracking of Individual Research Datasets

However, the movement of individual research datasets can not be tracked with this approach. Although the DSID approach is specifically addressed for group (A) research data repositories, it can be possible that specific datasets move to another research data repository. In order to additionally sustain the ability of tracking the movement of individual research datasets, for a naming authority, it is necessary to employ two different resolution functions: The first function is an ordinary lookup for an entry match for the incoming PID in its database. The second function is to apply the special locator composition function, which is specified in the DSID record. In order to respond a resolution request, the naming authority always has to start with the ordinary database lookup first. If there is no matching entry found, the special composition function is applied, provided that there is a locator composition rule in the corresponding prefix record. Otherwise, the resolution request is responded by an error message indicating that a matching entry could not be found in the database.
The overall resolution algorithm of a naming authority is depicted by Figure 3.8. The greyed area represents the special function block for resolving DSID-prefixed PIDs. This special function block is incorporated into the regular resolution algorithm path between the red and blue dots. The black dashed line, denotes the processing path of the original resolution path.

Another issue is the verification of the existence of an individual PID record. Provided that the DSID record of an individual prefix is associated with a locator composition function Handle Value, if the extended resolution path would directly lead from the red dot to the yellow dot (grey dashed line), without involving the green box, the resulting resolution algorithm would always return a positive answer. Even when there is no individual PID record stored in the database associated with an individual PID. Therefore, the green box represents a mechanism, which enables to distinguish between an individually registered locator and a dynamically generated locator.
For a dynamically generated locator, the existence of the corresponding dataset can be ultimately verified by the respective research data repository:
The naming component (cf. Figure 3.4) of the research data repository, which generates the internal identifiers, usually also records additional information about the assigned identifiers. Hence, by means of this additional information, it is possible to confirm the validity of the suffixes appended to the corresponding DSID.

3.2.5.2 Metadata Associated with PIDs

Another shortcoming of the DSID approach is related with metadata. As already indicated in Section 3.2.3, PID systems usually also provide the possibility to retrieve individual metadata associated with the identified research dataset. The metadata is often included in the respective

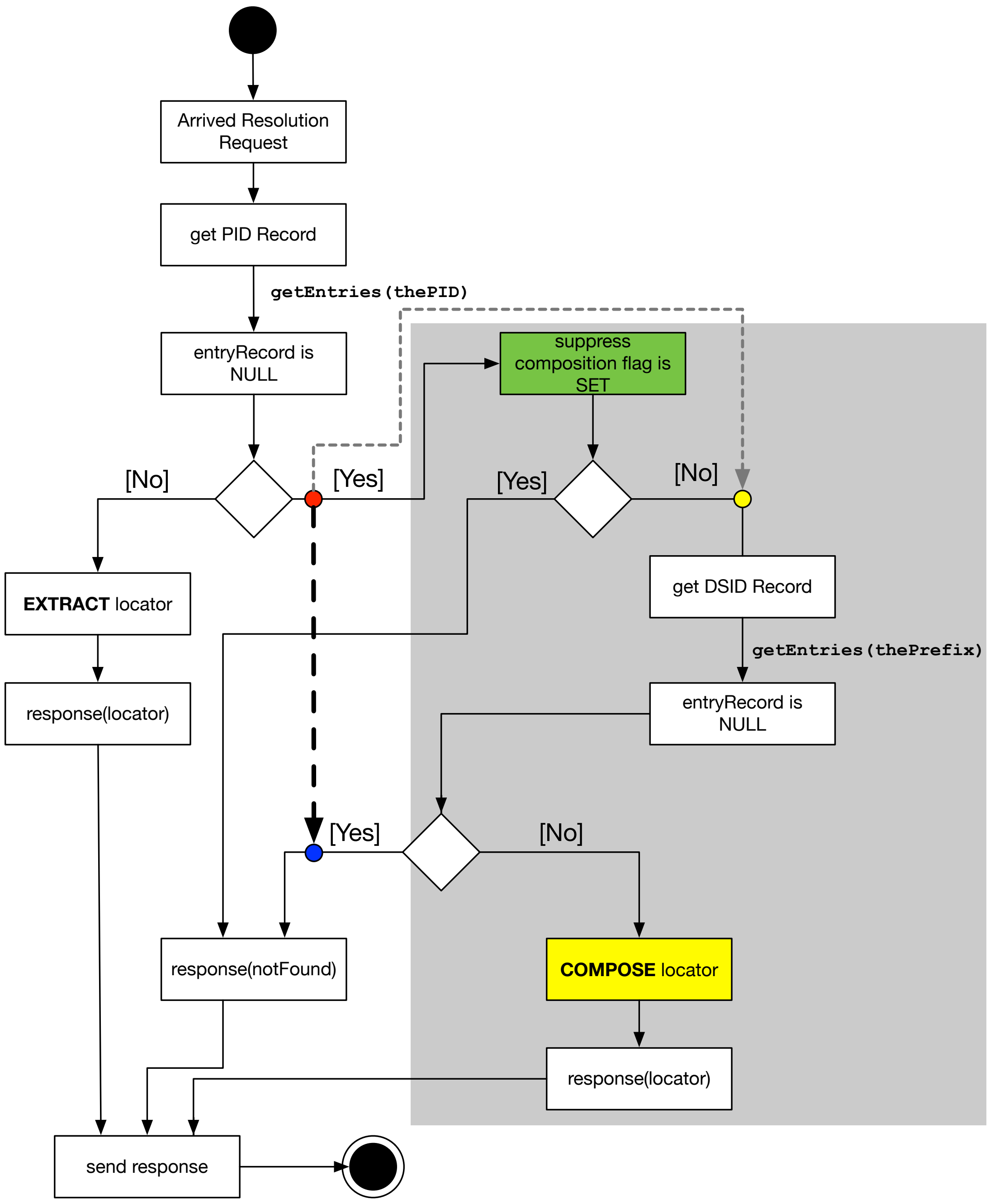


Figure 3.8: Naming authority resolution algorithm: The greyed area represents our extension to realize the DSID resolution approach. In this area, the yellow box represents the core function of our extension. The function represented by the green box is used to suppress the execution of the locator composition path.

PID records in addition to the sole locators. The DSID approach does not comprise individual metadata. Therefore, this approach is not appropriate for group (B) research data repositories.
The following subsection discusses a combined usage of DSIDs with regular PIDs for group (C) research data repositories.

3.2.6 Combined Application of DSIDs with regular PIDs

For research data repositories which belong to group (C), the DSID approach can be used for a subset of the contained datasets, for which the PIDs are only required for locator abstraction. For the remaining subset of datasets, which also require the imposition of semantic information into the PID records, the datasets can be registered individually at the naming authority. The resolution of both subsets of PIDs would be then accomplished as described in Section 3.2.5.1: First function, to resolve individual PID records. Second function, used to dynamically compose the locators for the incoming identifiers.
Furthermore, it is also possible that the first subset is composed of datasets, which are only in an early phase of the research data lifecycle. And with an increasing level of the research data lifecycle, the individual metadata might become more important. In this case, the research datasets, which were previously identified with DSIDs, can then be registered individually at the naming authority. This would lead the research dataset to be moving from the first subset into the second subset of datasets.

In summary this means that the DSID approach is reversible: datasets identified with the DSID approach can be always identified (again) with individual PIDs.

3.2.7 Search Interface Registration

The approach of data interface registration is in principle an approach to register a specific service endpoint at a globally interconnected naming authority. Such a naming authority can then be considered as a system, which provides a common service endpoint on top of the underlying research data repositories.
In addition to a service endpoint for research dataset retrieval, this approach also enables a service endpoint for searching. Since PID systems can also be considered as central data silo registries, these PID systems can also be used as search engines on top of the registered research data repositories.
Therefore, a side effect for our DSID approach is that a naming authority can also be equipped with a special searching functionality. Such a naming authority would then act as a federated search engine on top of all registered research data repositories. A search query submitted to such a naming authority would then be delegated to all the registered research data repositories.
Technically, this can be realized by the specification of the individual search interface at the one-off registration of the research data repository. However, this requires further research to develop appropriate filtering and mapping strategies at the naming authorities, which is not the scope of this thesis. The challenge is to elaborate mechanisms, which provide a sophisticated delegation on the relevant research data repositories.

3.3 Implementation

In order to provide a realization of our concept of *Data Access Interface Registration*, in this section, we examine the possibilities of an implementation into the Handle System. The implementation consists of modifying the resolution algorithm in the overlay network formed by the

Handle System to comply with the algorithm depicted in Figure 3.8. Therefore, we proceed with a quick introduction into the composition of the Handle System.

3.3.1 Handle System Overlay Network

The components of the overall Handle System can be categorized into three parts: *Global Proxy Resolver* (GPR), *Global Handle Registry* (GHR) and *Local Handle Service* (LHS). A LHS and the GHR are basically naming authorities consisting of one to many so called Handle servers. The entirety of Handle servers form an overlay network on top of the current Internet. The communication between and with these Handle servers is carried out through the Handle protocol [95]. The GPR in turn, consists of special servers, functioning as proxy servers which enable the translation of HTTP-requests into native Handle protocol resolution requests. Figure 3.9 depicts the overall retrieval procedure for a research dataset assigned with a Handle-PID. At first, to resolve the Handle, a requesting application (App) has to send a HTTP-request to the GPR (①), which first contacts the GHR ((②)) via the Handle protocol to determine the responsible LHS (③). This is followed by a request to the determined LHS (④). The LHS in turn, responds with a Handle Record including also the current locator of the research dataset (⑤). Finally, with the response from the LHS, the GPR sends back (only) the current locator of the identified research dataset (⑥). To retrieve the complete Handle Record, it has to be specified by the parameter ”?noredirect” appended to the requested Handle-PID.

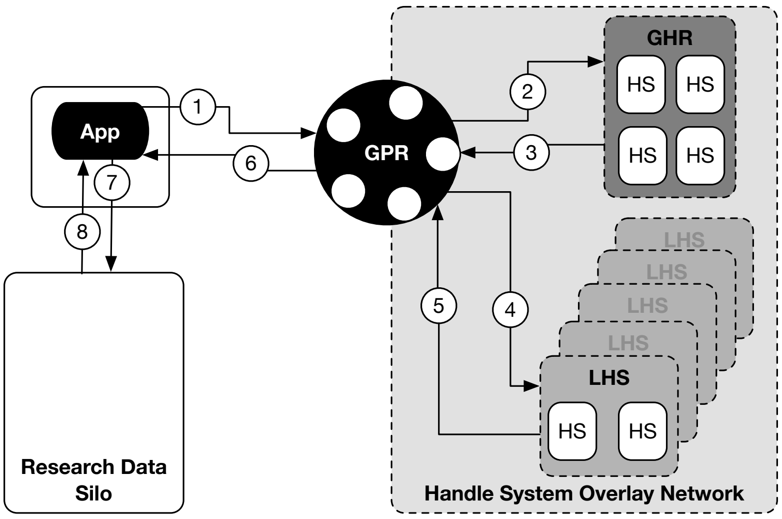


Figure 3.9: Retrieval of research data assigned with Handle-PIDs: Step ① to ⑥ for resolving the PID. Step ⑦ to ⑧ for retrieving the actual research dataset.

It should be noted that Figure 3.9 only illustrates the levels of the prefix hierarchy (GHR and LHS), representing a great majority of Handle-PIDs. However, it is also possible that a LHS only provides a referral to another LHS instead of returning the requested Handle Record.

The GHR represents the root level of the hierarchical structure of the Handle System. It only responds with a referral to the (next) responsible LHS. Therefore, a query submitted to the GHR is usually responded with a Handle Record containing an IP-address and a port number of a LHS.

Currently, the GHR is under control of DONA[7] and consists of eight[8] globally distributed special Handle servers (HS). Each of these special Handle servers in turn, is operated by so called Multi-Primary Administrators, which are member data centres of DONA. GWDG is one such a member.

A LHS is composed of at least one primary Handle server (HS) and none to many mirror Handle servers. A single LHS stores all the Handle Records under a specific set of prefixes. These prefixes are also called to be ”homed” at that LHS. A LHS can also function as a parent for other LHS. A parent LHS then responds with Handle Records containing IP-addresses and port numbers of the (next) responsible LHS. The administration of Handle Records is always done at the primary Handle server. The mirror Handle servers of the LHS are only used for resolution to distribute the load from the primary Handle server and ensure high availability. The mirror Handle servers periodically poll the primary Handle server for data synchronization. In contrast to the single GHR, there are numerous LHS, each under control of another organization or PID service provider.

The GPR currently consists of five globally distributed special proxy resolvers accessible through the `hdl.handle.net` domain name, where the selection is based on the DNS round robin mechanism.

3.3.2 Implementing into Handle System

To implement the resolution algorithm for our DSID approach, the regular resolution algorithm has to be extended by a special locator composition functionality. This special composition functionality is represented by the greyed area in Figure 3.8.
All components of the Handle System overlay network are based on a common core software package, which incorporates a Java reference implementation of the Handle protocol. This core software package implements the entire operation mechanism of the Handle System including the resolution algorithm for Handle-PIDs. Therefore, to realize our proposed concept with the Handle System, it has to be implemented into this core software package.

In the Handle System, there are two different options for performing the locator composition functionality. The first option is to make use of an already built-in technique, whereas the second option is to extend the core software package of the Handle System. In the following subsections we provide insight into both options and discuss their benefits and drawbacks.

3.3.2.1 Option (A): Making use of a pre-defined `HS_NAMESPACE` Handle Value Type

First of all, this option is based on the ordinary approach of setting up a LHS and homing a DSID prefix dedicated to an individual research data repository. To enable the locator composition function, we make use of a special Handle Value with the `HS_NAMESPACE` type, which belongs to the set of pre-defined types of the original Handle protocol. Originally, the `HS_NAMESPACE` type has been conceived to identify sub-parts of a single research dataset. Therefore, the data field of such a Handle Value type allows to define XML-based extension rules, which can be applied onto a particular base Handle-PID, which is usually an ordinary Handle-PID composed of a prefix and a suffix. Within the database of the LHS, the Handle-PID is associated with a specific set of Handle Values, which are then used to dynamically compose individual Handle Records for the sub-part of a research dataset. The identification of individual sub-parts is achieved by appending additional parameters on the Handle-PID. However, since only the base Handle-PID is associated

[7]https://www.dona.net
[8]https://www.dona.net/mpas

with a Handle Record, the LHS has to be able to recognize the base of an extended Handle-PID. This recognition of a base Handle-PIDs is specified in the data field of a HS_NAMESPACE-typed Handle Value, which first specifies the delimiter and secondly, the extension rules to be applied on the Handle Values of the base Handle-PID. The resulting Handle Values are then a composition of the Handle Values already stored in the base Handle Record including the extension parameters appended to the base Handle-PID.
Example: Assume a HS_NAMESPACE-typed Handle Value specifying the "@" character to be the delimiter. Further assume that the base Handle-PID 21.T11996/A105C-FC251 is associated with a Handle Record stored in the database of the responsible LHS. Then, the resolution request for 21.T11996/A105C-FC251@12345 is accomplished such that the extension 12345 is used as an input parameter for the XML-based rules defined in the data field of the HS_NAMESPACE-typed Handle Value. These rules are then applied onto the Handle Values associated with Handle-PID. For further information and more examples we refer to the technical manual[9] of the Handle System, defined under the section of "Template Handles".

This ultimately means, that in the core software package of the Handle System, there is already a built-in and generic enough composition functionality. For our approach, within the HS_NAMESPACE-typed Handle Value, it is necessary to define the "/" character to be the delimiter. In addition, the data data field of this Handle Value type has to specify the current data access interface of the corresponding research data repository. With the "/" character as the delimiter, the suffix becomes the extension used as an input parameter to be included into the specified current data access interface.
The specification of the HS_NAMESPACE has to be stored at the GHR within the DSID Handle Record. This is then retrieved by the LHS at the resolution process between step ④ and ⑤ in Figure 3.9.
By using this Handle Value type, a LHS dynamically composes the current locator of a research dataset based on the definition stored in the data field and the incoming internal identifier appended to the base Handle-PID.
However, with this approach, the LHS would always respond with a Handle Record, even when there is no associated Handle Record stored in the database for an individual Handle-PID. This means that a research data repository, which for example requires the pre-check of the existence of an individual Handle Record, will always get a positive answer (nonempty Handle Record). Therefore, we additionally propose to extend the native resolution operation of the Handle protocol by an additional flag, which enables to suppress the dynamic Handle Record generation. This can be used to determine if there is an individual Handle Record really stored in the database. This extension is represented by the green box in Figure 3.8.

3.3.2.2 Option (B): Making use of the new HS_RDS_URL Handle Value Type

Another option is to introduce a new type into the pre-defined set of Handle Value types: the HS_RDS_URL type. The introduction of such a new Handle Value type is based on the resolution procedure of the Handle System:

The first reason is that to resolve an extended Handle-PID for which the extension mechanism is specified by the HS_NAMESPACE type, there is always a LHS required. This means that the resolution request has to be always delegated from the GPR to the LHS, except for Handle-PIDs which have been previously resolved, since these are usually cached at the GPR. Hence, the resolution procedure is the same as for regular Handle-PIDs.

[9]http://www.handle.net/hnr_documentation.html

In contrast to that, with the new HS_RDS_URL type, the composition of the locator can be performed right at the GPR without the necessity to delegate the resolution request to the LHS. As it is the case with the HS_NAMESPACE type, the specification of the new HS_RDS_URL type is also stored at the GHR within the DSID record. This also enables to cache the HS_RDS_URL-typed Handle Values at the GPR used to compose the locators without involving the GHR. This finally means that with the new type, in addition to an improved PID assignment mechanism, we can also achieve an improved resolution performance by shortening the delegation procedure.
Another major benefit with the new type is that, since there is no need for a LHS, there is also no additional maintenance overhead for the operation of a LHS necessary.

The second reason is to enable a combined usage with option (A). This is for example necessary to enable the tracking of individual research datasets when they move out of the research repository. Or, to apply the DSID approach in combination with individual PIDs for group (C) research data repositories.

Since there is already a built-in locator composition functionality in the core software package which can be directly used at the GPR, there is no need to implement a separate composition algorithm for the GPR. And since the existing composition algorithm is based on XML, the data field of the new HS_RDS_URL type has to contain the same XML extension rules as for the HS_NAMESPACE type. This means, that these types are used to distinguish between the application of option (A) and option (B). Finally, for a DSID Handle Record containing a HS_NAMESPACE-typed Handle Value, the locator composition is accomplished by the LHS, where with a HS_RDS_URL-typed Handle Value, it is performed directly at the GPR. This, however, requires an additional mechanism at the GPR to be able to employ option (B).

3.3.2.3 XML Locator Composition Rule

For the ARCHE repository, an appropriate XML locator composition rule is given by Listing 3.2. Line 2 of this listing is specifying the slash character "/" to be delimiter between the DSID and the internal identifier. The actual composition of the locator is specified in line 3. As already pointed out, the successful resolution operation in the Handle System always results in a Handle Record composed of Handle Values. Since a Handle Value in turn is basically composed of a type-data pair, the result of the locator composition has to be a Handle Value with a type field specifying the type of the locator, which is currently usually a URL. The corresponding data field in turn, has to contain the actual locator.
For the ARCHE repository, we can first of all see that the locators are URLs. We can also see that these locators are composed by `https://id.acdh.oeaw.ac.at/uuid/${extension}`, where the `${extension}` expression is used as a placeholder for the internal ARCHE data identifiers. Hence, the locators of the ARCHE repository do not require the DSID to be included in the locator composition. Otherwise, this would have to be specified with the `${base}` expression representing the DSID label.

```
1 <namespace>
2   <template delimiter="/">
3     <value data =
4      "https://id.acdh.oeaw.ac.at/uuid/${extension}" type="URL"/>
5   </template>
6 </namespace>
```
Listing 3.2: XML Composition Rule for ARCHE Respository

Currently the datasets stored in the ARCHE repository are only registered by the regular approach under prefix `21.11115`, which is under control of a (naming authority) LHS provided by GWDG. If our proposed concept would be applied for the ARCHE repository, the Handle-PID `21.11115/0000-000B-C8D7-1` for example, would become to `21.11115/84b8b78a-321a-4239-e091-3ee565a9737f`, without being registered at that LHS.

3.3.2.4 Resolution Algorithm at GPR

The resolution algorithm, which is depicted by Figure 3.8, is only appropriate for the naming authorities. For the GPR, however, it is necessary to implement another algorithm to perform the locator composition with the `HS_RDS_URL`-typed Handle Values. Such an appropriate algorithm for the locator composition is illustrated by Figure 3.10. Again the red and blue dots indicate the extension points to the regular resolution procedure. Thus, at the GPR to resolve a Handle-PID, it first retrieves the prefix Handle Record of the responsible LHS. When there is a `HS_RDS_URL`-typed Handle Value in that prefix Handle Record, the resolution procedure continues with the locator composition based on the rule in the corresponding data field. However, if there is no such a Handle Value indicating the locator composition approach, the regular resolution procedure is executed. To force a real lookup for an individual Handle Record, we additionally introduce the parameter ”`?nocomposition`”, which is appended to the HTTP resolution request sent to the GPR. With the `HS_RDS_URL` approach, if this parameter is set, the GPR immediately responds with a negative answer (”Handle Not Found”), whereas with the `HS_NAMESPACE`, the special flag to suppress the dynamic locator composition is set in the resolution request to the LHS (Figure 3.9: step ④).
Finally, our extension of the original core software package of the Handle System can be found at [102]. Note that this implementation also includes the algorithm for the `HS_NAMESPACE` approach.

3.4 Evaluation

In this section, we provide an experimental evaluation of the two DSID approaches discussed in the previous section. Since with these approaches, there is no need to register and update individual PIDs, the evaluation is limited only to the resolution operation of PIDs, which are composed by concatenating the DSID with an internal identifier, such that `PID = DSID / INTERNAL_IDENTIFIER`. Note that the acceleration of the resolution of PIDs, which are associated with individual PID records is comprehensively addressed in Chapter 6.
Hence, the response times of the administration for individual PID records, depicted in Figure 3.3, remain unimproved. However, our proposed approach results in the effect that these expensive administration operations have not to be applied for each individual PID record. Instead, only when the research data repository is subjected to modifications affecting its data access interface, the corresponding DSID record has to be transactionally updated. However, also the tracking of individual research datasets still requires with option (A) transactional administration procedures. The performance improvement for individual PID records is addressed in Chapter 4 and Chapter 5.

3.4.1 Evaluation Setup

The evaluation setup is depicted by Figure 3.11 and includes several components. Therefore, we proceed with a description of these components:

Testing GPR: Since the approach with the `HS_RDS_URL`-typed Handle Value requires a modified

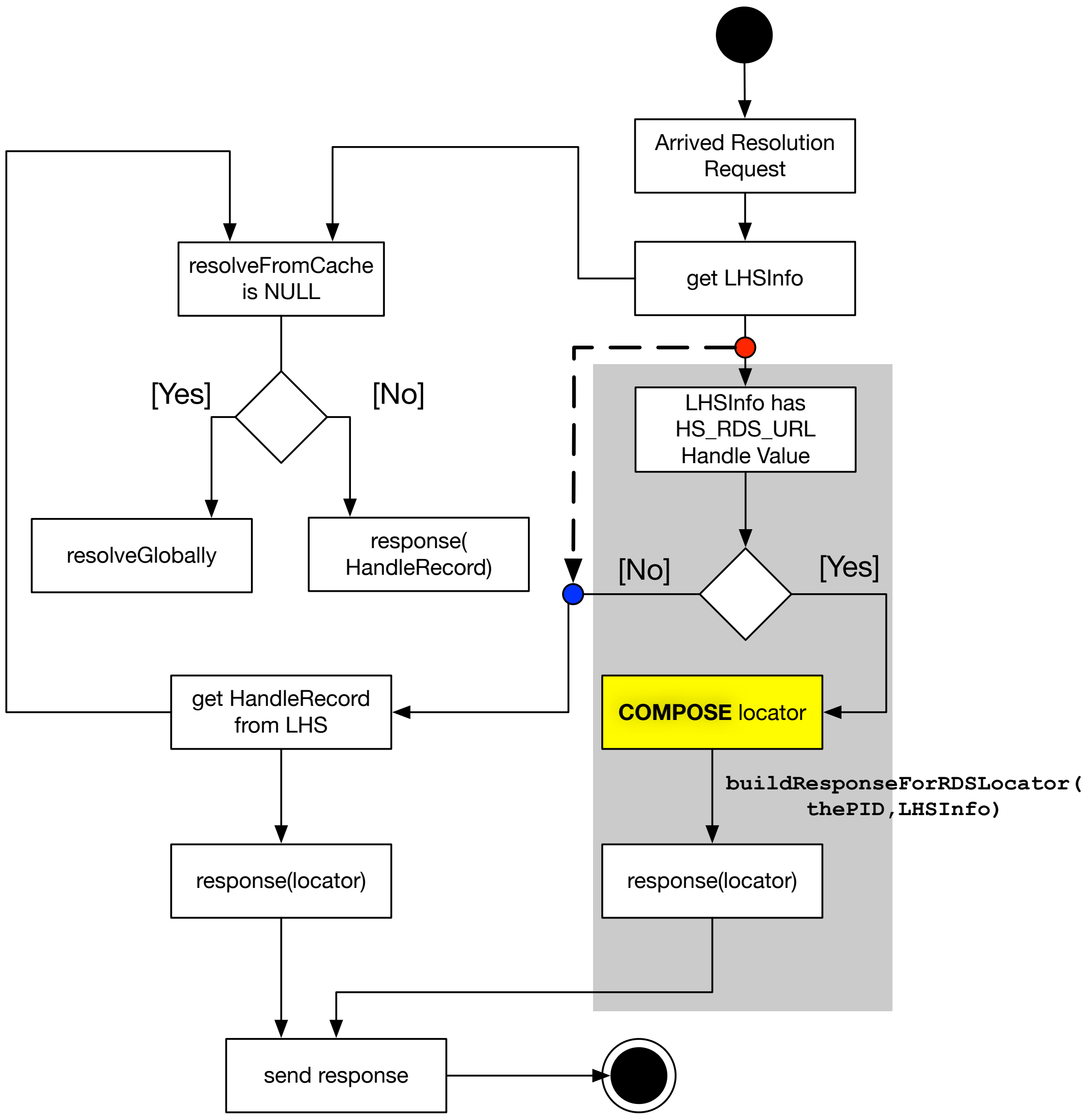


Figure 3.10: GPR resolution algorithm: The greyed area represents our extension to realize the DSID resolution approach at the GPR. Again, the yellow box represents the core function of our extension.

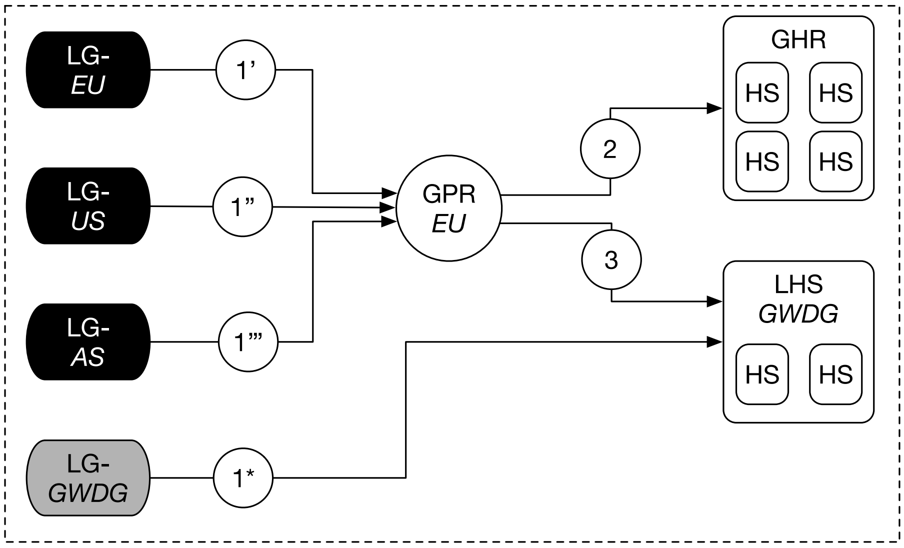


Figure 3.11: Evaluation Setup

processing at the GPR, we placed a testing GPR at an Amazon EC2 c3.xlarge instance in Ireland. In Figure 3.11, this testing GPR is labeled as *GPR-EU*. This testing GPR was equipped with our extended core software package of the Handle System [102].

Productive GHR: The productive GHR was used in the usual way for storing the `HS_RDS_URL`-typed Handle Values into a prefix Handle Record. For this evaluation, we used the `0.NA/21.T11996` prefix Handle Record.

Testing LHS: The LHS, responsible for the `21.T11996`-prefix, was installed at GWDG. It was composed of a primary and a mirror Handle server. This LHS was particularly interesting for the approach with the `HS_NAMESPACE` type.

Load-Generators: To produce appropriate workload, we placed four load-generators at different geographical locations. Three of them were hosted on an Amazon EC2 c3.xlarge instance, where the first load-generator (*LG-EU*) was hosted in Frankfurt, the second in US-east (*LG-US*) and the third in Singapore (*LG-AS*). The remaining load-generator (*LG-GWDG*) was hosted at GWDG.

3.4.2 Measurements

The overall evaluation procedure includes two different measurement procedures. The first measuring procedure was composed of two one-hour measurement runs. In both runs, the testing GPR was subjected to HTTP resolution requests from the three load-generators hosted on Amazon EC2 instances. In the first run, the Handle-PIDs were resolved by means of the `HS_RDS_URL` type approach (option (B)). In contrast to that, in the second run, they were resolved by means of the `HS_NAMESPACE` type approach (option (A)).
Also the second measuring procedure was composed of two one-hour measurement runs. The intention behind the second measuring procedure is to reveal the difference in the performance for resolving ordinary individual PIDs and the dynamic locator composition approach based on the `HS_NAMESPACE` type. In contrast to the first measuring procedure, in this procedure only the

GWDG load-generator was issuing HTTP resolution requests directly to the LHS without involving the testing GPR.

Finally, the measurements of the first measuring procedure are depicted by Figure 3.12 and Figure 3.13. Figure 3.12 illustrates the average resolution times. In addition to that, Figure 3.13 reveals the relative improvements for the resolution times. The relative improvement is measured by the formula $\left(\frac{R_S(B)-R_S(A)}{R_S(B)}\right)$, where $R_S(A)$ and $R_S(B)$ denote the resolution times recorded with option (A) and option (B) respectively.
Furthermore, Figure 3.14 shows the per request resolution times at the LHS for the ordinary and option (A)-approach (remember that option (B) does not involve the LHS).

3.4.3 Analysis

From Figure 3.12 we can see that the resolution time is obviously highly dependent on the network latency between the involved components. For the resolution by means of the option (A)-approach, the resolution time is mainly composed of the latency between three connections: first, between the load-generator and the testing GPR, secondly, between the testing GPR and the GHR and thirdly, between the testing GPR and the LHS. In Figure 3.11, the communication latencies are depicted as directed graphs.
Hence, the resolution time via option (A) includes the latencies ① - ③. In contrast to that, since with option (B) the locator composition is performed directly at the GPR, the resolution time with the `HS_RDS_URL` type approach is composed of only the latencies ① - ②. Therefore, for each load-generator, the resolution time with option (B) is always shorter than with option (A). In addition, with option (B), the resolution time is more stable than with option (A), which can be seen from the deviation whiskers plotted on the bars in Figure 3.12.
With increasing geographical distance between the load-generator and the GPR, the overall resolution time becomes expectingly longer. However, also the improvement effect with option (B) becomes smaller, which can be clearly seen from Figure 3.13. For the EC2 load-generator located in Frankfurt, the relative improvement is about 43%, whereas for the EC2 load-generator located in Singapore it is only about 10%.

Furthermore, Figure 3.14 shows that there is no significant difference between the resolution time with the `HS_NAMESPACE` type approach and the ordinary resolution approach. The average resolution time for resolving ordinary Handle-PIDs is $8.09ms$, where with the `HS_NAMESPACE` resolution approach it is $8.15ms$. Also the fluctuations in the resolution times of both approaches are pretty similar. This means that the algorithm for composing the locators at the LHS does not introduces a significant overhead onto the resolution time. It should be noted that in our setup, the MySQL databases attached to the Handle servers (primary and mirror) had about three million records.
Note that due to caching at the productive GPR, frequently resolved ordinary Handle-PID would have the same resolution times as the with option (B)-approach. The difference is that with option (B), all (not only the cached ones) DSID-prefixed Handle-PID would benefit from the shortened delegation procedure.

3.5 Summary

In this chapter, we have revealed that the fundamental performance problem with PIDs is based on a maintenance overhead involving a number of expensive transactional administration operations. Therefore, we have proposed a novel identification concept, which emphasizes on the performance improvement by reducing the number of these transactional administration operations. The core

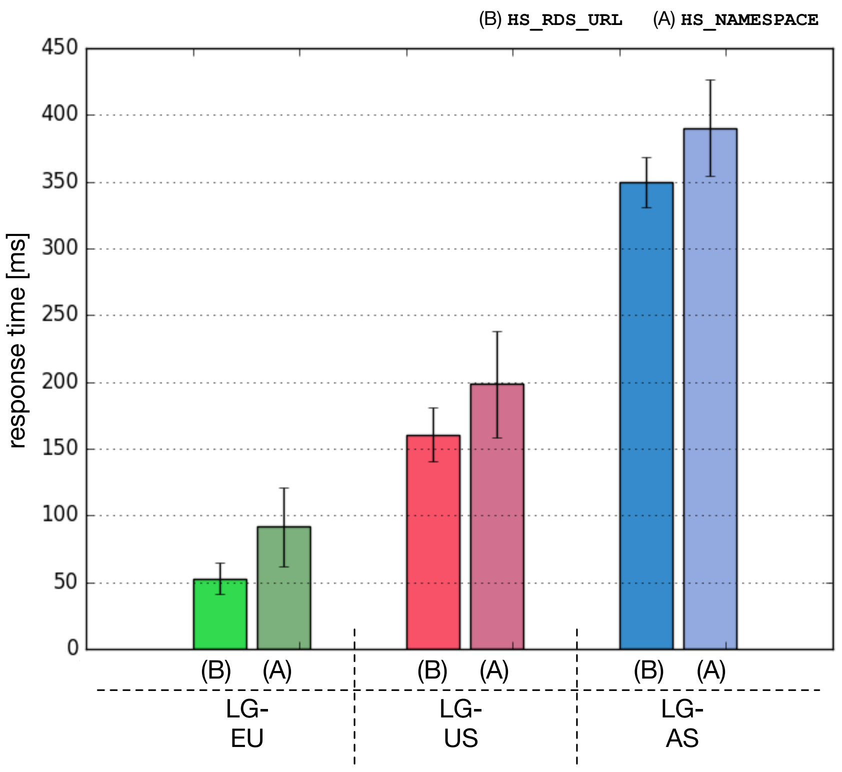


Figure 3.12: Comparison of average resolution times of the DSID approaches.

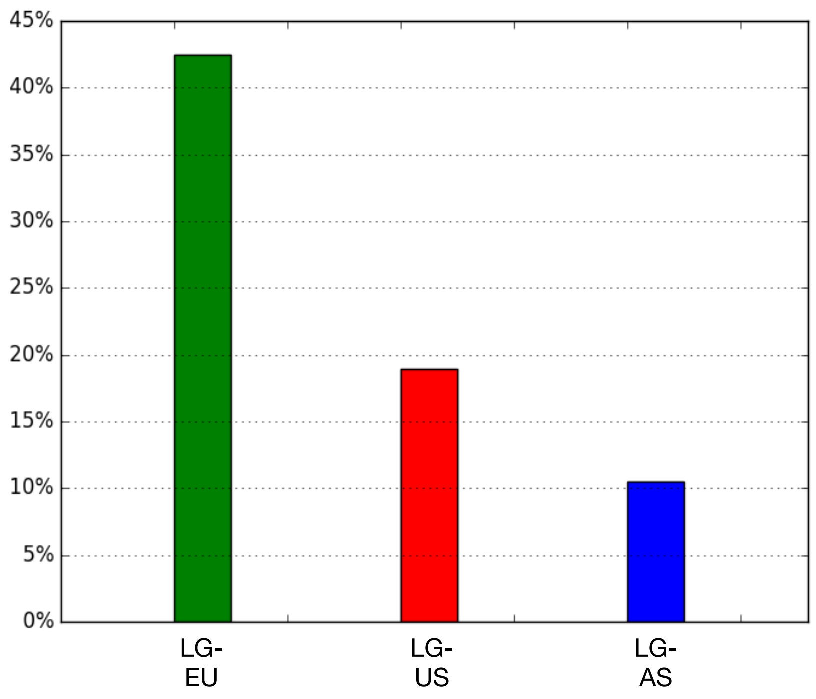


Figure 3.13: Comparison of the relative improvements $\left(\frac{R_S(B)-R_S(A)}{R_S(B)}\right)$ of the resolution times of the DSID approaches for different geographical positions.

idea is to enable internal identifiers to be globally resolvable without being individually registered at a PID system.

Suppose a group (C) research data repository, with two million datasets divided into two subsets: one million, only depending on the locator abstraction function of PIDs and the remaining million of datasets, for which also semantic information is imposed into the PID records. Since with our DSID approach we would have to only register the second million of datasets. Based on the response time of the ePIC *Persistent Identifier* (PID) system in Figure 3.3, this would result into the following calculation:

$$\begin{aligned} 1,000,000 \text{ PIDs} \times 250ms &\,\widehat{=}\, 250,000,000ms \\ &\,\widehat{=}\, 250,000s \\ &\,\widehat{=}\, 69h, \end{aligned}$$

instead of 138 hours. The improvement of the remaining 28 hours is addressed in the following two chapters Chapter 4 and Chapter 5.

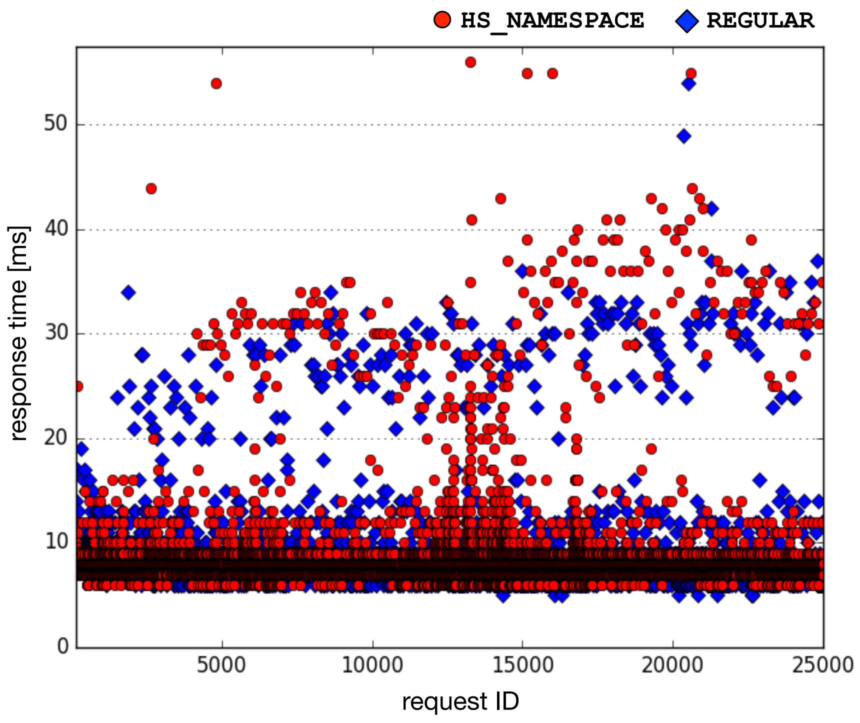


Figure 3.14: Comparison of the resolution times at the LHS of the regular resolution and the HS_NAMESPACE resolution approach. Hereby the mean resolution times are $8.09ms$ for the regular resolution and $8.15ms$ for the HS_NAMESPACE resolution approach.

Chapter 4

Persistent Identifier System Optimization

In the previous chapter we have seen that global PID systems are composed of myriads of naming authorities, which are used as distributed registries to maintain PID records. To ensure a durable and consistent PID record management, these naming authorities usually employ expensive transactional procedures. Therefore, in this chapter, we directly focus on the performance improvement of the naming authorities itself (cf. Figure 4.1), whereas the approach in the previous chapter was to reduce the amount of the PID record management operations. A naming authority can be

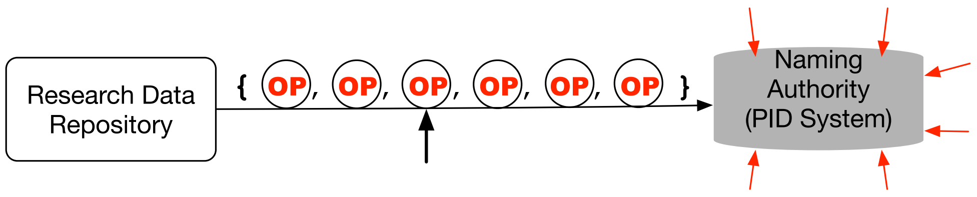


Figure 4.1: Focus in this chapter: Accelerating the administration operations (**OP**) by improving the response time of naming authorities.

considered as an ordinary transactional system, of which there are innumerable available in the current Internet. Usually, ordinary transactional Internet systems are composed as multi-tier systems, whereby each tier is tasked with a specific functionality.
The increasing volume of data leads to higher performance requirements at these transactional multi-tier Internet systems. Therefore, developers are compelled to steadily improve their systems, which is also valid for PID system developers.
Another aspect with PIDs is that their importance for research data management is increasing continuously. This is essentially based on the ability of PIDs of storing more complex information about the identified research dataset in addition to the sole locator. The increasing research dataset complexity in turn, leads also PID systems to become more complex, which finally results in an even higher overhead at the administration of PIDs.
We therefore emphasize on accelerating the PID administration procedures by improving the response time R_T of PID systems. Our discussion is initiated with the modeling of a multi-tier Internet system as a queuing network. This is then analyzed with the established *Mean Value Analysis* (MVA) algorithm. After determining the asymptotic behavior of the MVA algorithm, based on this, we deduce estimation formulas for the investigation of improvement effects in individual tiers onto the overall response time.

The base of our approach has been published in a conference paper at the 36th IEEE International Performance Computing and Communications (IPCCC) conference 2017 in San Diego, CA, USA

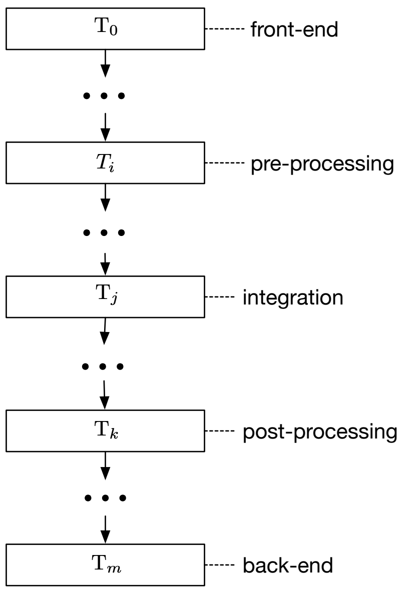


Figure 4.2: Typical Multi-Tier PID System.

[103]. In this thesis, however, we provide a revised and extended version.

4.1 PID Systems as Multi-Tier Internet Systems

As already indicated, transactional Internet systems, such as PID systems, are usually composed as multi-tier systems. This composition allows to partition the entire system into function blocks. Such a chaining of function blocks enables an efficient modification of the system for further development without significantly affecting the overall functioning. In the case of PID systems, with the advent of the *Digital Era*, the chaining of function blocks, usually tend to be extended. This is based on the fact that the massive research dataset growth not only requires a durable identification and retrieval concept, but also increasingly a concept, which provides a common structure and interpretability for very diverse research datasets. Therefore, the requirements on PID systems increase rapidly. These requirements are basically threefold:

(1) Persistent identification and access.

(2) Common data structure for diverse research datasets.

(3) High-performance registration and resolution.

As we have already discussed in the previous chapter, even the first requirement entails a particular overhead, which often leads to a significant impact on the performance. With the second requirement, the performance problem is even more aggravated, since it means further (data) processing in PID systems and hence, usually also a further overhead.
Figure 4.2 depicts a typical multi-tier PID system. It first of all consists of a front-end tier T_0, for receiving and responding to requests from applications or users, and a back-end tier T_m, which is often a transactional storage system for storing the PID records. The integration tier T_j constitutes the core component of the entire PID system. Its essential function is to integrate the system into the overlay network of the respective global PID system infrastructure, which is realized by im-

plementing the corresponding protocol. The integration tier of a PID system which for example is based on the Handle System, implements the Handle protocol in order to be able to communicate within the overlay network of the Handle System.
So far, with the described tiers, a PID system only meets the first and possibly also the third requirement. To comply with the second requirement, a PID system is also equipped with one or more data processing tiers T_i and/or T_k. The basic function of these data processing tiers is to transform the incoming data into a specific data structure, which is appropriate for more sophisticated analyses and use cases.

PID service providers often have to address two goals: On the one side, they are compelled to advance their PID systems with more functionality and on the other side, they have to optimize it to ensure a high-performant system.
Since PIDs are designed to provide long-term retrieval, the underlying PID systems have to be conceived for long-term operability. And since IT technology is developing incredibly fast, bringing forth new, faster hardware resources, operation systems, programing languages and more sophisticated and efficient frameworks, PID systems have to keep up with these technological advancements. The particular challenge with PID systems is the evolution in conjunction with the necessity to keep the PID-to-locator mappings intact.
The advancing technology also provides numerous possibilities for composing an appropriate and efficient function block chaining of multi-tier PID systems. The consequence is that PID service providers have to make strategic choices, which are usually based on particular parameters such as the system's response time, system's throughput, costs for development and deployment, etc. The ultimate aim is to achieve a best possible benefit-effort ratio for the optimization effort.

4.2 Response Time of Multi-Tier Internet Systems

The response time R_T is one of the most fundamental measures of a multi-tier Internet system. This is especially true for transactional multi-tier systems. From the perspective of a requesting application which issues requests against a multi-tier system, the response time is the time, the application has to wait until it gets a response from the multi-tier system. In fact, in a multi-tier Internet system, the response time R_T itself is usually a composition of various other measures, in particular, the contributions of the processing times of the individual tiers. Thus, an improvement of the overall response time R_T of a multi-tier system is usually achieved by improving the processing times of the individual tiers.
The overall goal of this chapter is therefore to provide an approach, which finally supports a strategic optimization procedure. The basic idea is to deduce:

- A mathematical analysis of the improvement impacts at individual tiers onto the overall response time R_T.

For this, we provide an analysis of the response time behavior in the following sections.

Before we proceed with our mathematical approach, we first introduce a definition of the terms used in this chapter.

4.2.1 Definition of Terms

Note that in this chapter, a superscript such as n in R_T^n denotes the number of requests which are concurrently processed in the overall system. In contrast to that, a superscript in brackets such as (k) in $a^{(k)}$ denotes a real exponent.

Since a multi-tier Internet system is usually subjected to multiple requests which are then concurrently processed, the response time R_T is highly dependent on the number of requests populating the system. To analyze the system's load dependent behavior, requires a specific modeling approach for which we provide a definition of the used terms:

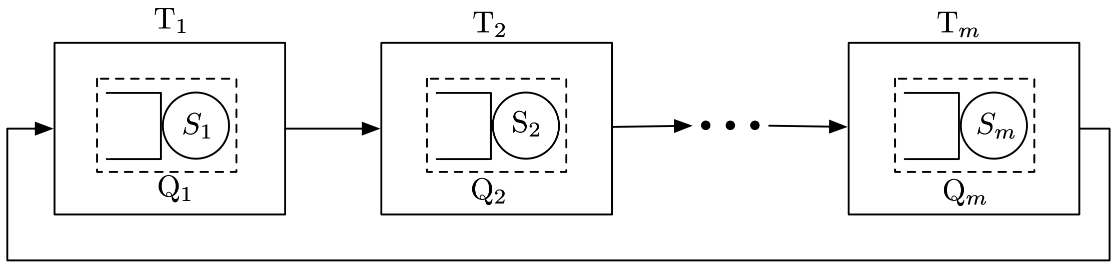


Figure 4.3: ($T_i \equiv Q_i$)-Multi-tier system with m tiers modeled as a closed-loop queuing network with m queues. The loop back from queue Q_m to Q_1 symbolizes a constant request population at a specific concurrency level n, which is a necessary condition for the analysis with the MVA algorithm.

Tier (T_i) As already described in Section 4.1, a tier is a function block, which is tasked with a specific function within a multi-tier system.

Queue (Q_i) In this chapter, a queue represents a M/M/1 queuing system, which is characterized by a Poisson arrival process ("M") and exponentially distributed service times ("M") (cf. Section 2.7). Another important characteristic is that the queue is assumed to have an unlimited capacity for queuing the incoming requests, which are then processed by a single service station ("1").

Number of Tiers (m) This number specifies the count of tiers composing the multi-tier system.

Request Population (n) This number specifies the count of requests, which are concurrently processed in the multi-tier system.

System Property ($T_i \equiv Q_i$) In this chapter, a multi-tier system, in which each tier can be modeled as a single queue is defined to be a *($T_i \equiv Q_i$)-multi-tier system*. Such a multi-tier system is schematized in Figure 4.3.

Service Time (S_i) The average time an individual tier T_i needs to process an incoming request, excluding the time being blocked while calling subsequent tiers, without being interfered by another request.

Queue Length (L_i^n) The average queue length of tier T_i when there are n requests concurrently processed in the overall system. Hence, it holds $L_i^n \leq n$ and $\sum_i^m L_i = n$.

Processing Time (R_i^n) The service time S_i including the queuing time when there are n requests concurrently processed in the overall system. In contrast to the service time, the processing time also includes the time, in which a request is waiting for the processing in an individual tier T_i.

Response Time (R_T^n) The average amount of time a multi-tier system needs for a complete processing of an individual request, while it is also busy with the concurrent processing of $(n-1)$ other requests. Note that the multi-tier system is then populated by n $(= (n-1)+1)$ requests.

With these definitions, we begin with our actual mathematical investigation approach.

4.2.2 Response Time Decomposition of a Multi-Tier System

When there is only a single request processed in the multi-tier system with m tiers $\{T_1, \cdots, T_m\}$, the response time can be written as the sum of the service times S_i of all involved tiers T_i:

$$R_T^1 = \sum_{i=1}^{m} S_i. \tag{4.2.1}$$

Such a situation is very common for testing systems, which are used by developers to test modifications in their systems. However, this situation does usually not cover the workload of the productive system, which is often subjected to multiple requests originating from multiple different requesting applications.

With an increasing concurrent request count, the individual service times get usually superposed with queuing times. To model the response time of a multi-tier Internet system, there are various different approaches. The state-of-the-art approach is to model the entire multi-tier system as a (complex) queuing network composed of multiple simple queuing systems and to apply the MVA algorithm to calculate the mean response time for a particular request population.
The basic approach behind the MVA algorithm is to consider the individual queues of the queuing network independently from each other. The overall network solution then results in the product of the individual solutions obtained from the individual queues. Such queuing networks are therefore also called *product-form queuing networks* (cf. Section 2.7.1).
The determination of the number of queues per tier T_i is one of the important aspects in the of modeling multi-tier systems. This, however, is beyond the scope of this thesis. Instead, the goal in this chapter is to provide insight into the effect of an improvement in an individual tier onto the overall system. This is important to understand how a planned or already implemented optimization will affect the overall systems' behavior. This is also important to determine whether a costly improvement effort is worthy to be tackled. Otherwise, this can often lead to a waste of valuable time and money. Therefore, to simplify the discussion onto the most relevant aspect, in this chapter we focus on multi-tier systems, which comply with the $(T_i \equiv Q_i)$ property, such as depicted in Figure 4.3.
Our approach is fundamentally based on the load distribution analysis of the established MVA algorithm. Therefore, in the following, we provide insight into several aspects of its asymptotic behavior.

4.2.2.1 The Basic MVA Algorithm

Algorithm 1 represents a basic form of the MVA algorithm, which was also used by Urgaonkar et al. [79]. Note that its background was already explained in Section 2.7.1. In this section, the given algorithm serves the purpose of clarity.
If we model each tier as a single queue, this algorithm can be used to predict a multi-tier system's response time for an increasing concurrency level n.

Algorithm 1 Basic MVA

```
1:  input: n, $S_i$, $1 \leq i \leq m$
2:  initialization:
3:  for i = 1 to m do
4:      $L_i = 0$
5:  end for
6:  for k = 1 to n do
7:      for i = 1 to m do
8:          $R_i = S_i(1 + L_i)$
9:      end for
10:     $\tau = \left(\frac{k}{\sum_{i=1}^{m} R_i}\right)$
11:     for i = 1 to m do
12:         $L_i = \tau R_i$
13:     end for
14: end for
15: $R_T^n = \sum_{i=1}^{m} R_i$
```

Since the MVA algorithm is a recursion, to determine the response time R_T^n for a concurrency level n, the algorithm always starts with the calculation of R_T^1 and terminates at the response time R_T^n for the concurrency level in question.
The lines 8, 10 and 12 of Algorithm 1 constitute the fundamental recurrence relation of the MVA algorithm, which can also be written as:

$$R_i^n = S_i(1 + \tau^{n-1} R_i^{n-1}), \tag{4.2.2}$$

where R_i^n denotes the processing time as defined above. The variable

$$\tau^{n-1} = \frac{(n-1)}{\sum_{i=1}^{m} R_i^{n-1}} \tag{4.2.3}$$

denotes the throughput of the multi-tier system for concurrency level n. The overall response time R_T^n is then the sum of the per tier processing times R_i^n.
The recurrence relation (4.2.2) only shows how the processing time at concurrency level n is composed from the processing time for $n-1$. To reveal all the terms incorporated from previous concurrency levels, the following theorem provides a compact representation of R_T^n. It also provides insight into the development of the response time R_T^n calculated by means of the MVA algorithm.

Theorem 1. *For the response time R_T^n, calculated by means of the MVA algorithm for a queuing network composed of m iterative queues, such as illustrated in Figure 4.3, it holds:*

$$R_T^1 = \sum_{i=1}^{m} S_i \quad \widehat{=} \quad (4.2.1) \tag{4.2.4}$$

and for $n > 1$ it holds:

$$R_T^n = R_T^1 + \sum_{i=1}^{m} \sum_{k=1}^{n-1} \left[\frac{(n-1)!}{(n-1-k)!}\right] \left[\frac{1}{\prod_{j=0}^{k-1} R_T^{n-1-j}}\right] S_i^{(k+1)}. \tag{4.2.5}$$

Proof. Since the overall response time is a sum of the processing times, it holds

$$R_T^n = \sum_{i=1}^{m} R_i^n.$$

Hereby, the relation (4.2.4) is trivially proven. The proof for (4.2.5) in turn, is then accomplished by deducing a compact representation for exactly these processing times:

$$\begin{aligned}
R_i^n &= \left(1+L_i^{n-1}\right)S_i, \quad \text{with} \quad L_i^{n-1} = \frac{(n-1)R_i^{n-1}}{\sum_{i=1}^m R_i^{n-1}} = \frac{(n-1)R_i^{n-1}}{R_T^{n-1}} \\
\Longrightarrow R_i^n &= \left(1+\frac{(n-1)R_i^{n-1}}{R_T^{n-1}}\right)S_i = \left(1+\frac{(n-1)\left[(1+L_i^{n-2})S_i\right]}{R_T^{n-1}}\right)S_i \\
&= \left(1+\frac{(n-1)\left[(1+\frac{(n-2)R_i^{n-2}}{R_T^{n-2}})S_i\right]}{R_T^{n-1}}\right)S_i = S_i + \frac{(n-1)}{R_T^{n-1}}S_i^{(2)} + \frac{(n-1)(n-2)R_i^{n-2}}{R_T^{n-1}R_T^{n-2}}S_i^{(2)} \\
&= S_i + \frac{(n-1)}{R_T^{n-1}}S_i^{(2)} + \frac{(n-1)(n-2)\left[(1+L_i^{n-3})S_i\right]}{R_T^{n-1}R_T^{n-2}}S_i^{(2)} \\
&= S_i + \frac{(n-1)}{R_T^{n-1}}S_i^{(2)} + \frac{(n-1)(n-2)\left[(1+\frac{(n-3)R_i^{n-3}}{R_T^{n-3}})S_i^{(2)}\right]}{R_T^{n-1}R_T^{n-2}}S_i^{(2)} \\
&= S_i + \frac{(n-1)}{R_T^{n-1}}S_i^{(2)} + \frac{(n-1)(n-2)}{R_T^{n-1}R_T^{n-2}}S_i^{(3)} + \frac{(n-1)(n-2)(n-3)R_i^{n-3}}{R_T^{n-1}R_T^{n-2}R_T^{n-3}}S_i^{(3)}
\end{aligned}$$

By continuing with the successive substitution of the emerging terms, for the processing times $R_i^{n-3}, R_i^{n-4}, \cdots R_i^1$ we finally get:

$$\begin{aligned}
R_i^n &= S_i + \frac{(n-1)}{R_T^{n-1}}S_i^{(2)} + \frac{(n-1)(n-2)}{R_T^{n-1}R_T^{n-2}}S_i^{(3)} + \\
&\quad \frac{(n-1)(n-2)(n-3)}{R_T^{n-1}R_T^{n-2}R_T^{n-3}}S_i^{(4)} + \cdots + \frac{(n-1)!}{R_T^{n-1}\cdots R_T^1}S_i^{(n)}
\end{aligned} \tag{4.2.6}$$

The final step of the proof is to build the sum of all R_i^n, $i = 1, \cdots, m$, $\sum_{i=1}^m R_i^n$, which finally yields (4.2.5). □

Theorem 1 reveals that the response time R_T^n calculated by means of the MVA, is basically composed of power series of the services times S_i with complex coefficients depending on the response times from previous steps, which therefore again results in a recursion.
However, this theorem is not appropriate for practical situations. First of all, it is numerically highly unstable due to floating point errors. More importantly, it is not suitable to reveal specific asymptotic behaviors of the response time. Yet, to understand the overall system behavior for increasing concurrency this asymptotic behavior is essential.

Therefore, we proceed with a reformulation of the recurrence relation in equation (4.2.2):

$$R_i^n = S_i(1+\tau^{n-1}R_i^{n-1}) = S_i\left(1+\frac{(n-1)R_i^{n-1}}{\sum_{j=1}^m R_j^{n-1}}\right). \tag{4.2.7}$$

$$\Longrightarrow \boxed{R_i^n = S_i\left(1+(n-1)\omega_i^{n-1}\right)}, \tag{4.2.8}$$

with

$$\omega_i^{n-1} := \frac{R_i^{n-1}}{\sum_{j=1}^{m} R_j^{n-1}}, \quad 0 < \omega_i^{n-1} < 1 \tag{4.2.9}$$

and

$$\sum_{i=1}^{m} \omega_i^{n-1} = 1. \tag{4.2.10}$$

The term ω_i^n, also called weighting factor, is a relative measure of how many of the n requests in the entire system are processed at tier T_i. The relation (4.2.9) is true, because an individual weighting factor is a ratio between the corresponding processing time and the overall response time (composed of all processing times).
Note that $L_i^n \leq n-1$, since L_i^n denotes the queue length of T_i at the moment when another request is going to populate the system while there are already $(n-1)$ requests in processing. With the recurrence relation (4.2.8), the term for the queue length L_i^{n-1} is expressed as a per tier weighting factor, distributing the overall load among the individual tiers.
The advantage with these weighting factors is that they enable to analyze the asymptotic behavior of the response time calculated with the MVA algorithm. The analysis is based on the characteristic of the weighting factors that they are always bounded by $0 < \omega_i < 1$. Our analysis of the MVA algorithm behavior is essentially based on the development of these weighting factors. This is dedicatedly examined in the following section.

4.2.3 MVA Algorithm Response Time Behavior

If we assume that an individual request has to traverse all m tiers of the multi-tier system (cf. Figure 4.3), the overall response time R_T^n for a successful request processing, consists of all the service times S_i including the queuing times: $S_i L_i^n = (n-1) S_i \omega_i^{n-1}$.
With an increasing request number n, the queuing times grow monotonically, where the growth rate is different from to tier to tier. The growth of an individual queuing time is determined by the development of the corresponding weighting factor ω_i.
The analysis in this chapter is fundamentally based on the following theorem. It provides insight into the behavior of the overall response time R_T^n calculated by the MVA algorithm. More importantly, it also reveals insight into the growth of the weighting factors:

Theorem 2. *Assume S_{max} to be the dominating service time with $S_{max} = max\{S_1, \cdots, S_m\}$ among the m tiers of a ($T_i \equiv Q_i$)-multi-tier system. Further, assume $\mathcal{D}_S$ to be a subset of the tier indices, for which it holds $\mathcal{D}_S = \{i \in \{1, \cdots, m\} \mid S_i = S_{max}\}$ with a cardinality $k := \texttt{card}(\mathcal{D}_S)$ (cardinality $\widehat{=}$ number of elements of a set).*
Then, for the weighting factors ω_i^n, for an increasing concurrency level ($n \to \infty$) it holds:

$$\lim_{n\to\infty} \omega_i^n = \frac{1}{k} \quad \forall i \in \mathcal{D}_S \tag{4.2.11}$$

and

$$\lim_{n\to\infty} \omega_i^n = 0 \quad \forall i \notin \mathcal{D}_S \tag{4.2.12}$$

Proof. First of all, for the processing time of all $l \in \mathcal{D}_S$ it holds:

$$R_{max}^n = S_{max}\left(1 + (n-1)\omega_{max}^{n-1}\right) = S_l\left(1 + (n-1)\omega_l^{n-1}\right) = R_l^n$$

with

$$\omega_{max}^{n-1} = \frac{R_{max}^{n-1}}{R_T^{n-1}} = \omega_l^{n-1}.$$

Secondly, for the remaining tiers with indices $p \notin \mathcal{D}_S$ it holds:

$$R_p^n = S_p \left(1 + (n-1)\omega_p^{n-1}\right)$$

with

$$\omega_p^{n-1} = \frac{R_p^{n-1}}{R_T^{n-1}}.$$

Furthermore, the monotonic increase of the ω_{max}^n is proven by the following explanation:
For $(n = 1)$ we have:

$$R_{max}^1 = S_{max} \quad \text{with} \quad \omega_{max}^1 = \frac{S_{max}}{R_T^1}$$

and

$$R_p^1 = S_p \quad \text{with} \quad \omega_p^1 = \frac{S_p}{R_T^1} \quad \forall p \notin \mathcal{D}_S$$

Hence, it holds $\omega_{max}^1 > \omega_p^1$, which means that the tiers $l \in \mathcal{D}_S$ have the largest contribution onto the overall response time R_T^1.
For $(n = 2)$ we have:

$$R_{\text{max}}^2 = S_{max}(1 + \omega_{max}^1) \quad \text{with} \quad \omega_{max}^2 = \frac{R_{max}^2}{R_T^2}$$

and

$$R_p^2 = S_j(1 + \omega_p^1) \quad \text{with} \quad \omega_p^2 = \frac{R_p^2}{R_T^2} \quad \forall p \notin \mathcal{D}_S$$

Obviously, it holds $R_{max}^2 > R_{max}^1$ and $R_p^2 > R_p^1$. However, the growth rate between R_{max}^1 and R_{max}^2 is higher than between R_p^1 and R_p^2. This is based on the the dominating weighting factors (ω_{max}^1) from the previous step. Therefore, the contribution of the tiers $l \in \mathcal{D}_S$ onto the overall response time R_T^2 is even more increased. Thus, it holds $\omega_{max}^2 > \omega_{max}^1$. On the other side, with the growing weighting factors of the dominating tiers, from relation (4.2.10) it immediately follows $\omega_p^2 < \omega_p^1$.
In the next step $(n = 3)$, the ω_{max}^2 will lead the growth rate between R_{max}^2 and R_{max}^3 to be again higher than between R_p^2 and R_p^3. This in turn leads again to $\omega_{max}^3 > \omega_{max}^2$ and $\omega_p^3 < \omega_p^2$.

Therefore, for an arbitrary n, the growth rate between R_{max}^{n-1} and R_{max}^n will be always higher than between R_p^{n-1} and R_p^n, which finally leads to $\omega_{max}^n > \omega_{max}^{n-1}$ and $\omega_p^n < \omega_p^{n-1}$.
Ultimately, the monotonic decrease of the ω_p^n leads to $\omega_p^n \to 0$ for $n \to \infty$. Where the monotonic increase of the ω_{max}^n leads with relation (4.2.10) for $n \to \infty$ to $\omega_{max}^n \to 1/(\mathtt{card}(\mathcal{D}_S)) = \frac{1}{k}$. $\square$

Theorem 2 basically reveals insight into the load distribution behavior of the MVA algorithm. Hence, according to Theorem 2, the core characteristic of the MVA algorithm is that the overall response time will be increasingly dominated by the processing times of the dominating tiers.
The key for this analysis has been provided by our introduction of the weighting factors in equation (4.2.8).

This monotonic behavior allows to deduce another theorem, which reveals boundaries for the overall response time:

Theorem 3. *For $n > 1$, the overall response time R^n_T of a ($T_i \equiv Q_i$)-multi-tier system composed of m tiers and calculated with the MVA algorithm, is bounded by:*

$$R^n_\bigtriangledown \leq R^n_T \leq R^n_\triangle,$$

with

$$R^n_\bigtriangledown := \sum_{i=1}^{m} S_i \left(1 + (n-1)\omega_i^1\right), \quad \omega_i^1 = \frac{S_i}{\sum_{j=1}^{m} S_j}$$

and

$$R^n_\triangle := \sum_{i=1}^{m} S_i + (n-1)S_{max}$$

Proof. First of all, to obtain an upper boundary for the overall response time R^n_T, it is necessary to weight the dominating processing times R^n_l, $l \in \mathfrak{D}_S$ (T_l is a dominating tier) with the maximal possible weighting factors. In addition, for the remaining processing times R^n_p, $p \notin \mathfrak{D}_S$ (T_p is not a dominating tier) it is required to weight them with the minimal possible weighting factors.
Since the contributions ω_l of the dominating tiers T_l, $l \in \mathfrak{D}_S$ are monotonically increasing, we have $l \in \mathfrak{D}_S : \lim_{n\to\infty} \omega_l^n = \frac{1}{k}$ and, therefore, also $p \notin \mathfrak{D}_S : \lim_{n\to\infty} \omega_p^n \to 0$.
Therefore, an upper boundary $R^n_\triangle$ is achieved by weighting the processing times R^n_l, $l \in \mathfrak{D}_S$ with $1/k$, and, by weighting the processing times R^n_p, $p \notin \mathfrak{D}_S$ with 0:

$$\begin{aligned}
R^n_T &= \sum_{p\notin\mathfrak{D}_S} S_p \left(1 + (n-1)\omega_p^n\right) + \sum_{l\in\mathfrak{D}_S} S_{max}\left(1 + (n-1)\omega_l^n\right) \\
&\leq \sum_{p\notin\mathfrak{D}_S} S_p \left(1 + (n-1)0\right) + \sum_{l\in\mathfrak{D}_S} S_{max}\left(1 + (n-1)\frac{1}{k}\right) \\
&= \sum_{p\notin\mathfrak{D}_S} S_p + \sum_{l\in\mathfrak{D}_S} S_{max}\left(1 + (n-1)\frac{1}{k}\right) \\
&= \sum_{p\notin\mathfrak{D}_S} S_p + kS_{max}\left(1 + (n-1)\frac{1}{k}\right) \\
&= \sum_{p\notin\mathfrak{D}_S} S_p + kS_{max}(n-1) = \sum_{i=1}^{m} S_i + (n-1)S_{max} = R^n_\triangle
\end{aligned}$$

A lower boundary is then obtained by using the initial weighting factors ω_i^1 for the processing times R^n_i with $n \geq 1$. By this, the contributions of the dominating processing times R^n_l, $l \in \mathfrak{D}_S$ get damped, where the contributions of the remaining tiers R^n_p, $p \in \mathfrak{D}_S$ get amplified:

$$\begin{aligned}
R^n_T &= \sum_{p\notin\mathfrak{D}_S} S_p \left(1 + (n-1)\omega_p^n\right) + \sum_{l\in\mathfrak{D}_S} S_{max}\left(1 + (n-1)\omega_l^n\right) \\
&\geq \sum_{p\notin\mathfrak{D}_S} S_p \left(1 + (n-1)\omega_p^1\right) + \sum_{l\in\mathfrak{D}_S} S_{max}\left(1 + (n-1)\omega_l^1\right) \\
&\geq \sum_{i=1}^{m} S_i \left(1 + (n-1)\omega_i^1\right) = R^n_\bigtriangledown
\end{aligned}$$

This is true, because the response time development with the MVA algorithm is based on the fact that, the dominating tiers with the service time S_{max} are in each step weighted with the largest weighting factors (ω^n_{max}). Since the initial weighting factors (ω^1_{max}) for the dominating tiers are always less than or equal to the weighting factor ω^n_{max}, the sum of the resulting processing times will be always smaller than the actual R^n_T. □

Until this point, we have analyzed the response time behavior of the MVA algorithm and deduced boundaries for the actual response time. This analysis is essentially based on the load distribution behavior of the MVA algorithm represented by the weighting factors. In the following, these results are used to investigate the effect of optimizations achieved in individual tiers. Moreover, in contrast to the analysis so far, the following discussion basically considers two systems: a multi-tier system and its improved version.

4.3 Speedup Effects in Multi-Tier Internet Systems

The aim of this section is to provide an understanding for the resulting effects of an improvement endeavor in a tier T_i of a multi-tier system. Therefore, after defining our terms to measure the results of improvements in individual tiers, we proceed with the actual analysis of the optimization impacts.

4.3.1 Speedup Measures

The notion of speedup is mainly established in the context of parallelization. In this chapter, we use the speedup to quantify the improvement of an individual tier's service time or processing time in a multi-tier system. For this reason, we first define the service time speedup α_i as:

$$\alpha_i = \frac{S_i}{\widehat{S}_i}, \tag{4.3.1}$$

where S_i and $\widehat{S}_i$ denote the service times of the old and improved tier T_i respectively.
Since an improvement in the service times also affects the processing times R_i^n,we define the speedup for the concurrency level n as:

$$\alpha_i^n = \frac{R_i^n}{\widehat{R}_i^n}, \tag{4.3.2}$$

where $n = 1$ leads to (4.3.1).
Likewise, we define the overall response time speedup κ^n for concurrency level n of the multi-tier system as:

$$\kappa^n = \frac{R_T^n}{\widehat{R}_T^n}, \tag{4.3.3}$$

where R_T^n and $\widehat{R}_T^n$ denote the response times of the old and improved system respectively. The response time $\widehat{R}_T^n$ of the improved system is the result of any improvement endeavor in at least one of the tiers of the old system.
For the rest of this thesis, we define $\kappa := \kappa^1$.

4.3.2 Response Time Speedup

The following lemma provides insight into a multi-tier system's response time speedup κ^n achieved by an improvement endeavor in individual tiers.

Lemma 1. *Assume a processing time improvement in at least a single tier T_i with a speedup factor $\alpha_i^n \geq 1$ for the concurrency level n of a ($T_i \equiv Q_i$)-multi-tier system composed of m tiers. Then for the response time speedup κ^n it holds:*

$$\kappa^n = \left(\sum_{i=1}^{m} \frac{\omega_i^n}{\alpha_i^n} \right)^{-1},$$

with the corresponding weighting factors

$$\omega_i^n = \frac{R_i^n}{\sum_{i=1}^m R_i^n}. \tag{4.3.4}$$

Proof. With definition (4.3.3) and from the response time decomposition (4.2.1), for the reciprocal of the response time speedup it follows:

$$\frac{1}{\kappa^n} = \frac{\widehat{R}_T^n}{R_T^n} = \frac{\sum_{i=1}^m \widehat{R}_i^n}{\sum_{i=1}^m R_i^n},$$

and with (4.3.2)

$$\frac{1}{\kappa^n} = \frac{\sum_{i=1}^m R_i^n/\alpha_i^n}{\sum_{i=1}^m R_i^n} = \sum_{i=1}^m \frac{1}{\alpha_i^n} \underbrace{\left[\frac{R_i^n}{\sum_{i=1}^m R_i^n}\right]}_{=:\omega_i^n} = \sum_{i=1}^m \frac{\omega_i^n}{\alpha_i^n}.$$

$$\Longrightarrow \kappa^n = \left(\sum_{i=1}^m \frac{\omega_i^n}{\alpha_i^n}\right)^{-1}.$$

□

Lemma 1 implies that the impact of an individual α_i^n onto the overall κ^n is (naturally) dependent on the corresponding weighting factor ω_i^n. Thus, the larger the weighting factor ω_i^n, the higher its contribution to the overall response time speedup κ^n.

In contrast to this lemma, the following subsection analyzes the relation between the weighting factors of the improved and unimproved multi-tier system, which is important to understand how an improvement effect will affect the improved system's behavior.

4.3.3 Effect of Load Redistribution

Since an improvement of an individual tier basically leads to a redistribution of the weighting factors, the following theorem gives insight into this *effect of load redistribution*.

Theorem 4. *Suppose an optimization effort in tier T_i of a ($T_i \equiv Q_i$)-multi-tier system to result in a speedup factor κ^n for the overall response time. Then, for the weighting factors $\widehat{\omega}_i^n$ of the improved system it holds:*

$$\widehat{\omega}_i^n = \omega_i^n \frac{\kappa^n}{\alpha_i^n} \tag{4.3.5}$$

Proof.

$$\frac{\widehat{\omega}_i^n}{\omega_i^n} = \frac{\widehat{R}_i^n}{\sum_{j=1}^m \widehat{R}_j^n} \frac{\sum_{j=1}^m R_j^n}{R_i^n} = \underbrace{\left(\frac{\sum_{j=1}^m R_j^n}{\sum_{j=1}^m \widehat{R}_j^n}\right)}_{\kappa^n} \underbrace{\left(\frac{\widehat{R}_i^n}{R_i^n}\right)}_{1/\alpha_i^n}$$

$$\Longleftrightarrow \widehat{\omega}_i^n = \omega_i^n \frac{\kappa^n}{\alpha_i^n}$$

□

Hence, the resulting weighting factor $\widehat{\omega}_i^n$ of the improved system is determined by the factor (κ^n/α_i^n), which we define to be the *redistribution factor*.
For the particular case of concurrency level $n = 1$, Theorem 4 results in the following corollary:

Corollary 1. *Suppose for the concurrency level $n = 1$ there is a service time improvement only in tier T_i of a ($T_i \equiv Q_i$)-multi-tier system with $\alpha_i = \left(S_i/\widehat{S}_i\right)$, resulting in a response time speedup κ, then for the weighting factors ω^1 it holds:*

$$\widehat{\omega}_{j\neq i}^1 = \omega_{j\neq i}^1 \kappa \quad \textit{and} \quad \widehat{\omega}_i^1 = \omega_i^1 \frac{\kappa^n}{\alpha_i^1}.$$

Proof. Since we have $\widehat{S}_j = S_j$ for $j \neq i$, it follows:

$$\frac{\widehat{\omega}_j^1}{\omega_j^1} = \left(\frac{\widehat{S}_j}{\sum_{j=1}^m \widehat{S}_j}\right)\left(\frac{\sum_{j=1}^m S_j}{S_j}\right) = \frac{\sum_{j=1}^m S_j}{\sum_{j=1}^m \widehat{S}_j} = \frac{R_T^1}{\widehat{R}_T^1} = \kappa$$

For the ratios of the weighting factors of the improved tier T_i we have:

$$\frac{\widehat{\omega}_i^1}{\omega_i^1} = \left(\frac{\widehat{S}_i}{\sum_{i=1}^m \widehat{S}_i}\right)\left(\frac{\sum_{i=1}^m S_i}{S_i}\right) = \frac{\sum_{i=1}^m S_i}{\sum_{i=1}^m \widehat{S}_i}\left(\frac{\widehat{S}_i}{S_i}\right) = \frac{R_T^1}{\widehat{R}_T^1}\left(\frac{1}{\alpha_i}\right) = \frac{\kappa}{\alpha_i}.$$

□

The difference between Theorem 4 and Corollary 1 is twofold: First, Corollary 1 is only valid for the concurrency level $n = 1$. In addition, it considers an improvement effort in only one tier T_i. Hence, $\alpha_{j\neq i}^1$ and $\alpha_i^1 \geq 1$. Therefore, the redistribution factor between $\widehat{\omega}_{j\neq i}^1$ and $\omega_{j\neq i}^1$ results in $\kappa^1 = \kappa$, where between $\widehat{\omega}_i^1$ and ω_i^1 (the improved tier) it results in κ/α_i. In contrast to that, Theorem 4 provides a general mathematical description of the load redistribution effect, caused by improving the service time of at least a single tier.

4.3.4 Speedup Effect Estimation

Remember that our goal was to mathematically analyze the improvement impact at an individual tier onto the overall system's response time R_T^n. Since the overall R_T^n is determined by the processing times of the individual tiers, we have to focus on the resulting effects onto the processing times. The processing times in turn are composed by the services times superposed with the queuing times. The queuing times in turn are specified by the individual population counts, which are determined by the weighting factors ω_i^n. Finally, the result of an improvement endeavor can be captured by the modifications in these weighting factors, which are mathematically described by Theorem 4. Since this mathematical description depends on the ratio (κ^n/α_i^n), requiring the two terms κ^n and α_i^n, the following discussion is mainly intended to deduce estimation formulas for these two terms.
These estimation formulas are necessary to enable predications for the resulting effects for an intended improvement endeavor.

Commonly, improvements on systems are first implemented into testing systems. Often, only when the improvements successfully endure a particular evaluation procedure, these improvements become a release for productive deployment. However, a comprehensive evaluation procedure itself, usually entails a significant sophistication. This is especially true for the workload

with which the testing system gets stressed.
To quantify the benefit of an improvement effort, an individual developer usually first measures the response time for iteratively submitted requests. In this case, the system is only populated by one request. By taking the ratio of the response times R_T^1 and $\widehat{R}_T^1$ then reveals the achieved response time speedup κ. The effort for this evaluation approach is often relatively small: Current multi-tier Internet systems are commonly equipped with sophisticated logging modules, which provide extensive information about the system's metrics, with the response time often being a default measure. Thus, the response time can usually be easily extracted from the logging files of the system.
However, a productive multi-tier Internet system is generally subjected to multiple requests which are concurrently processed in the system. Hence, the productive workload is typically characterized by a concurrency level $n > 1$. The essential question here is which quantity the response time speedup κ^n will have for the productive concurrency level $n > 1$, instead of only κ for $n = 1$.
To capture the behavior of the testing system for a productive workload further effort is required, which is especially true for multi-tier Internet systems, which only provide special interfaces, such as the ePIC PID system. To evaluate such systems, either the deployment of a special proprietary stress testing framework or an in-house development is necessary. Both are again associated with additional costs.
In the next sections, we provide another evaluation methodology, which is based on estimations of the formulas deduced in the previous sections. The underlying assumption is that the measures of the unimproved system are already at hand or easily extractable.

4.3.4.1 Response Time Speedup Boundaries

By means of the measures gathered from the testing system for iteratively submitted requests, the following lemma can be used to determine the overall boundaries for the response time speedup.

Lemma 2. *Given a ($T_i \equiv Q_i$)-multi-tier system with a dominating service time $S_{max} = max\{S_1, \cdots, S_m\}$. Further, assume a service time improvement α_i in tier T_i to result in a response time speedup κ for concurrency level $n = 1$. Moreover, assume $\widehat{S}_{max} = max\{\widehat{S}_1, \cdots, \widehat{S}_m\}$ to be the dominating service time of the improved (future) multi-tier system.*
Then, for an upper and a lower boundary for the response time speedup, it holds:

$$\kappa_{\bigtriangledown} := min\left(\frac{S_{max}}{\widehat{S}_{max}}, \kappa\right) \quad (4.3.6)$$

$$\kappa_{\triangle} := \frac{R_{\triangle}^n}{\widehat{R}_{\triangle}^n} \quad (4.3.7)$$

$$\Longrightarrow \kappa_{\bigtriangledown} \le \kappa^n \le \kappa_{\triangle} \quad (4.3.8)$$

Proof. From Theorem 3 we have:

$$\lim_{n\to\infty} R_T^n = \sum_{i=1}^{m} S_i + (n-1)S_{max}$$

and

$$\lim_{n\to\infty} \widehat{R}_T^n = \sum_{i=1}^{m} \widehat{S}_i + (n-1)\widehat{S}_{max}$$

for the response times of the old and improved system respectively.
This results in the following:

$$\kappa^{\infty} := \lim_{n\to\infty} \kappa^n = \frac{R_T^n}{\widehat{R}_T^n} = \frac{\sum_{i=1}^m S_i + (n-1)S_{max}}{\sum_{i=1}^m \widehat{S}_i + (n-1)\widehat{S}_{max}}.$$

Furthermore, we define:

$$r(n) := \sum_{i=1}^{m} S_i + (n-1)S_{max}$$

and

$$\widehat{r}(n) := \sum_{i=1}^{m} \widehat{S}_i + (n-1)\widehat{S}_{max},$$

which results:

$$\Longrightarrow \kappa^{\infty} := \lim_{n\to\infty} \kappa^n = \frac{r(n)}{\widehat{r}(n)}. \tag{4.3.9}$$

Applying L'Hospital's rule on equation (4.3.9) finally yields:

$$\kappa^{\infty} = \lim_{n\to\infty} \kappa^n = \frac{\frac{d}{dn} r(n)}{\frac{d}{dn}\widehat{r}(n)} = \left(\frac{S_{max}}{\widehat{S}_{max}}\right).$$

Hence, the response time speedup factor starts at κ and moves towards $\kappa^{\infty} = S_{max}/\widehat{S}_{max}$. However, during the course towards κ^{∞}, it is possible that κ^n can have values larger than κ and $S_{max}/\widehat{S}_{max}$. Therefore, with $min\left(\frac{S_{max}}{\widehat{S}_{max}}, \kappa\right)$ we also have a lower boundary for the response time speedup.
An upper boundary is determined by taking the ratio between an upper boundary for the response time of the old system and an lower boundary for the response time of the improved (future) system. Thus, $\kappa^n \leq \left(R_{\triangle}^n / \widehat{R}_{\triangledown}^n\right)$.
In practical situations, where R_T^n is at hand, it can be used to derive a more accurate estimation:

$$\begin{aligned} \kappa_{\triangle}^n &= \frac{R_T^n}{\widehat{R}_{\triangledown}^n} \\ &= \frac{R_T^n}{\widehat{R}_T^1 + (n-1)\sum_{i=1}^m \widehat{S}_i \widehat{\omega}_i^1} \end{aligned} \tag{4.3.10}$$

□

Lemma 2 provides an important tool for the whole improvement endeavor, as it only relies on the services times and not on the processing times. If for instance, an improvement effort leads to $\widehat{S}_{max} = S_{max}$, for increasing n, the response time speedup κ^n moves towards $\kappa^{\infty} = 1$, which means that the benefit of the effort becomes diminished with increasing concurrency level.
The situation where it holds $\widehat{S}_{max} = S_{max}$, is achieved by either improving any non-dominating tier T_p with $S_p \neq S_{max}$ or by not improving the complete set of the dominating tiers T_l with $\mathcal{D}_S = \{l \mid S_l = S_{max} \wedge l \in 1, \cdots m\}$. For multi-tier systems subjected to high concurrency levels, this means that for a significant speedup of the response time, the improvement endeavor has to address all the dominating tiers.

4.3.4.2 Weighting Factor Boundaries

As already discussed, an optimization in an individual tier will lead to a redistribution of the load. The theoretical insight has already been summarized with Theorem 4. As indicated in the first paragraph of Section 4.3.4, in the following, we provide estimations for the resulting load at individual tiers. In principle, all so far deduced formulas are incorporated into these estimations, in particular the load distribution behavior specified by Theorem 2.

This subsection includes two corollaries: The first corollary is applicable for multi-tier systems, which are composed of multiple dominating tiers. The second corollary in turn, is appropriate when the multi-tier system only includes a single dominating tier.

Corollary 2. *Suppose an optimization effort at one of the dominating tiers T_d with $d \in \mathcal{D}_S$ ($\equiv S_d = S_{max}$) and $k = \mathtt{card}(\mathcal{D}_S)$ of a ($T_i \equiv Q_i$)-multi-tier system to result in a response time speedup κ for concurrency level $n = 1$. Further, suppose the service time speedup α_d to be of such a degree that for $\widehat{T_d}$ it holds $d \notin \widehat{\mathcal{D}}_S$ (tier T_d is not part of dominating tiers in the improved system) and $(k-1) = \mathtt{card}(\widehat{\mathcal{D}}_S)$.*
Then, for the new weighting factors $\widehat{\omega}_i^n$ of the improved system, we have the following three cases:

Case 1: *T_l with $l \in \widehat{\mathcal{D}}_S$ (remaining dominating tiers) $\Longrightarrow$*

$$\omega_l^n \kappa_\triangledown \le \widehat{\omega}_l^n \le \omega_l^n \kappa_\vartriangle \left(\frac{1 + \frac{(n-1)}{(k-1)}}{1 + (n-1)\omega_l^{n-1}} \right)$$

Case 2: *T_p with $p \notin \mathcal{D}_S$ and $p \notin \widehat{\mathcal{D}}_S$ (remaining tiers) $\Longrightarrow$*

$$\omega_p^n \kappa_\triangledown \le \widehat{\omega}_p^n \le \omega_p^n \kappa_\vartriangle \left(\frac{1 + (n-1)\kappa\omega_p^1}{1 + (n-1)\omega_p^{n-1}} \right)$$

Case 3: *T_d with $d \in \mathcal{D}_S$ and $d \notin \widehat{\mathcal{D}}_S$ (improved tier) $\Longrightarrow$*

$$\omega_d^n \left(\frac{\kappa_\triangledown}{\alpha_d^1 (1 + \frac{n-1}{k})} \right) \le \widehat{\omega}_d^n \le \omega_d^1 \frac{\kappa}{\alpha_d^1}$$

Proof. The proof is based on equation (4.3.5), which basically reveals how the weighting factors are changed after a tier improvement. To accomplish the proof, for each case we have to find appropriate upper and lower boundaries for κ^n and $1/\alpha_i^n$. Since a lower and an upper boundary for κ^n is already given by Lemma 2, the proof basically consists of deducing boundaries for $1/\alpha_i^n$:

Case 1:

In this case, the weighting factors $\widehat{\omega}_l^n$ of the remaining dominating tiers are increasing towards

$1/(k-1)$ instead to $1/k$. By this, for an upper boundary we have:

$$\begin{aligned}
\widehat{\omega}_l^n &= \omega_l^n \frac{\kappa^n}{\alpha_l^n} \\
&\leq \omega_l^n \kappa_\triangle \frac{1}{\alpha_l^n} \leq \omega_l^n \kappa_\triangle \frac{\widehat{R}_l^n}{R_l^n} \\
&\leq \omega_l^n \kappa_\triangle \left(\frac{\widehat{S}_l^n (1 + (n-1)\widehat{\omega}_l^{n-1})}{S_l^n (1 + (n-1)\omega_l^{n-1})} \right) \\
&\leq \omega_l^n \kappa_\triangle \left(\frac{1 + \frac{(n-1)}{(k-1)}}{1 + (n-1)\omega_l^{n-1}} \right)
\end{aligned}$$

The lower boundary is deduced by the following:

$$\begin{aligned}
\widehat{\omega}_l^n &= \omega_l^n \frac{\kappa^n}{\alpha_l^n} \\
\widehat{\omega}_l^n &\geq \omega_l^n \kappa_\triangledown \frac{1}{\alpha_l^n} \geq \omega_l^n \kappa_\triangledown \frac{\widehat{R}_l^n}{R_l^n} \\
&\geq \omega_l^n \kappa_\triangledown \frac{1}{\alpha_l^n} \geq \omega_l^n \kappa_\triangledown \underbrace{\left(\frac{\widehat{R}_l^n}{R_l^n} \right)}_{\geq 1} \\
&\geq \omega_l^n \kappa_\triangledown
\end{aligned}$$

Hence, we finally have:

$$\omega_l^n \kappa_\triangledown \leq \widehat{\omega}_l^n \leq \omega_l^n \kappa_\triangle \left(\frac{1 + \frac{(n-1)}{(k-1)}}{1 + (n-1)\omega_l^n} \right)$$

Case 2:

Since these tiers are not part of the set of dominating tiers, the corresponding weighting factors are decreasing in both systems. In the old system, the decrease starts from ω_p^1 towards 0, whereas in the improved system it starts from $\widehat{\omega}_p^1 = \kappa \omega_p^1$. With this behavior, for an upper boundary we have:

$$\begin{aligned}
\widehat{\omega}_p^n &= \omega_p^n \frac{\kappa^n}{\alpha_p^n} \\
&\leq \omega_p^n \kappa_\triangle \frac{1}{\alpha_p^n} \leq \omega_p^n \kappa_\triangle \frac{\widehat{R}_p^n}{R_p^n} \\
&\leq \omega_p^n \kappa_\triangle \left(\frac{1 + (n-1)\widehat{\omega}_p^{n-1}}{1 + (n-1)\omega_p^{n-1}} \right) \\
&\leq \omega_p^n \kappa_\triangle \left(\frac{1 + (n-1)\kappa\omega_p^1}{1 + (n-1)\omega_p^{n-1}} \right)
\end{aligned}$$

The lower boundary is deduced in the same way as for **Case 1**.
With the lower boundary, we finally have:

$$\omega_l^n \kappa_\triangledown \leq \widehat{\omega}_l^n \leq \omega_l^n \kappa_\triangle \left(\frac{1 + \frac{(n-1)}{(k-1)}}{1 + (n-1)\omega_l^n} \right)$$

Case 3:

The improvement of the service time S_d with a speedup factor α_d results in an accelerated service time $\widehat{S}_d$, whereas it holds $\alpha_d = S_d/\widehat{S}_d$. However, this improvement also leads that tier T_d to be no more part of the dominating tiers of the improved system. Therefore, in contrast to ω_d^n, the weighting factor of the improved system $\widehat{\omega}_d^n$ decreases towards 0 starting at $\widehat{\omega}_d^1 = \omega_d^1\kappa^1/\alpha_d^1$, which is an appropriate upper boundary.
The lower boundary is deduced as follows:

$$\begin{aligned}
\widehat{\omega}_d^n &= \omega_d^n\frac{\kappa^n}{\alpha_d^n} \\
&\geq \omega_d^n\kappa_\triangledown\frac{1}{\alpha_d^n} \\
&\geq \omega_d^n\kappa_\triangle\left(\frac{\widehat{S}_d(1+(n-1)\widehat{\omega}_d^{n-1})}{S_d(1+(n-1)\omega_d^{n-1})}\right) \\
&\geq \omega_d^n\kappa_\triangle\left(\frac{1}{\alpha_d^1(1+\frac{n-1}{k})}\right)
\end{aligned}$$

□

To proceed with the next corollary, we first define another subset of tiers of the multi-tier system. We define the set of *secondary dominating tiers* with the corresponding tier indices set $\mathcal{F}_S := \{i \mid S_i = S_{max2} := max\{S_j\} \wedge j \notin \mathcal{D}_S\}$. Hence, $\mathcal{F}_S$ defines the subset of tier indices for which the tiers have the maximum service time S_{max2} among the tiers which do not belong to the dominating set $\mathcal{D}_S$. Therefore they are called secondary dominating tiers.
From this definition it follows $S_{max2} < S_{max}$ and the corresponding cardinality results in $\mathtt{card}(\mathcal{F}_S) =: c < m$.

Corollary 3. *Suppose a ($T_i \equiv Q_i$)-multi-tier system with m tiers consisting of only one dominating tier T_d with $d \in \mathcal{D}_S$ and a set of tiers T_f forming a secondary dominating tier set, $f \in \mathcal{F}_S \wedge f \notin \mathcal{D}_S$, with $c = \mathtt{card}(\mathcal{F}_S)$ and further set of tiers T_p with $p \notin \{\mathcal{F}_S, \mathcal{D}_S\}$, which do not belong to any of the two dominating tier sets.*
Further, suppose an optimization effort in the service time of the dominating tier T_d to result in α_d for the service time speedup and κ for the response time speedup at the concurrency level $n = 1$. Then for the new weighting factors $\widehat{\omega}_i^n$ of the improved system, we have the following three cases:

Case 1: $\widehat{S}_d > \widehat{S}_f = S_f$ *(T_d is still dominating tier)* $\Longrightarrow$

(i) T_p *with* $\widehat{S}_p < \widehat{S}_d$ *(remaining tiers)*

$$\omega_p^n\kappa_\triangledown \leq \widehat{\omega}_p^n \leq \omega_p^n\kappa_\triangle\left(\frac{1+(n-1)\kappa\omega_p^1}{1+(n-1)\omega_p^{n-1}}\right)$$

(ii) T_d *with* $\widehat{S}_d < S_d$ *(T_d is still dominating tier)*

$$\omega_d^n\kappa_\triangledown\left(\frac{1+(n-1)\omega_d^1\kappa^1/\alpha_d^1)}{\alpha_d^1(1+(n-1)\omega_d^{n-1})}\right) \leq \widehat{\omega}_d^n \leq \omega_d^n$$

(iii) T_f *with* $\widehat{S}_f < \widehat{S}_d$ *(T_f's are secondary dominating tiers)*
- cf. ***Case 1****-(i)*

Case 2: $\widehat{S}_d = \widehat{S}_f$ *(secondary dominating tiers including* T_d *become dominating)* $\Longrightarrow$

(i) T_p *with* $\widehat{S}_p < \widehat{S}_f = \widehat{S}_d$ *(remaining tiers)*
- cf. ***Case 1****-(i)*

(ii) T_d *with* $\widehat{S}_d = \widehat{S}_f$ *(T_d is part of dominating tier set)*

$$\omega_d^n \kappa_\bigtriangledown \left(\frac{1+(n-1)\omega_d^1 \kappa^1/\alpha_d^1)}{\alpha_d^1(1+(n-1)\omega_d^{n-1}} \right) \leq \widehat{\omega}_d^n \leq \left(\frac{1}{c+1} \right)$$

(iii) T_f *with* $\widehat{S}_p < \widehat{S}_f = \widehat{S}_d$ *(T_f become part of dominating tier set)*

$$\omega_f^1 \kappa^1 \leq \widehat{\omega}_f^n \leq \left(\frac{1}{c+1} \right)$$

Case 3: $\widehat{S}_d < \widehat{S}_f$ *(secondary dominating tiers become dominating without* $T_d \Longrightarrow$

(i) T_p *with* $\widehat{S}_p < \widehat{S}_f$ *and* $p \neq d$ *(remaining tiers)*
cf. ***Case 1****-(i)*

(ii) T_d *with* $\widehat{S}_d < \widehat{S}_f$ *(T_d is no more dominating tier)*

$$\omega_d^n \kappa_\bigtriangledown \left(\frac{1}{\alpha_d^1(1+(n-1)\omega_d^{n-1}} \right) \leq \widehat{\omega}_d^n \leq \omega_d^1 \kappa^1/\alpha_d^1$$

(iii) T_f *with* $\widehat{S}_p < \widehat{S}_f \wedge \widehat{S}_d < \widehat{S}_f$ *(T_f become dominating tier set)*

$$\omega_f^n \kappa_\bigtriangledown \left(\frac{1+(n-1)\omega_f^1 \kappa^1}{1+(n-1)\omega_f^{n-1}} \right) \leq \widehat{\omega}_f^n \leq \left(\frac{1}{c} \right)$$

Proof. In principle, the proof is similar to the proof of Corollary 2.

Case 1:

(i) The weighting factors of the tiers T_p are still decreasing. However, due to an improvement in the dominating tier T_d it comes to a load redistribution, which causes the load at the tiers T_p to increase. Hence, the processing time $\widehat{R}_p^n$ is always larger or equal than the processing of the old system R_p^n.

$$\widehat{\omega}_p^n = \omega_p^n \kappa^n \left(\frac{\widehat{S}_p(1+(n-1)\overbrace{(\widehat{\omega}_p^{n-1})}^{\leq \widehat{\omega}_p^1})}{S_p(1+(n-1)\omega_p^{n-1})} \right) \leq \omega_p^n \kappa_\bigtriangleup \left(\frac{1+(n-1)\kappa\omega_p^1}{1+(n-1)\omega_p^{n-1}} \right)$$

$$\widehat{\omega}_p^n = \omega_p^n \kappa^n \left(\frac{1+(n-1)\widehat{\omega}_p^{n-1}}{1+(n-1)\omega_p^{n-1}} \right) \geq \omega_p^n \kappa_\bigtriangledown \left(\frac{1}{1+(n-1)\omega_p^{n-1}} \right)$$

(ii) T_d is still the only dominating tier, but the improvement results in the reduction of its weighting factors. Hence, we have:

$$\begin{aligned} \widehat{\omega}_d^n &\le \omega_d^n \\ \widehat{\omega}_d^n = \omega_d^n \kappa^n \left(\frac{1+(n-1)\,\overbrace{(\widehat{\omega}_d^{n-1})}^{\ge \widehat{\omega}_d^1}}{\alpha_d^1(1+(n-1)\omega_d^{n-1})} \right) &\ge \omega_d^n \kappa_\nabla \left(\frac{1+(n-1)\omega_d^1\kappa^1/\alpha_d^1}{\alpha_d^1(1+(n-1)\omega_d^{n-1}})\right) \end{aligned}$$

(iii) Since T_d is still the only dominating tier, the weighting factors of all other tiers are decreasing, which is therefore also true for the secondary dominating tiers. Therefore, the estimations in **Case 1**-(i) are also valid for the secondary dominating tiers.

Case 2:

(i) This is the same situation as in **Case 1**-(i).

(ii) T_d is no more the only dominating tier. Instead, together with the originally secondary dominating tiers, it forms a new dominating set of tiers. Therefore, all the weighting factors of this dominating tier set are increasing towards $1/(c+1)$. Hence, for T_d we have the following boundaries:

$$\begin{aligned} \widehat{\omega}_d^n &\le \left(\frac{1}{c+1}\right) \\ \widehat{\omega}_d^n = \omega_d^n \kappa^n \left(\frac{1+(n-1)\,\overbrace{(\widehat{\omega}_d^{n-1})}^{\ge \widehat{\omega}_d^1}}{\alpha_d^1(1+(n-1)\omega_d^{n-1})} \right) &\ge \omega_d^n \kappa_\nabla \left(\frac{1+(n-1)\omega_d^1\kappa^1/\alpha_d^1}{\alpha_d^1(1+(n-1)\omega_d^{n-1}})\right) \end{aligned}$$

(iii) Since T_d is still the only dominating tier, the weighting factors of all other tiers are decreasing, which is therefore also true for the secondary dominating tiers. Therefore, the estimations in **Case 1**-(i) are also valid for the secondary dominating tiers.

(iii) As in this case the tiers T_g form with T_d the set of the dominating tiers, the upper boundary is already deduced in **Case 2**-(ii).

$$\begin{aligned} \widehat{\omega}_g^n &\le \left(\frac{1}{c+1}\right) \\ \widehat{\omega}_g^n = \omega_g^n \kappa^n \left(\frac{1+(n-1)\,\overbrace{(\widehat{\omega}_g^{n-1})}^{\ge \widehat{\omega}_g^1}}{1+(n-1)\omega_g^{n-1}} \right) &\ge \omega_g^n \kappa_\nabla \left(\frac{1+(n-1)\omega_g^1\kappa^1}{1+(n-1)\omega_g^{n-1}}\right) \end{aligned}$$

Case 3:

(i) This is same situation as in **Case 1**-(i).

(ii) In this case, T_d is part of the set, consisting of non-dominating tiers. Therefore, with increasing concurrency level n, its weighting factor decreases starting from $\widehat{\omega}_d^1$ towards 0. For the weighting factor boundaries this results in:

$$\begin{array}{rcl} \widehat{\omega}_d^n & \leq & \omega_d^1 \kappa^1 / \alpha_d^1 \\ \widehat{\omega}_d^n = \omega_d^n \kappa^n \left(\dfrac{1+(n-1)\widehat{\omega}_d^{n-1}}{\alpha_d^1(1+(n-1)\omega_d^{n-1}} \right) & \geq & \omega_d^n \kappa_\triangledown \left(\dfrac{1}{\alpha_d^1(1+(n-1)\omega_d^{n-1}}) \right) \end{array}$$

(iii) In this case, the secondary dominating tiers have become a dominating tier set. Hence, the corresponding weighting factors are increasing towards $1/c$, whereas the boundaries are deduced as follows:

$$\begin{array}{rcl} \widehat{\omega}_f^n & \leq & \left(\dfrac{1}{c} \right) \\ \widehat{\omega}_f^n = \omega_f^n \kappa^n \left(\dfrac{1+(n-1)\overbrace{(\widehat{\omega}_f^{n-1})}^{\geq \widehat{\omega}_f^1}}{1+(n-1)\omega_f^{n-1}} \right) & \geq & \omega_f^n \kappa_\triangledown \left(\dfrac{1+(n-1)\omega_f^1 \kappa^1}{1+(n-1)\omega_f^{n-1}} \right) \end{array}$$

□

In summary, these formulas can be used to predict and understand the resulting effects of an improvement effort. To conclude the discussion so far, based on the monotonous behavior of the MVA algorithm, we have derived the following estimation tools, which can support efficient approaches to optimize multi-tier systems:

- Theorem 3, which provides a quick tool to estimate the asymptotic boundaries of the response time of a multi-tier system.
- Lemma 2, which constitutes another tool to approximate the asymptotic boundaries of the resulting response time speedup factor.
- Corollary 2 and Corollary 3, which enable to estimate the resulting load in individual tiers.

The fundamental base behind these formulas is provided by Theorem 2, which describes how the load within a $(T_i \equiv Q_i)$-multi-tier system is distributed for increasing concurrency level n.

In contrast to other research efforts related to the MVA algorithm (cf. Section 2.7), our methodology does not require the implementation and execution of a MVA-based algorithm. Instead, by analyzing different improvement cases, which can occur after an optimization in the multi-tier system, we have provided appropriate estimation formulas. The core benefit of these formulas is that they are free of recursions and hence, applicable without a particular algorithm implementation.

4.4 Model Limitations

In this section, the limits of the above modeling approach are described. In principle, there are four possible limiting aspects. The first aspect is the modeling of a tier as a single queue. Another aspect is our consideration of a tier to only offer a single operation. A third constraint is the assumption of an unlimited queuing capacity at each tier. All these aspects can lead the load within the multi-tier system to be differently distributed than we have examined with our $(T_i \equiv Q_i)$ modeling approach. Note that our modeling approach has revealed a monotonic load distribution

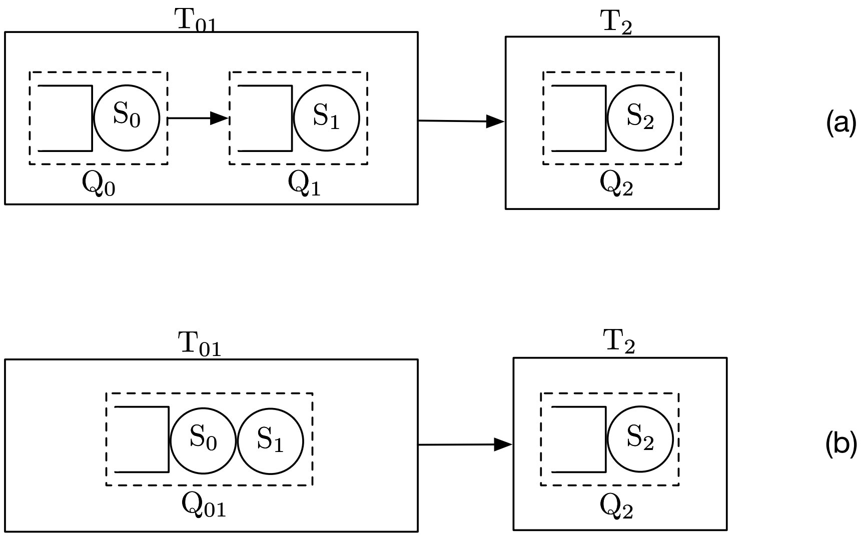


Figure 4.4: Two-Tier System: (a) Modeling as three-queue system. (b) Modeling as two-queue system.

behavior, where the load is increasingly concentrated at the tiers with the longest service times, which we defined to be the dominating tiers. Therefore, we provide insight into the these three aspects, which can lead to a different load distribution behavior.
The final constrain is based on the architecture of the multi-tier system: Our approach is based on a multi-tier system, which preserves the number of tiers m after the optimization endeavor.

4.4.1 Multiple Queues per Tier

The most important assumption for these formulas is to model each tier with a single queue ($T_i \equiv Q_i$). However, this is not always valid, since there are often also tiers, which actually incorporate multiple queues for various internal resources, also known as *simultaneous resource possession*.
Modeling such a tier as a single queue would lead its simulated processing time to be superposed with redundant queuing times. The result would be that the simulated processing time and therefore, also the simulated overall response time would be longer than the exact values. This effect of redundant queuing time superposition can be seen by the following explanation:

Given a multi-tier system composed of the two tiers T_{01} and T_2. Further, suppose tier T_{01} to incorporate two resources, while tier T_2 only includes a single resource. Figure 4.4 depicts two possible models of the overall multi-tier system. In case (a), for tier T_{01} each resource is modeled as an independent queue Q_0 and Q_1 with the corresponding service times S_0 and S_1, while in case (b), for tier T_{01}, both resources are modeled as a single queue Q_{01} with a service time $S_{01} = S_0 + S_1$. To reveal the effect of redundant queuing times, it is sufficient to apply the MVA algorithm for only $n = 2$:

Case (a):
$n = 1:$

$$\begin{aligned}
R_0^1 &= S_0\\
R_1^1 &= S_1\\
R_2^1 &= S_2\\
\Rightarrow R_T^1 &= R_0^1 + R_1^1 + R_2^0\\
L_0^1 &= S_0/R_T^1\\
L_1^1 &= S_1/R_T^1\\
L_2^1 &= S_2/R_T^1
\end{aligned}$$

$n = 2:$

$$\begin{aligned}
R_0^2 &= S_0(1+L_0^1)\\
R_1^2 &= S_1(1+L_1^1)\\
R_2^2 &= S_2(1+L_2^1)\\
\Rightarrow R_T^2 &= R_0^2 + R_1^2 + R_2^2
\end{aligned}$$

Case (b):
$n = 1:$

$$\begin{aligned}
R_{01}^1 &= S_0 + S_1\\
\\
R_2^1 &= S_2\\
\Rightarrow R_T^1 &= R_{01}^1 + R_2^0\\
L_{01}^1 &= (S_0+S_1)/(R_T^1)\\
\\
L_2^1 &= S_2/R_T^1
\end{aligned}$$

$n = 2:$

$$\begin{aligned}
R_{01}^2 &= (S_0+S_1)(1+L_{01}^1)\\
\\
R_2^2 &= S_2(1+L_2^1)\\
\Rightarrow R_T^2 &= R_{01}^2 + R_2^2
\end{aligned}$$

For $n = 1$, the response times are equal for both modeling approaches. However, for $n = 2$, the processing time R_{01}^2 is larger than $R_0^2 + R_1^2$, due to the redundant queuing times in R_{01}^2:

$$\begin{aligned}
R_{01}^2 &= (S_0+S_1)(1+L_{01}^1) = S_0 + S_1 + S_0 L_{01}^1 + S_0 L_{01}^1\\
&= S_0 + S_1 + S_0\left(\frac{S_0+S_1}{R_T^1}\right) + S_1\left(\frac{S_0+S_1}{R_T^1}\right)\\
&= S_0 + S_0\left(\frac{S_0}{R_T^1}\right) + S_0\left(\frac{S_1}{R_T^1}\right) + S_1 + S_1\left(\frac{S_1}{R_T^1}\right) + S_1\left(\frac{S_0}{R_T^1}\right)\\
&= S_0(1+L_0^1) + S_0\left(\frac{S_1}{R_T^1}\right) + S_1(1+L_1^1) + S_1\left(\frac{S_0}{R_T^1}\right)\\
&\Rightarrow R_0^2 + R_1^2 + \boxed{S_0\left(\frac{S_1}{R_T^1}\right)} + \boxed{S_1\left(\frac{S_0}{R_T^1}\right)}
\end{aligned}$$

Generally, the problem of redundant queuing times stems from the fact that the formula in the MVA algorithm for computing MVA processing times (line 8 in Algorithm 1 or equation (4.2.6) is not an additive map:
Let define equation (4.2.6), to be the *processing time function* as follows:

$$p_t(S,n) := S_i + \frac{(n-1)}{R_T^{n-1}} S_i^{(2)} + \frac{(n-1)(n-2)}{R_T^{n-1} R_T^{n-2}} S_i^{(3)} + \frac{(n-1)(n-2)(n-3)}{R_T^{n-1} R_T^{n-2} R_T^{n-3}} S_i^{(4)} + \cdots + \frac{(n-1)!}{R_T^{n-1} \cdots R_T^1} S_i^{(n)}$$

By merging any two queues with service times S_a and S_b of the same overall multi-queue system into a single queue with $S_{ab} = S_a + S_b$, for $n > 1$ yields:

$$p_t(S_a + S_b, n) > p_t(S_a, n) + p_t(S_b, n).$$

This in turn is based on $S_a^{(n)} + S_b^{(n)} < (S_a + S_b)^{(n)}$, for $n > 1$. Thus, the queuing times are based on the power series of the service times S_a and S_b, while the redundant queuing times are induced by the mixed terms $(S_a + S_b)^{(n)} - S_a^{(n)} - S_b^{(n)}$.

In principle, multiple successive queues per tier mean to increase the number of the overall function block chaining, which compose the system. And although, the resulting system can not be considered as a ($T_i \equiv Q_i$)-multi-tier system, it can be considered as queuing network of iterative queues. This ultimately means, that our deduced formulas are still valid for such a iterative queuing network, even when the number of queues is higher than the number of tiers. If the number of iterative queues with the corresponding service times per tier is known, our approach would be to model each tier as a sub queuing network. All these sub queuing networks, would then again form an overall iterative queuing network composed of $M_{all} > m$ queues, where $m \equiv$ number of tiers. Our formulas can then be applied on this queuing network.

4.4.2 Multiple Operations per Tier

Another aspect, in addition to multiple queues per tier, is that individual tiers can also provide different operations, which do not involve all available internal queues or even subsequent tiers. Such tiers are characterized by different processing times for the same concurrency level n. The common approach is to model this behavior with transition probabilities: Within each tier, each processing path is associated with a certain probability. This results in multiple possible processing times at an individual tier, which ultimately means that there are also multiple possible overall response times.
To apply our approach to such a behavior, one strategy would be to determine the dominating tier set for a request processing path (from the first tier until the last tier). Since the overall monotonic behavior would be still existing, the dominating tier set would still dominate the response time per request processing path. And since there are multiple paths, each dominating set would be again associated with a specific probability.
However, in this thesis, we focus on transactional multi-tier systems, which involve all available tiers for a successful request processing. Such systems are commonly in use in the current Internet, which is particularly true for distributed systems applying the master/slave principle. The master node is usually used to perform uniform writing operations, whereas the slave nodes are used for backup and reading operations. Commonly, there is only a single master node, which is then replicated by multiple slave nodes. Another important purpose of such an architecture is to distribute the reading operation from the master to the slave nodes. In effect, the master node is then mainly subjected to writing operations. Prime examples for such an architecture are the DNS system and also various PID systems such as the Handle System. This means, that our approach is still applicable for a broad group of multi-tier Internet systems.

4.4.3 Limited Queuing Capacity

A further basic assumption in our approach is an unlimited queuing capacity in all queues. However, in practice all resources are usually limited available. Since the goal of our approach is to provide a prediction methodology to understand the asymptotic behavior of the system rather than a simulation technique, which provides more fine grained information about the system's measures, the assumption of an unlimited queuing capacity can not be considered as a drawback.
Our approach enables an estimation of the increased load at individual tiers after an improvement at a particular tier. These estimations in turn, can be used by an individual developer to analyze whether the capacities at specific tiers or queues could become overloaded. Therefore, the weighting factor boundary analysis provided in Section 4.3.4.2 is perfectly suitable for such a capacity utilization analysis.
The typical behavior when a tier becomes overloaded is that a part of the incoming requests will be discarded. The handling of such situations is highly dependent on the implementation. Previous

tiers can either employ a sophisticated retry mechanism or directly send back a response to the requesting application containing an error message. In either case, the overall response time for a successful processing increases rapidly. Hence, our approach enables to detect and solve such a problematic situation, rather than to model the rapid increase of the response time.

4.4.4 Architecture Preservation

An implicit assumption of our approach is that the architecture of the multi-tier system consisting of m tiers is preserved in the resulting improved system. This means that the improved system is again a multi-tier system consisting of m tiers and that the respective improved procedure still involves all the same tiers as in the unimproved system.
Our approach is not applicable for improvement endeavors, which result in a different system architecture and request processing path.

4.5 Evaluation

In this section, we provide an evaluation of our approach, which is conducted on the ePIC PID system.
The European Persistent Identifier Consortium (ePIC) [97] was founded to provide PIDs for the research community. Currently, the ePIC PID system is used by several research communities. However, as the amount of research datasets increase, also the requirements on the PID systems increase, which is especially true for the performance.
As already described in Section 3.2.2, the ePIC PID system is fundamentally based on the underlying Handle System. The ePIC PID system provides additional functions on top of the Handle server to realize an easily adoptable REST interface and to ensure the registered PID records, which are stored as Handle Records, to comply with a certain metadata scheme.
However, the additional data logic component, provided by the ePIC PID system, on top of the Handle System, comes at the expense of the performance. The increasing demand for PIDs therefore compels the ePIC consortium to improve its system in order to be able to meet the rising performance requirements.
Since the ePIC PID system is basically a transactional multi-tier Internet system, it is suitable to evaluate our approach.

4.5.1 ePIC PID System

The ePIC consortium currently consists of six European members, where the GWDG is a founding member and currently (year 2018) also chairing the consortium. Each of the ePIC members is hosting several instances of the ePIC PID system, whereby each instance operates on top of a dedicated primary Handle server. In addition, each of the primary Handle servers is replicated by at least one mirror Handle server, which is hosted by another ePIC member. Hence, all ePIC-PID administration operations are performed via the ePIC PID system at the primary Handle server. Since ePIC-PIDs are Handle-PIDs, their resolution is accomplished over the global resolution system provided by the Handle System without involving the ePIC PID system. It should be noted that Handle-PIDs with a `21.XXXXX`-prefix are homed at PID systems, which are controlled by the ePIC consortium.
To resolve a Handle-PID, the resolution request is delegated starting from the global root Handle resolver (`hdl.handle.net`) towards the responsible *Local Handle Service* (LHS) (cf. Section 3.3.1). Due to caching at several nodes and the mirroring mechanism, the primary Handle

server is usually predominantly stressed by administration operations. Therefore, for this evaluation, we only consider the improvement of the PID administration operation. This administration operation was already described in the previous chapter by Listing 3.1.
In the following, we provide insight into the multi-tier composition of the ePIC PID system, which is additionally depicted in Figure 4.5.

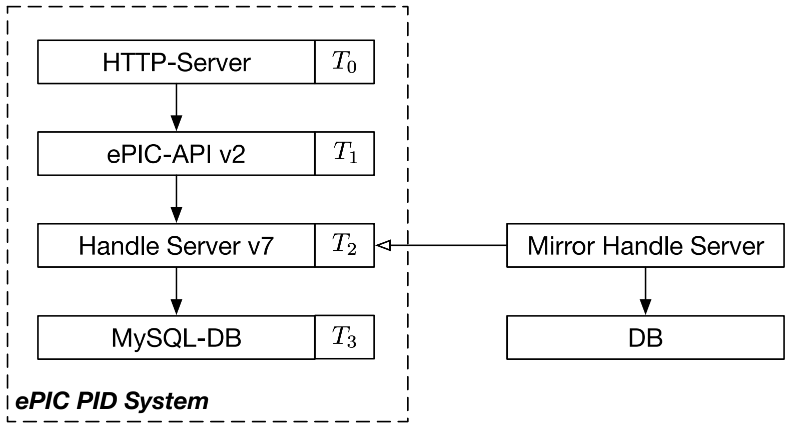


Figure 4.5: ePIC PID System

HTTP-Server (T_0): This is the first tier of the ePIC PID system. It is a JRuby[10] HTTP-Server, which translates incoming HTTP-requests into Ruby objects and likewise outgoing Ruby objects into HTTP-responses. In terms of Figure 4.2, this is the front-end tier.

ePIC-API v2 (T_1): The second tier, tier T_1, is the ePIC-API v2, which is the most important component of the overall ePIC PID system. The incoming requests (Ruby objects) are first examined for further data processing before the requests are transmitted to the Handle server via the native Handle protocol. The goal of the data processing at the ePIC-API v2 is to ensure the incoming datasets to comply with an individually defined data structure. The definition of a data structure is accomplished by the so called *ePIC-Profiles* module. In terms of Figure 4.2, this represents the pre-processing tier.

Handle Server v7.3.1 (T_2): The primary Handle server registers the incoming datasets as Handle Records into the attached database. Incoming requests are first decoded from the Handle protocol encoding into Java objects. With these Java objects, the transactional administration workflow is launched, where the final step is to store the Handle Record permanently into the attached database. In terms of Figure 4.2, this is mainly the integration but also the post-processing tier.

MySQL Database (T_3): The database is the final tier of the ePIC PID system. It basically stores all the data of the overall PID system into the underlying storage system. Usually, all the ePIC PID system instances are attached to a MySQL database with *InnoDB* as the underlying storage engine. InnoDB is characterized by ensuring a transactional processing in the MySQL database. In terms of Figure 4.2, this is finally the back-end tier.

It is the Handle server, which actually makes the ePIC PID system to a PID system. Since the

[10]http://jruby.org

entirety of all running Handle servers form a global overlay network with a common communication protocol, by a registration at a primary Handle server, an individual PID becomes globally resolvable. This the reason, why the Handle server represents the integration tier in terms of Figure 4.2.

4.5.2 Evaluation Methodology

As previously mentioned, in our evaluation, we only focus on the optimization of the PID record administration operation. This administration operation of the ePIC PID system involves all the tiers $T_0, \cdots, T_3$. Our evaluation methodology was to determine the impact of an improvement of one these tiers.

4.5.3 Evaluation Environment

Our evaluation covered two different hosting environments. The first hosting environment was a MacBook Air with 1.7 GHz Intel Core i7, 8 GB Ram and a SSD hard disk. This is reasonable, because system developers usually first implement, deploy and evaluate changes within their own working environment. In addition, we are mainly interested in the evaluation of the deduced formulas, rather than on the system's behavior on a specific hosting environment.
Nevertheless, we also made use of the productive hosting environment: At GWDG a productive ePIC PID system instance is hosted in a virtualized environment. The virtualization paradigm itself, provides much more efficiency and flexibility for service provides, however, at the same time it complicates the behavior of both, the underlying hardware and software systems. Therefore, the first hosting environment can also be considered as an *ideal host*. Hence, as the second hosting environment, we made use of a dedicated ePIC PID system instance from the productive environment.
In principle, the primary difference between the first and second environment is the computing power and the SSD hard disk. On the MacBook, the CPU is only shared between various parallel processes. In contrast to that, in the virtual host, the hardware CPU is additionally shared between multiple other virtual hosts.
However, in both environments, the databases of the productive systems were used, especially for the database system at tier T_3. This was a MySQL database instance from the special GWDG MySQL database cluster, the MySQL Appliance [11]. But also within tier T_1, to generate unique identifiers based on a counter, where the state of the counter was stored in a MySQL database.

To stress our testing ePIC PID systems with appropriate workload, we made use of a self-built load-generator. This load-generator steadily issued PID creation requests for sample research datasets extracted from the productive PID databases of GWDG. Every five minutes the count of parallel threads for creating PIDs was increased by one. Thus, in the first five minutes, only one thread was active, in the last five minutes the PID system was stressed by ten concurrent threads. It should be noted, that each productive ePIC instance at GWDG is usually shared by multiple research data repositories (dedicated prefix). Thus, the concurrency level is often between five and seven. But there are also many periods of time with a concurrency level of $n = 1$.

4.5.4 Improvement Analysis

In order to identify the problematic components of the ePIC PID system, we performed an extensive measurement analysis. This allowed us to locate the problems and to improve the ePIC PID

[11] https://www.gwdg.de/application-services/database-service-mysql

ID	Alias	S[ms]	$\widehat{S}$[ms]
T_0	HTTP-SERVER	1	1
T_1	EPIC	37	9
T_2	HS	7	7
T_3	DB	25	25
response time R_T		70	42

Table 4.1: **Ideal Host**: Service times for each tier of the old and improved system

system. During this analysis it turned out, that the ePIC-API v2 (T_1) was causing a high contribution into the overall response time. Therefore, our improvement effort was particularly focused on several components in the ePIC-API v2. These critical components can be categorized into logging, identifier generation and response handling.
The logging component was quite inefficiently involving unnecessary many I/O operations. This is turn caused many CPU context switches, which finally resulted in long processing times.
Also the identifier generator for creating unique counter-based identifiers was causing high overheads in the ePIC-API v2 tier, which was mainly because of a database access to store the current state of the counter.
In addition to that, the response handling was highly inefficient as well. The problem in this component was that in a successful processing, the response object was generated and handled in an exceptional processing path, which also caused high CPU context switches and hence, long processing times.
We were able to optimize all these critical components. The result of our optimization endeavor is implemented in version 2.5 of the ePIC-API, and can be found in the source code repository of the ePIC consortium[12]. Additionally, the PID `11022/0000-0007-C618-F` resolves directly to our contributions, which have been accepted by the ePIC consortium for productive operation.

4.5.5 Ideal Host Measurements Analysis

In this section, the focus is on the analysis of the measurements from the ideal host. For this, we provide the most important measurements of the two versions of the ePIC PID system. The first version is the unimproved system with version 2.3 of the ePIC-API v2. Whereas, the second version is the result of our improvement effort, which has been done in T_1, the ePIC-API v2.

4.5.5.1 Service Time Improvement

As can been seen from Table 4.1, the speedup factor achieved by our improvement effort in tier T_1 is $\alpha_1 = 37/9 \approx 4.11$.
For the overall response time improvement for the concurrency level $n = 1$, this results in $\kappa = 70/42 \approx 1.67$. The highest possible speedup would equal to $70/33 \approx 2.12$, which could be achieved by improving tier T_1 such that it would cause no overhead ($\widehat{=}\ \widehat{S_1} = 0$) .

4.5.5.2 Response Time Decomposition

In Figure 4.6, we can see the response time decomposition for increasing concurrency level n. The overall response time R_T^n is composed of the processing times R_i^n, which are shown with different colors on the bar plots. Note that the hatched bars denote the measurements of the improved

[12]https://github.com/pidconsortium/EPIC-API-v2

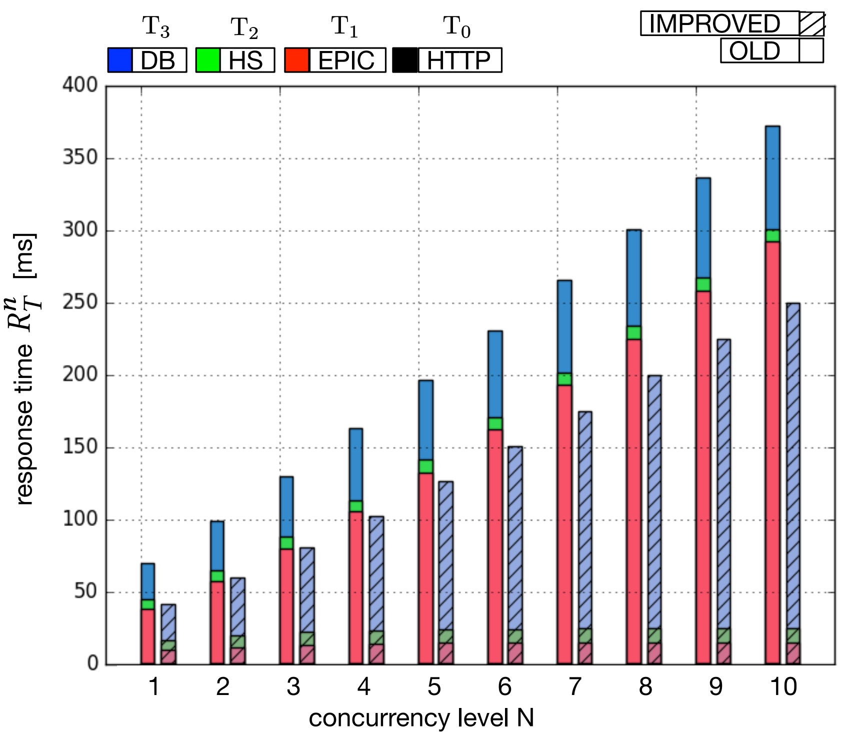


Figure 4.6: **Ideal Host**: Response Time Decomposition for increasing concurrency level. The response time is composed of the processing times of the tiers. Each processing time is highlighted as a colored part on the response time bar.

system. It should also be noted that due to the small values, the processing times of T_0 and $\widehat{T_0}$ are only barely recognizable. In addition to the response time, Figure 4.7 shows the respective lower and upper boundaries, which are deduced by Theorem 3. We can see that the actual response times always lie within these boundaries, which underlines the practical utility of Theorem 3. Finally, as expected, the overall response time R_T^n is increasing with the concurrency level n.

4.5.5.3 Weighting Factor Decomposition

From Figure 4.8, we can see the weighting factor decomposition for increasing concurrency level. The most important development in this figure is that the weighting factors ω_1^n for the ePIC-API tier (dominating tier of unimproved system) and $\widehat{\omega}_3^n$ for the database tier (dominating tier of improved system) are increasing with the concurrency level n, while the respective remaining weighting factors are decreasing. This is exactly the monotonicity behavior, which is described by Theorem 2.

4.5.5.4 Course of the Response Time Speedup

In our case it holds $\kappa = 1.67 > 1.48 = S_{max}/\widehat{S}_{max}$, this means that κ^n decreases towards $\kappa^\infty = 1.48$, which is therefore a lower boundary $\kappa_\bigtriangledown = 1.48$. This can also be seen from Figure 4.9. Based on

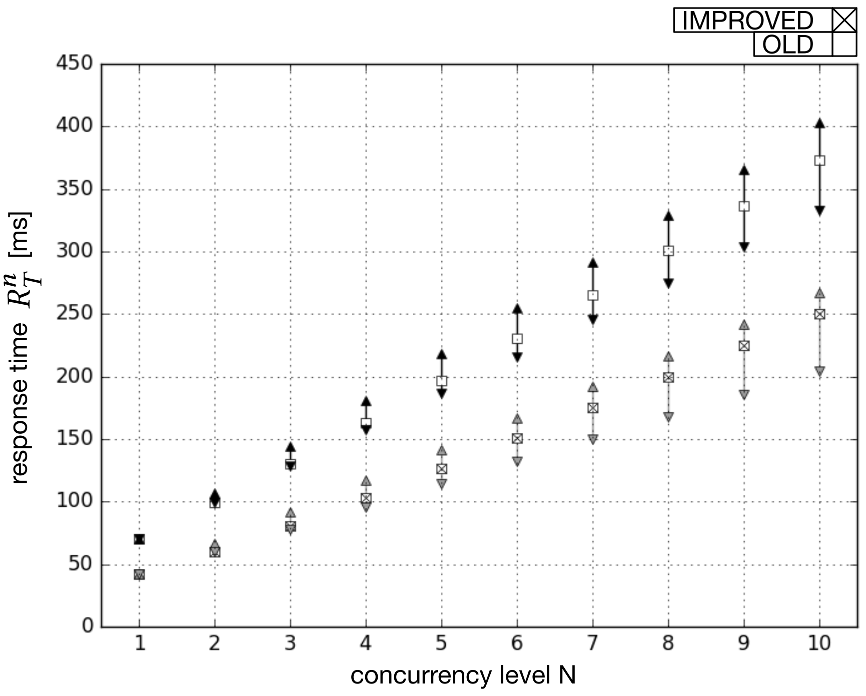


Figure 4.7: **Ideal Host**: Response Time R_T for increasing request count including lower and upper boundaries $R_\bigtriangledown$ and $R_\bigtriangleup$ provided by Theorem 3. The actual response times are plotted as squares, which are limited by an upper (↑) and lower (↓) arrow denoting the estimated boundaries ($R_\bigtriangleup$ and $R_\bigtriangledown$).

equation (4.3.10) in the proof of Lemma 2, for an upper boundary for κ^n, we can take:

$$\kappa^n_\bigtriangleup = \frac{R^n_T}{\widehat{R}^1_T + (n-1)\sum_{i=1}^m \left(\widehat{S}_i^{(2)}/\widehat{R}^1_T\right)} = \left(\frac{R^n_T}{42+18(n-1)}\right)$$

As an example, for $n = 5$, we have $R^5_T \approx 196ms$ and $\widehat{R}^5_T \approx 126ms$, which results in $\kappa^5 \approx 1.555$. Another example with $n = 10$, $R^{10}_T \approx 372ms$ and $R^{10}_T \approx 250ms$, leads to $\kappa^{10} \approx 1.488$.

4.5.5.5 Weighting Factor Redistribution

Table 4.2 and Table 4.3 show the exact weighting factors of the old and improved system respectively. In addition, Table 4.4 provides estimations for the weighting factors of the improved (future) system, which are based on our derived formulas from Section 4.3.4.2. Since in our setup we have only one dominating tier T_1 with $S_{max} = S_1$ ($\mathcal{D}_S = \{1\}$), we have to apply Corollary 3. The secondary dominating tier set also consists of only one tier: T_3 with $S_{max2} = S_3$ ($\mathcal{F}_S = \{3\}$). Furthermore, since our improvement effort in T_1 has resulted in $\widehat{S}_1 < S_{max2}$, to retrieve estimations for the weighting factors of the improved (future) system, we have to apply **Case 3** of Corollary 3. The estimations for T_0 and T_2 are based on **Case 3**-(i). In contrast to that, the estimations for the old dominating tier T_1 are calculated by means of **Case 3**-(ii). Finally, the estimations of the new dominating tier T_3 are determined by **Case 3**-(iii).
In Table 4.4, the values in the grey columns denote the actual weighting factors. Each column to the left of the grey column contains lower boundaries, where the columns to the right represent the

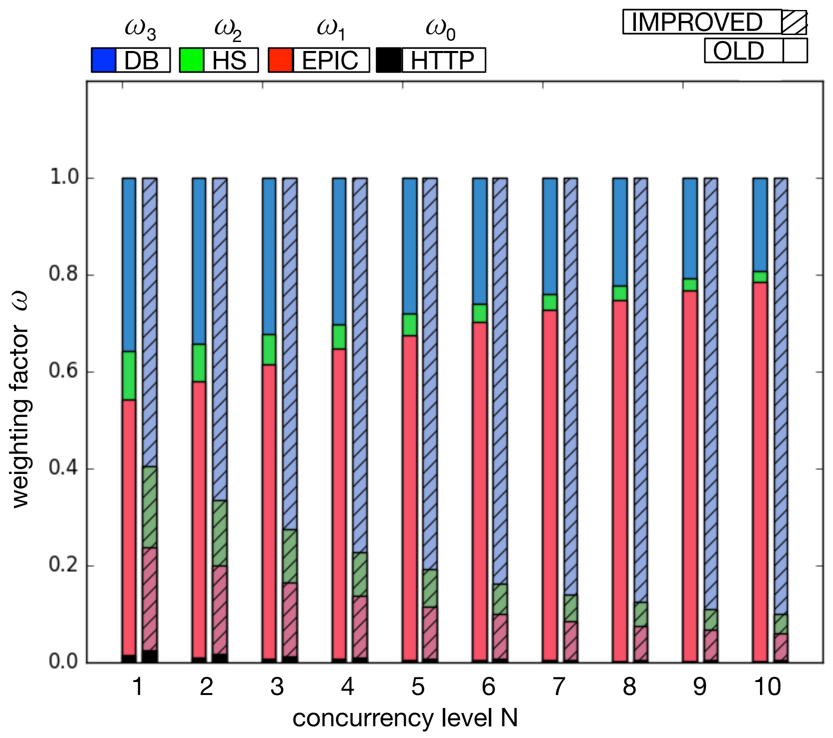


Figure 4.8: **Ideal Host**: Weighting factor decomposition for increasing concurrency level.

n	T_0	T_1	T_2	T_3
1	0.0143	0.5286	0.1	0.3571
2	0.0102	0.5701	0.0776	0.342
3	0.0078	0.6073	0.062	0.3229
4	0.0063	0.6407	0.0509	0.302
5	0.0052	0.6709	0.0429	0.281
6	0.0044	0.6982	0.0368	0.2605
7	0.0039	0.7228	0.0322	0.2412
8	0.0034	0.7448	0.0285	0.2233
9	0.003	0.7646	0.0255	0.2069
10	0.0028	0.7822	0.0231	0.1919

Table 4.2: **Ideal Host**: Weighting factors for the unimproved system.

upper boundaries. Ultimately, we can see that our deduced boundaries limit the actual weighting factors.

This means, by only having the weighting factors ω_i^n of the unimproved system at hand, together with the improvement parameters for $n = 1$ (κ, κ_∇, $\kappa_\triangle$, α_1), we were able to predict the weighting factors of the improved (future) system for concurrency levels $n > 1$, without the need to conduct extensive performance evaluations with the improved system.

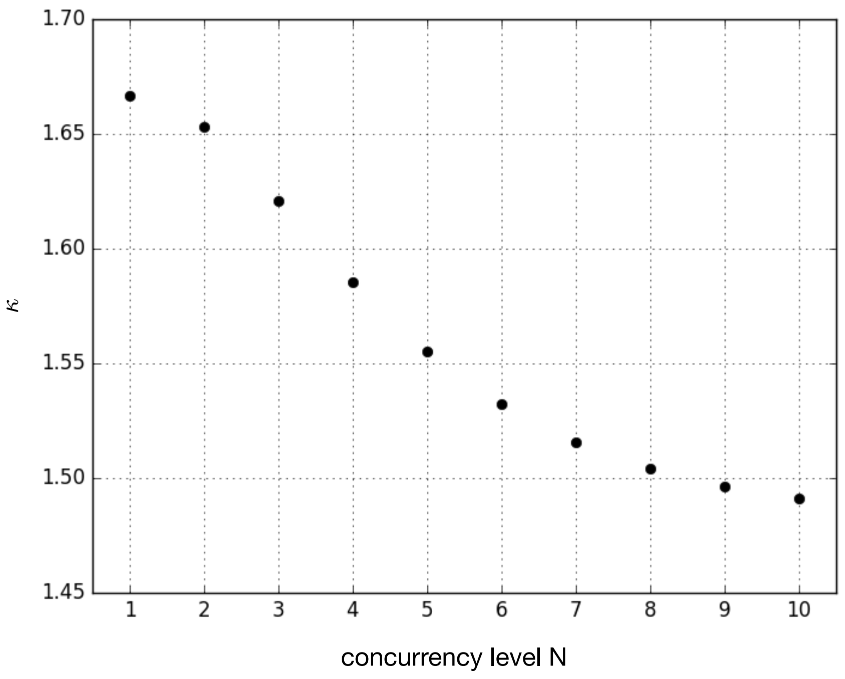


Figure 4.9: **Ideal Host**: Response time speedup for increasing concurrency level with the improvement of tier T_1.

n	$\widehat{T_0}$	$\widehat{T_1}$	$\widehat{T_2}$	$\widehat{T_3}$
1	0.0238	0.2143	0.1667	0.5952
2	0.0171	0.1822	0.1361	0.6647
3	0.0129	0.1526	0.1107	0.7238
4	0.0101	0.1277	0.0907	0.7715
5	0.0082	0.1076	0.0755	0.8086
6	0.0069	0.0919	0.064	0.8371
7	0.0059	0.0797	0.0553	0.8591
8	0.0052	0.0701	0.0485	0.8762
9	0.0046	0.0624	0.0432	0.8898
10	0.0042	0.0562	0.0389	0.9007

Table 4.3: **Ideal Host**: Weighting factors for the improved system.

4.5.5.6 Overloading

An important aspect, which has to be taken into account within an improvement endeavor, is the effect of load redistribution. This effect is caused by an improvement at an individual tier and can ultimately lead to an overloading of limited queuing capacities at the remaining tiers.
The thread pool sizes of the components of the ePIC PID system are usually set to sufficient high threshold values. However, if for example the thread pool size of the database at tier T_3 would had been set to three threads, our approach would enable to predict an overloading at the database although a limited queuing capacity is not directly addressed by our modeling approach. For the productive concurrency level $5 \leq n \leq 7$, this is described in Table 4.5: We can see that the

	T_0			T_1			T_2			T_3		
n	$\omega_\bigtriangledown$	$\widehat{\omega}$	$\omega_\bigtriangleup$	$\omega_\bigtriangledown$	$\widehat{\omega}$	$\omega_\bigtriangleup$	$\omega_\bigtriangledown$	$\widehat{\omega}$	$\omega_\bigtriangleup$	$\omega_\bigtriangledown$	$\widehat{\omega}$	$\omega_\bigtriangleup$
1	0.0212	0.0238	0.0238	0.1903	0.2143	0.2143	0.148	0.1667	0.1667	0.5285	0.5952	1
2	0.0151	0.0171	0.0172	0.1343	0.1822	0.2143	0.1148	0.1361	0.1372	0.5949	0.6647	1
3	0.0115	0.0129	0.0133	0.1022	0.1526	0.2143	0.0918	0.1107	0.1192	0.6216	0.7238	1
4	0.0093	0.0101	0.0109	0.0817	0.1277	0.2143	0.0753	0.0907	0.1073	0.6324	0.7715	1
5	0.0077	0.0082	0.0092	0.0678	0.1076	0.2143	0.0635	0.0755	0.099	0.6368	0.8086	1
6	0.0065	0.0069	0.0079	0.0577	0.0919	0.2143	0.0545	0.064	0.0925	0.6374	0.8371	1
7	0.0058	0.0059	0.0073	0.0501	0.0797	0.2143	0.0477	0.0553	0.0879	0.6367	0.8591	1
8	0.0050	0.0052	0.0064	0.0442	0.0701	0.2143	0.0422	0.0485	0.0839	0.6351	0.8762	1
9	0.0044	0.0046	0.0058	0.0396	0.0624	0.2143	0.0377	0.0432	0.0807	0.6332	0.8898	1
10	0.0041	0.0042	0.0055	0.0357	0.0562	0.2143	0.0342	0.0389	0.0782	0.6308	0.9007	1

Table 4.4: **Ideal Host**: Weighting factor estimations. The grey columns contain the exact values, whereas the respective columns to the left and right contain lower and upper boundaries respectively.

database of the improved system is populated by at least 4 requests for $5 \leq n \leq 7$. Whereas with our formulas, we were able to predict a minimum request population of $L_{\bigtriangledown DB} = 3.2$ at the database (T_3), after the improvement of tier T_1. Note that in the unimproved system, the request population at the database for $5 \leq n \leq 7$ was only between 1.4 and 1.7 requests.
The effect, which occurs at overloading is usually a sharp increase in the response time. This can be particularly seen in Figure 4.10, which depicts the response time behavior of a modified version of the improved ePIC PID system hosted on an Amazon EC2 instance. The core difference was that the MySQL database attached to the Handle server, was installed locally on the EC2 host. In addition, we set the thread pool size of the database to three threads.
We can see a sharp rise of the response time at the concurrency level $n = 5$.

n	$L_{\bigtriangledown DB}$	L_{DB}	$L_{\bigtriangleup DB}$
5	3.2	4.0	5
6	3.8	5	6
7	4.5	6	7

Table 4.5: **Ideal Host**: Request population L at database tier T_3. The grey column contains the exact values, whereas the columns to the left and right contain lower and upper boundaries respectively.

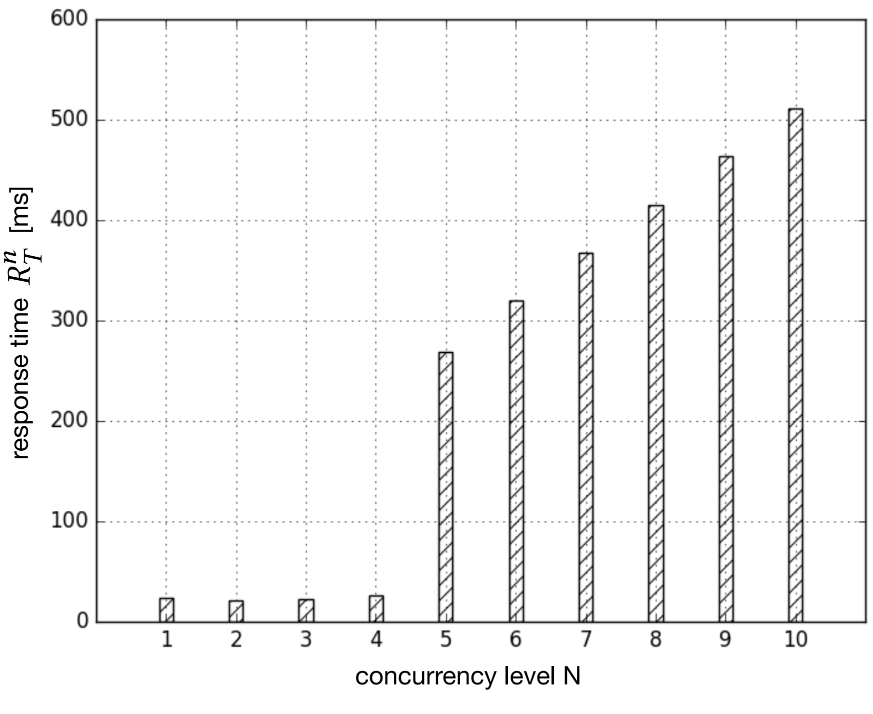


Figure 4.10: **EC2 Host**: Effect of overloading in the response time for a ePIC PID system instance hosted a Amazon EC2 instance. Beginning at a concurrency level $n = 5$, the response time is rapidly increasing.

4.5.6 Productive Host Measurements Analysis

In this section, we analyze the measurements from the productive host and compare these with the measurements from the ideal host.

ID	Alias	S[ms]	$\widehat{S}$[ms]
T_0	HTTP-SERVER	22	4
T_1	EPIC	51	24
T_2	HS	11	14
T_3	DB	17	15
response time R_T		101	57

Table 4.6: **Productive Host**: Service times for each tier of the old and improved system.

4.5.6.1 Service Time Improvement

For the productive ePIC PID system, we have reduced the PID creation time from at least $101ms$ to $57ms$.
Table 4.6 lists the service times of the old and improved ePIC PID system hosted on the productive system. The speedup factor for tier T_1 is $\alpha_{p1} = 51/24 \approx 2.13$. And the response time speedup factor is $\kappa_p = 101/57 \approx 1.77$, which is pretty close to the response time speedup from the ideal host (1.67).
However, by improving the service time of tier T_1, except for T_2, also the service times of all the remaining tiers have improved. This is particularly true for tier T_0. We assume that this is based on the reduced I/O operations, which first of all reduce the load on the attached storage medium (virtual hard disk). Additionally this leads to a reduction of the CPU context switches, while waiting for the I/O operation to complete. Thus, this has led the CPU to be used more efficiently for the ePIC PID system in the virtual (productive) host, leading to an overall improved performance.

4.5.6.2 Response Time Decomposition

Figure 4.11 illustrates the response time decomposition for the productive host. As for the ideal host in Figure 4.6, the processing times of the individual tiers are plotted with different colors on the bar plot. Again, the striped bars denote the improved ePIC PID system. In addition, the bars with crossed stripes denote the measurements, where the PIDs were created directly with the primary Handle server without involving the tiers T_0 and T_1. For this, the load-generator made use of the native Handle protocol to directly communicate with the primary Handle server.
Interestingly, in the productive host, the overall response time R_T^n first decreases until $n = 3$ for both versions of the ePIC PID system. And beginning at $n = 3$, the response times show the expected behavior of monotonic growth. We assume that is caused by a fluctuating load on the VMware-Cluster of GWDG during our measurement phase. The processing times R_i^3 are dedicatedly listed in Table 4.7.
In the unimproved ePIC PID system it is again tier T_1, which has the longest service time. For increasing concurrency level, however, its processing times are not significantly dominating the overall response times. Instead, the processing times of the tiers T_0, T_1 and T_3 almost equally contribute to the overall response times. We assume that this behavior is also based on the inefficient CPU usage due to high I/O operations in the unimproved ePIC system affecting the entire performance of the host itself.

According to the service times of the improved ePIC system, given by Table 4.6, the optimized tier T_1 would still be the dominating tier after the improvement effort. However, considering the processing times R_i^3, listed in Table 4.7, as in the ideal host, it is the database tier T_3, which is the new dominating tier. This is also underlined by the response times of the improved ePIC system for increasing load (cf. Figure 4.11).

ID	Alias	R^3[ms]	$\widehat{R}^3$[ms]
T_0	HTTP-SERVER	23	3
T_1	EPIC	40	13
T_2	HS	10	11
T_3	DB	15	17

Table 4.7: **Productive Host**: Processing times for each tier of the old and improved system for $n = 3$.

Moreover, for the unimproved ePIC system, the response times for increasing concurrency level have significantly lower magnitudes than in the ideal host (cf. Figure 4.11 and Figure 4.6). Our assumption for this is that due to the higher processing times at tier T_0 and T_3, the load on tier T_1 is damped in the productive host. In addition, due to the lower computing power at the productive host, we also assume that the load on the identifier generator component within tier T_1, is also reduced. Hence, likewise we assume that due to the higher computing power and faster hard disk at the ideal host, the load was more concentrated on the identifier generator component within tier T_1, which therefore has led its processing times to growth rapidly. This ultimately means, that in the productive host, modeling tier T_1 as a single queue was inappropriate (cf. Section 4.4.1).
Also for the improved ePIC PID systems, the response times of the ideal host, for increasing n, have a larger magnitude than in the productive host. This is again based on the load damping effect, as described for the unimproved system. However, as in the ideal host, also in the productive host, the processing times of the database tier T_3 are increasingly dominating the overall response time.
Our assumption of the load damping effect is also underlined by the response times of the direct PID creation at the primary Handle server. As we can also see from Figure 4.11, although for $n = 1$ it is significantly faster, for increasing concurrency level the overall response times are even longer than with involving the ePIC system. This is based on the fact that there is no load damping effect by subsequent tiers (T_0 and T_1). If we compare the response times of the improved ePIC system from the ideal host with the response times of the modified system (no T_0 and T_1) from the productive host, we can again see the effect of load damping: in the ideal host the response time starts at $42ms$ for $n = 1$ and increases until ≈ 250 for $n = 10$. Whereas in the modified system, it starts at only $25ms$ for $n = 1$, but also increases until ≈ 260 for $n = 10$.

4.5.6.3 Response Time Estimation

Figure 4.12 provides estimations for the response times based on Theorem 3. However, for the productive host, the estimations are not appropriate, since the actual response times, plotted as squares, are outside the respective estimated ranges. Only for the improved ePIC PID system, the estimations are relatively close to the estimated response time ranges. In both environments, productive and ideal, the database tier T_3 becomes the critical component after the improvement effort. Therefore, an increase of the load leads to a proportional rise of its processing time.

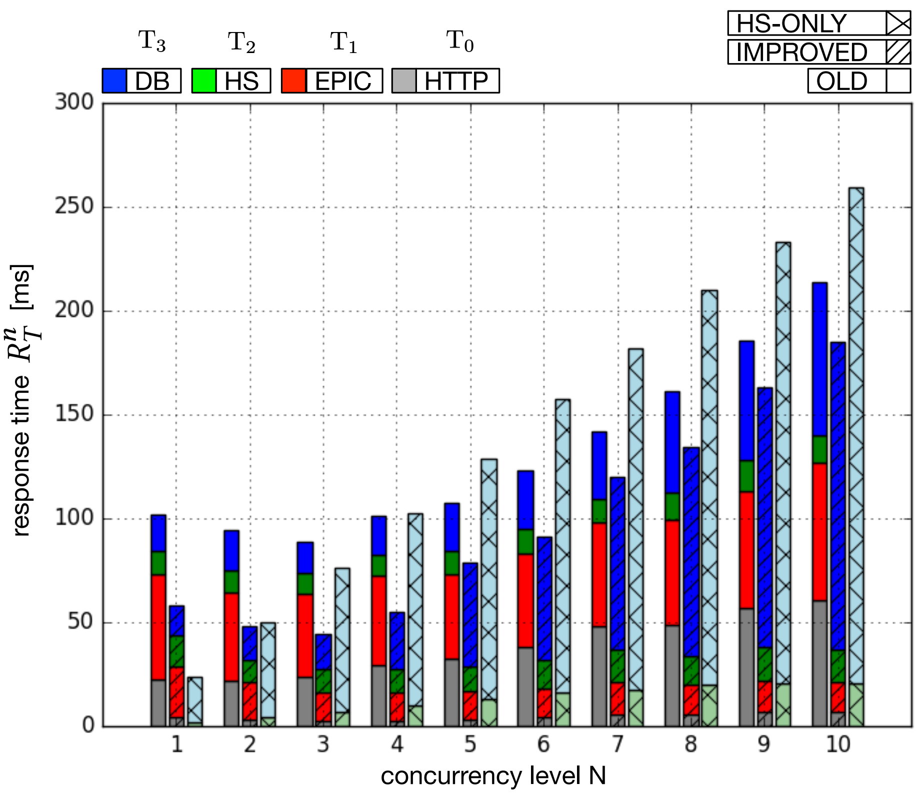


Figure 4.11: **Productive Host**: Response time decomposition for increasing concurrency level.

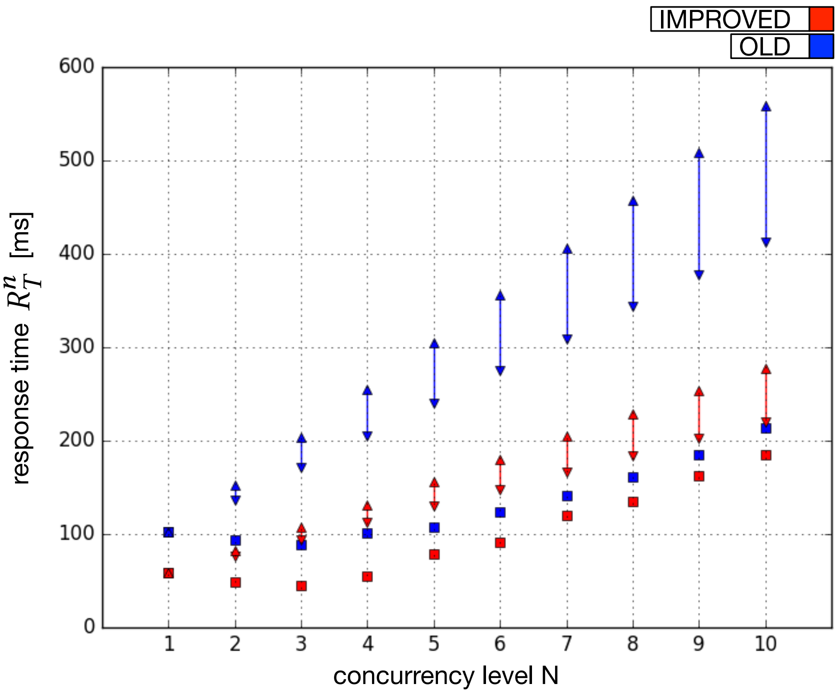


Figure 4.12: **Productive Host**: Response Time R_T for increasing request count including lower and upper boundaries $R_\triangledown$ and $R_\triangle$ provided by Theorem 3. The actual response times are plotted as squares, which are limited by an upper (↑) and lower (↓) arrow denoting the estimated boundaries ($R_\triangle$ and $R_\triangledown$).

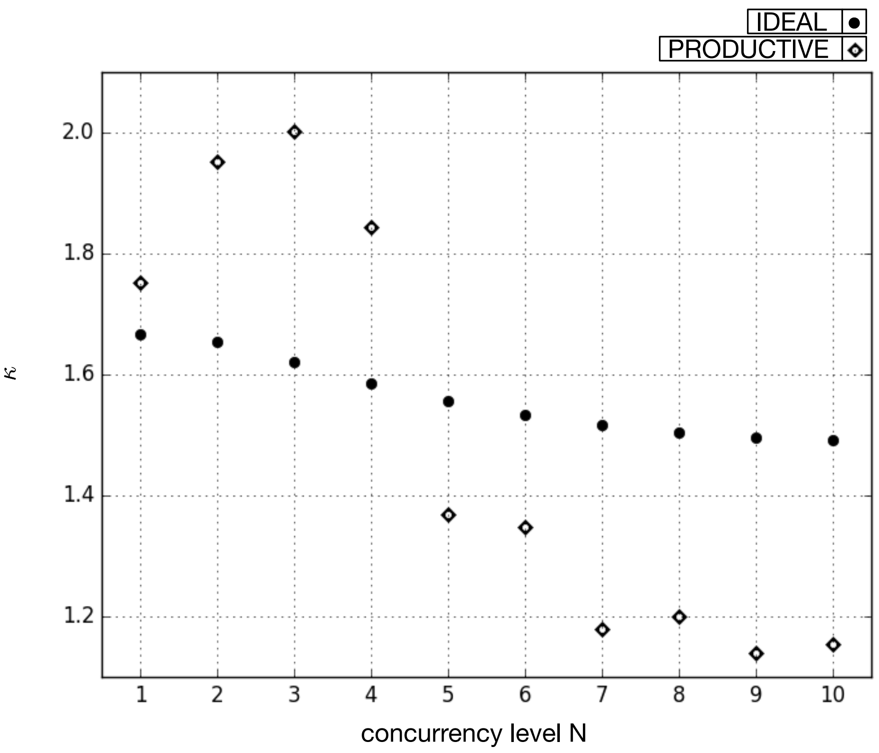


Figure 4.13: **Productive Host**: Response Time Speedup for increasing concurrency level

4.5.6.4 Course of the Response Time Speedup

A response time speedup comparison between the ideal and productive host is illustrated in Figure 4.13. As can be seen, for $n = 1$, the speedup factors κ are almost equal. However, while on the ideal host the response time speedup is decreasing, on the productive host it first increases approximately to the factor $\kappa_p^3 \approx 2.0$ and then it decreases rapidly to the factor of $\kappa_p^{10} \approx 1.1$. In principle, this behavior partially complies with the response time speedup behavior described in the proof of Lemma 2, which is that the speedup factor starts at κ_p^1 and moves towards a κ_p^∞, while κ_p^n can also have larger values than κ_p^1 and κ_p^∞. It should be noted that this does not mean that Lemma 2 and its deduced proof are disproved, but rather that in the productive host, the modeling of a tier as a single queue (remember the $T_i \equiv Q_i$ property) to be inaccurate.
In Lemma 2 we have deduced that $\kappa^\infty = S_{max}/\widehat{S}_{max}$, which would be in our case $\kappa_p^\infty = 51/24 \approx 2.1$ (cf. service times in Table 4.6) or $\kappa_p^\infty = 40/17 \approx 2.3$, if we would take the processing times R_i^3 from Table 4.7. With both results, this would mean that the response time speedup increases starting from $\kappa_p^1 \approx 1.77$ towards 2.1 or 2.3, which is, however, not the case for the actual response time speedup course as depicted in Figure 4.13.

Since the actual response time of the unimproved ePIC system for $n = 10$ is available, we can use this to estimate the expected response time speedup for high concurrency levels n. This approach was already described by equation (4.3.10). Since the processing time of the database tier T_3 is showing the expected behavior, with its service time $\widehat{S}_3 = 15$, we can calculate the expected upper boundary for the response time for $n = 10$, which is:

$$\widehat{R}^{10}_{\triangle} = S_3(n-1) + \widehat{R}^1_T = 135 + 57 = 192[ms]$$

Together with the response time $R^{10}_T \approx 213[ms]$ of the unimproved ePIC system we have:

$$\kappa^{10}_{\triangledown p} = \frac{213}{192} \approx 1.1 \leq \kappa^{10}_p = 1.15$$

which complies with the response time speedup course in Figure 4.13.
Hence, since the response time of the improved ePIC PID system is developing as expected, we made use of Theorem 3 to obtain an upper boundary for its response time at $n = 10$. In conjunction with the available response time R^{10}_T of the unimproved ePIC system, we could estimate the actual response time speedup factor for $n = 10$.

4.5.6.5 Population Count Distribution

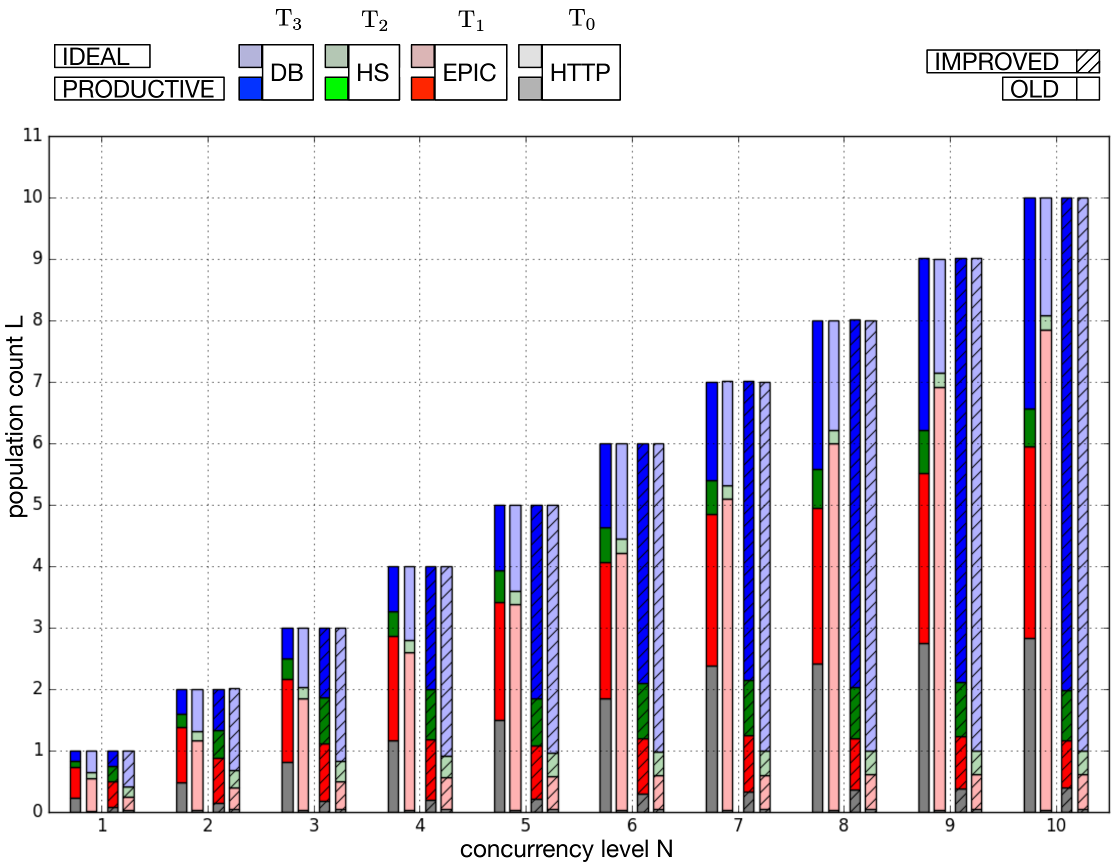


Figure 4.14: Request population comparison of four ePIC systems: (corresponding to the ordering in each 4-bar group) unimproved on productive host, unimproved on ideal host, improved on productive host, improved on ideal host . The request count for each tier is highlighted with a distinct color on the overall system's population bar.

Figure 4.14 shows the request population of the individual tiers of the unimproved and improved ePIC system. The bars with the pale colors denote the measurements from the ideal host (also: every second bar in the 4-bar group), where the bars with saturated colors denote the measurements

of the productive host.
As we can see, until $n = 7$ the population counts in the database tier T_3 of both unimproved ePIC systems are quite similar. This also means that the sum of the population counts of tier T_0 and T_1 grow as expected until $n = 7$, in both environments. However, starting at $n = 8$, the sum of their population counts begin to stagnate in the productive host, where in the ideal host, it is further growing. Instead, from $n = 8$ on, the population count of the database tier T_3 begins to growth significantly in the productive host.
We assume that this is again caused by the load damping effect before tier T_1 in the productive host. In the ideal host, such an effect is not occurring, therefore the load is concentrated at tier T_1. Furthermore, we assume that within tier T_1, the load in the ideal host is concentrated at the database access of the identifier generator component, which causes the processing time of T_1 to grow rapidly.
However, as it is the case for the processing times, also the population count distribution of both improved ePIC systems behave quite similar: The population count of the database tier T_3, as expected, is growing monotonously.

4.5.6.6 Weighting Factor Estimation

According to the service times in Table 4.6, we have only one dominating tier in both, the unimproved and improved system. This first of all means that we again have to apply Corollary 3. However, the core problem is that Corollary 3, as Corollary 2, are based on the monotonous growth of the weighting factors of the dominating tier in the unimproved system, which was only true in the ideal host. In the productive host, however, although T_1 is according to its service time S_1 the dominating tier, for increasing concurrency level, its processing times do not dominate the overall response times. This implies that the service time S_1 itself is again composed of multiple other internal component service times within tier T_1. As already stated, in the productive host, the modeling of tier T_1 as a single queue (the $T_i \equiv Q_i$ property) was inappropriate.
Ultimately, in order to be able to apply Corollary 2 or Corollary 3, we would have to retrieve the service times of the internal components within tier T_1.
Nevertheless, another option is to consider the insights gathered in the ideal environment as heuristic values which can be used as an orientation for the productive host. In our case this is even confirmed by the fact that both improved systems show a similar behavior, especially for the database tier T_3. In the ideal host we were able to predict the population count at tier T_3 to become at least $L_{\bigtriangledown DB} = 3.2$ (Section 4.5.5.6 and Table 4.5) for the productive concurrency level ($5 \leq n \leq 7$), which is also approximately true in the productive host (cf. Figure 4.14).

4.5.7 Summary

We have seen that our deduced formulas are applicable in the ideal host. For the productive host, however, they are only partially valid. In particular, our formulas for the weighting factor estimations were not applicable in the productive host. This was mainly caused by the inaccurate modeling of tier T_0 and especially of tier T_1 as a single queue. Due to less computation power, inefficient CPU utilization and slower (virtual) hard disk, within those tiers, the service times were actually composed of multiple other internal component service times.
However, for system optimization or advancement, such an ideal host is often involved. System developers usually do source code developments on the their local machines, which thus can be considered as ideal hosts. In addition, usually prior to the deployment on a particular testing system, the modifications are tested locally on such an ideal host. The investigation based on our formulas on the ideal host can finally reveal heuristic measures, which can be used for the

productive environment analysis.

Furthermore, for a high-performance PID system, the improved ePIC PID system is still too slow. The average response time an individual requestor would have to wait for a single PID registration is $\widehat{R}_T^{avg} \approx 100ms$ with the improved ePIC system at GWDG. For one million PIDs this would mean:

$$\begin{aligned} 1,000,000 \text{ PIDs} \times 100ms &\mathrel{\widehat{=}} 100,000,000ms \\ &\mathrel{\widehat{=}} 100,000s \\ &\mathrel{\widehat{=}} 28h \end{aligned}$$

Note that this calculation does not consider the network latency between a requestor and the PID system. Note also that recent research data repositories steadily issue PID administration requests.

Our investigation has also revealed that to achieve high-performance, the next improvement efforts have to address the tiers which follow the ePIC-API, the Handle server and the database tier. Although, in our evaluation the database tier proved to be the bottleneck, we assume that this is caused by an inappropriate handling of large amounts of administration requests within the primary Handle server. Therefore, the following chapter specifically focuses on the Handle protocol and on the administration procedures within the primary Handle server.

Chapter 5

High-Performance Persistent Identifier Management Protocol

As already indicated in the beginning of the previous chapter, PIDs are further gaining in importance for research data management, based on their ability of holding semantic information about the identified research dataset in addition to a sole locator. This has led PIDs to play an essential role for global research data exchange. The assignment of a PID to an individual research dataset can also be considered as a mapping from a local, community-specific into a globally interoperable representation of the research dataset. Hence, the concept of persistent identification enables community-specific research data repositories to be embedded into a virtual global research data network.

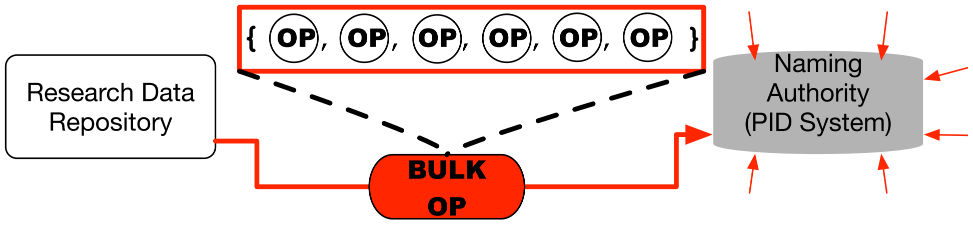


Figure 5.1: Focus in this chapter: Accelerating the administration of large groups of PID records by conceiving an efficient bulk administration operation (**BULK_OP**), which significantly improves the throughput of naming authorities.

The consequence is that these PID systems increasingly become an integral part of recent research data repositories as an additional component. However, since they are very much oriented on the DNS system, whose primary function is to act as a global lookup registry system for computer host addresses, rather than a global data management system, they are often not able to ensure a high performant management of large amounts of data. Due to this aspect, it is necessary for PID systems to be adapted to the workload of research data repositories to finally provide a high performant management of PID records.
Thus, in this chapter we focus on a high-performance persistent identifier management protocol, which is based on the Handle protocol.

The structure of this chapter is as follows: We first examine certain application areas of PIDs, which involve their advanced usage in addition to their basic function of locator abstraction. This is followed by a comprehensive comparison between the DNS and Handle System. The intention behind this comparison is to reveal the differences between both systems, especially in terms of

the respective workload they are subjected to. This comparison serves also the purpose of providing a better understanding of the Handle System and the need for a bulk administration operation, which is often lacking in recent PID systems, in particular, in systems based on the Handle System, such as the *Digital Object Identifier* (DOI) system or the ePIC system. Therefore, we finally propose to extend the Handle protocol with a bulk operation, for which we provide an appropriate specification and data model.
In contrast to the previous chapter, in this chapter, the focus is on the conception of bulk administration operation, which improves the throughput of naming authorities based on the Handle System (cf. Figure 5.1).

The base of our approach has been published in a conference paper at the 12th Networking, Architecture and Storage (NAS) conference 2017 in Shenzhen, China [104]. In this thesis, however, we provide a revised and extended version.

5.1 Towards a Global Virtual Research Data Network

The idea of depositing semantic information into PID records has led diverse research datasets to be assigned a PID. This in turn, is causing a steadily increasing load at PID registration agencies. The importance of semantic information in PID records is based on the huge variety of digital research data. With the increasing amount of research datasets new challenges arise for the handling of such a data diversity and volume. In order to mitigate research datasets from being solely stored in a specific research data repository and forgotten for future use cases, recent related research initiatives, such as the *Research Data Alliance*[13], focus on concepts, which can provide an active global exchange of research datasets among different scientific disciplines. To ensure this, a standardized representation of research datasets is required.
PID systems are globally connected systems forming a certain global research data registry. The idea, therefore, is to equip PID records with more complex, standardized semantic information. This is also important, because such huge volumes of research data are only processable by autonomous machines rather than by individual humans. Therefore, the need for a standardized semantic is once more enforced to be machine readable. In addition to processibility, also the interpretability of these research datasets has to be preserved during technological transition.
This pushes the PID systems to be the driving medium for global research data management and exchange. In other words, the concept of persistent identification moves from being a simple redirection mechanism towards a concept realizing a global virtual research data network with a common interface for managing and consuming research datasets.

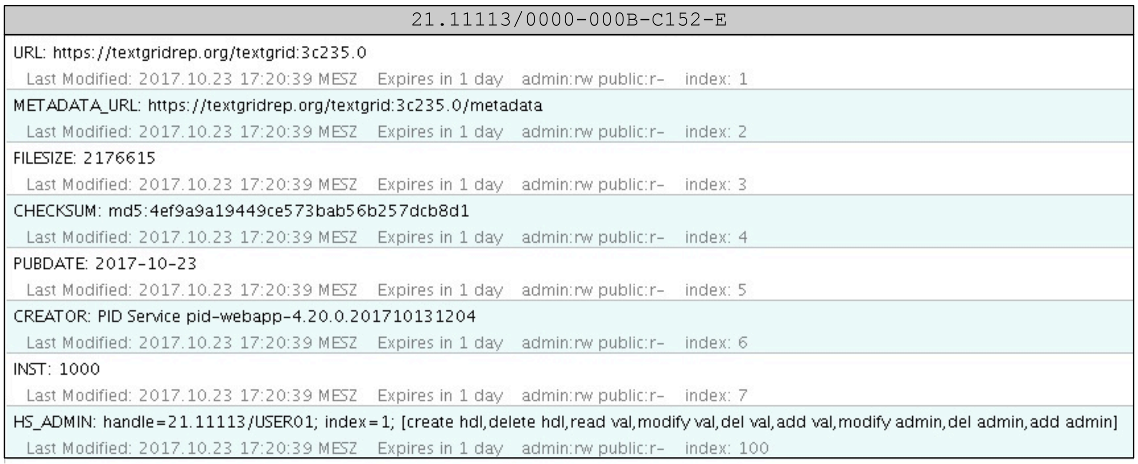
The following three subsections provide examples for such advanced application areas of PIDs.

5.1.1 Advanced PID Usage: Data Integrity Check

For the TextGrid repository PIDs are, in addition to identification and access, also used to ensure data integrity. This is realized by the *TextGridConsistencyCheck* [105] component in which specific object attributes (URL, file size, checksum) of a TextGrid object and the corresponding PID record are compared. In case of an inconsistency, the repository operators are notified, which then initiate appropriate steps to solve the inconsistency issue. For more detailed information about the working principle of this component we refer to the TextGrid documentation [106].
Note that the TextGrid repository constitutes an example for a group (B) repository (cf. Section3.2.3), which is not only relying on the locator abstraction function of PIDs, but also on

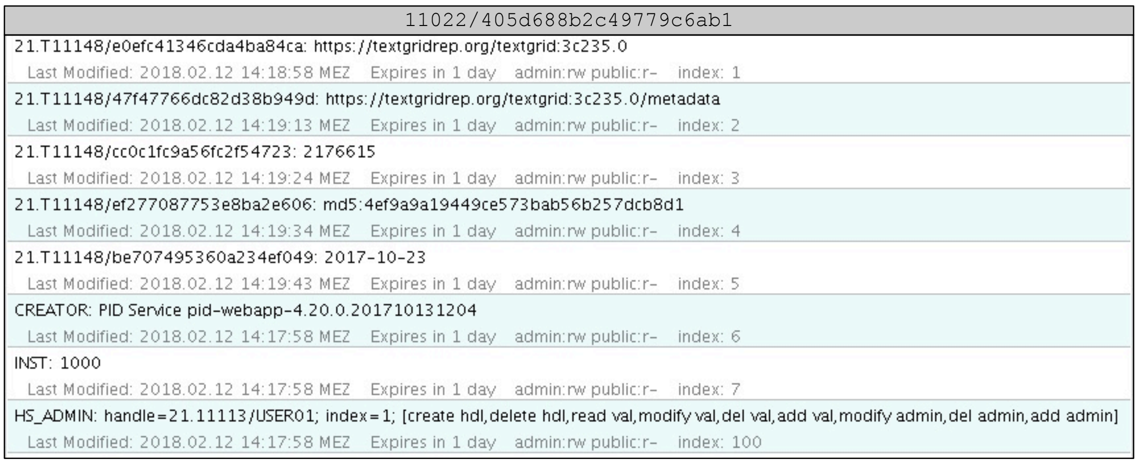
[13]https://www.rd-alliance.org

the individual metadata in the PID records. Therefore, our proposed *Research Data Silo Identifier* (DSID) approach is not appropriate for TextGrid.
Figure 5.2 illustrates such a PID record, which corresponds to a TextGrid object. This figure also provides an example for a PID record imposed with semantic information in addition to the sole locator (URL).



```
21.11113/0000-000B-C152-E
URL: https://textgridrep.org/textgrid:3c235.0
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 1
METADATA_URL: https://textgridrep.org/textgrid:3c235.0/metadata
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 2
FILESIZE: 2176615
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 3
CHECKSUM: md5:4ef9a9a19449ce573bab56b257dcb8d1
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 4
PUBDATE: 2017-10-23
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 5
CREATOR: PID Service pid-webapp-4.20.0.201710131204
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 6
INST: 1000
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 7
HS_ADMIN: handle=21.11113/USER01; index=1; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin]
  Last Modified: 2017.10.23 17:20:39 MESZ   Expires in 1 day   admin:rw public:r-   index: 100
```

Figure 5.2: Handle Record of a TextGrid object composed of eight Handle Values. The first six Handle Values are specific to the identified object, where the last two (INST, HS_ADMIN) are only used for administrative purposes within the ePIC system and Handle server.



```
11022/405d688b2c49779c6ab1
21.T11148/e0efc41346cda4ba84ca: https://textgridrep.org/textgrid:3c235.0
  Last Modified: 2018.02.12 14:18:58 MEZ   Expires in 1 day   admin:rw public:r-   index: 1
21.T11148/47f47766dc82d38b949d: https://textgridrep.org/textgrid:3c235.0/metadata
  Last Modified: 2018.02.12 14:19:13 MEZ   Expires in 1 day   admin:rw public:r-   index: 2
21.T11148/cc0c1fc9a56fc2f54723: 2176615
  Last Modified: 2018.02.12 14:19:24 MEZ   Expires in 1 day   admin:rw public:r-   index: 3
21.T11148/ef277087753e8ba2e606: md5:4ef9a9a19449ce573bab56b257dcb8d1
  Last Modified: 2018.02.12 14:19:34 MEZ   Expires in 1 day   admin:rw public:r-   index: 4
21.T11148/be707495360a234ef049: 2017-10-23
  Last Modified: 2018.02.12 14:19:43 MEZ   Expires in 1 day   admin:rw public:r-   index: 5
CREATOR: PID Service pid-webapp-4.20.0.201710131204
  Last Modified: 2018.02.12 14:17:58 MEZ   Expires in 1 day   admin:rw public:r-   index: 6
INST: 1000
  Last Modified: 2018.02.12 14:17:58 MEZ   Expires in 1 day   admin:rw public:r-   index: 7
HS_ADMIN: handle=21.11113/USER01; index=1; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin]
  Last Modified: 2018.02.12 14:17:58 MEZ   Expires in 1 day   admin:rw public:r-   index: 100
```

Figure 5.3: Handle Record of a TextGrid object composed of eight Handle Values. The first five Handle Values have a standardized type, where the type itself is again identified with a PID of the form (`21.T11148/XXX`).

5.1.2 Advanced PID Usage: Attribute Type Definition

Another example is the ePIC PID Information Type Registry[14] [43], which provides a registry for data type specification. Based on the these registries, machine actors can automatically de-

[14]http://dtr.pidconsortium.eu

termine the encoding of individual types used in a PID record and employ corresponding algorithms to consume the actual data byte stream. In Figure 5.3, we can see a copy of the TextGrid PID record, which is already depicted in Figure 5.2. Note that this copy is under the `11022`-prefix. The core difference between these PID records is that the data types in the `11022/405d688b2c49779c6ab1` PID record are also specified by PIDs instead of plain text type names, such as in the previous version shown in Figure 5.2. A machine actor can use these PIDs (`21.T11148/XXX`), which specify a type to determine a specification of the actual data byte stream stored in the data field.
As an example, Listing 5.1 shows the specification of the MD5 checksum, which can be retrieved by resolving the type PID `21.T11148/ef277087753e8ba2e606` in the Handle Value at $index = 4$ in Figure 5.3.

```
{
 "identifier" :"21.T11148/ef277087753e8ba2e606",
 "name" :"md5sum-string",
 "description" :"string of md5sum checksum ( context :data_integrity )",
 "standards" :[{
    "natureOfApplicability" :"depends",
    "name" :"1321",
    "issuer" :"RFC"
 }],
 "provenance" :{
    "contributors" :[{
      "identifiedUsing" :"Text",
      "name" :"Ulrich Schwardmann",
      "details" :"GWDG"
  }],
  "creationDate" :"2016-03-02T12:54:34.791Z",
  "lastModificationDate" :"2018-01-18T15:30:13.279Z"
 },
 "properties" :[{
   "dataType" :"string",
   "regexp" :"^((md|MD)5( |:|=))?( )*([0-9]|[a-f]){32}$",
   "flavour" :"ecma-262-RegExp"
 }],
 "validationSchema" :"{\"pattern\": \"^((md|MD)5( |:|=))?( )*([0-9]|[a-f])
    {32}$\", \"$schema\": \"http://json-schema.org/draft-04/schema#\", \"
    type\": \"string\", \"description\": \"md5sum-string@21.T11148/ef2770
    87753e8ba2e606\", \"definitions\": {}}"
}
```

Listing 5.1: JSON MD5 Checksum Specification

5.1.3 Advanced PID Usage: Complex Search Index

The imposing of semantic information into PID records offers yet another functionality: A search index composed of the PID system databases. We have already discussed a search function in conjunction with PIDs in Section 3.2.7, which was to register the search interface of the underlying research data repository. The actual query is then delegated from the PID system to the research

data repository.
Since PID records increasingly contain complex information, the PID systems itself can be extended to provide sophisticated search functions on top of their underlying databases.
Both, the ePIC PID system and the DOI system already provide a search function based on the underlying SQL databases. However, with the new database technologies, such as the special graph databases, it would be also possible to reveal relationships between different PID records. Moreover, the execution of special SQL queries can be rather slow. Therefore, GWDG is currently working on a semantic search functionality based on databases, which ensure high-performance searching. The approach of GWDG is to extend the mirror Handle server software to be able to support the attachment of a special graph database. This special mirror Handle server then replicates ordinary Handle Records from multiple primary Handle servers into the attached graph database. In contrast to the SQL database, attached to the primary Handle server, the graph database allows more complex queries, which are additionally processed much faster.

So far, we have discussed the increasing importance of PIDs by providing examples of their advanced usages. Before we address our actual approach of extending the Handle protocol, in the following section, we provide a comparison between the DNS and Handle System. This is important to have a better understanding of the Handle System and its protocol.

5.2 DNS and Handle System Comparison

As already indicated in the earlier chapters, PIDs are technically very well comparable with DNS domain names. The fundamental operations offered by a PID and DNS system are first a registration and secondly, a resolution operation. In both systems the registration is accomplished transactionally and hence, it poses an expensive operation.
A DNS registration agency is usually subjected to a large amount of registration requests by a large amount of individual registrants. These registrants are often still individual human users. In contrast to that, a PID registration agency is often subjected to requests by a relative small number of registrants, which however are characterized by a high individual request frequency. Furthermore, these registrants are mostly machine agents of research data repositories. For an individual registrant, where the frequency of registrations or updates is fairly low, an expensive writing operation is usually acceptable. However, for a research data repository with a high data ingest or modification frequency, this can often quickly cause a serious performance degradation in the whole data management workflow. Therefore, a high-performance PID management protocol is required to be able to efficiently manage such large amounts of research datasets.
Because of its versatility and sophisticated protocol, in this thesis we particularly concentrate on the Handle System. Thus, in this chapter we focus on improving the Handle protocol to realize such a high-performance persistent identifier management protocol. Since the Handle System constitutes the core component of various other PID systems, an improvement of the Handle protocol can potentially also lead to an improvement of these systems, such as the ePIC system. Moreover, in Section 4.5.7, we have already indicated that the underlying Handle server and its communication pattern with the attached database could pose a bottleneck in the ePIC PID system.

Both systems, DNS and Handle, can be considered as a hierarchical distributed database, for which the global operation is ensured by a specific communication protocol, the DNS and the Handle protocol respectively. However, there are also significant differences in these systems, which are revealed by a comparison in the following subsections.

5.2.1 Namespace

The DNS domain namespace is composed as a hierarchical tree structure consisting of labels which correspond to individual domains. A single domain name is uniquely addressed by a its corresponding label including all of its parent labels separated by a dot ”.”.

The Handle System namespace consists of two parts: a prefix and a suffix part. Both parts are separated by the slash character ”/”. The prefix part is comparable with a domain name since it consists of hierarchical labels separated by dots. The suffix in turn, is a locally unique name assigned to an individual research dataset. In conjunction with the prefix, the locally unique suffix becomes globally unique.

5.2.2 Architecture

Each domain is under the stewardship of exactly one zone. A zone in turn, manages and controls one or more domains. Each zone consists of one or more master nameservers and zero to many slave nameservers. There are myriads of globally distributed zones, which are ordered by complying with a certain hierarchy. The root zone of this hierarchy is represented by the zone with the label ”.”. It holds the information about each responsible *Top Level Domain* (TLD), which in turn form the second level in the hierarchy.
A resolution request is therefore delegated beginning from the root zone towards the responsible zone for an individual domain name.

The architecture of the Handle System has already been described in Section 3.3.1. Nevertheless, in the Handle System, the counterpart of the DNS root zone is the *Global Handle Registry* (GHR), where for a DNS nameserver it is an individual Handle server of a *Local Handle Service* (LHS). Likewise, a Handle-PID resolution request is delegated from the GHR towards the responsible LHS.

5.2.3 Data Model

In the DNS system, the data stored in the nameservers are denoted as Resource Records. In principle, an individual Resource Record can be reduced to a key-value pair containing a binding of a domain name and its corresponding IP-address.

Data in the Handle System is stored in Handle Records consisting of multiple Handle Values, which are comparable with Resource Records of the DNS system. Examples for a Handle Record have been already provided by Figure 5.2 and Figure 5.3. In addition to that, Figure 5.4 provides a general representation of a Handle Record and a Handle Value.

The core difference between a Handle Record and a Resource Record is that for Handle Records there is no restriction on a set of permissible data types as it is the case for Resource Records. This is a major advantage of the Handle System, which enables it to be positioned in other areas, especially in the area of the *Internet of Things* (IoT).

It should be noted that in Chapter 6, which dedicatedly deals with the resolution problem, the discussion also covers the data model of both systems. However, the focus in the current chapter is on the composition of Handle Records and Handle Values, whereas in Chapter 6 it is more on the transformation of Handle Records into Resource Records.

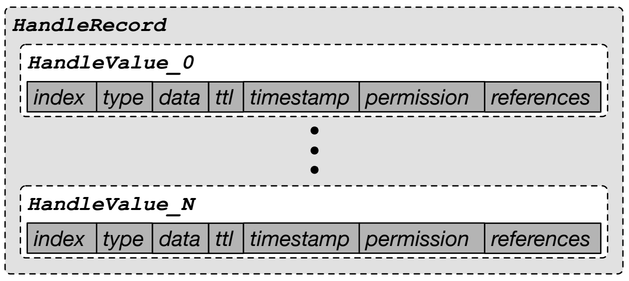


Figure 5.4: Generic Handle Record consisting of multiple Handle Values.

5.2.4 Protocol

Initially, the DNS protocol was designed for queries only. This is the reason for the Resource Records to be stored in a zone file, instead of a real database. However, with the increase of updating requests, the protocol has been extended by an update operation [107].
In the reference implementation, an update is transactionally appended to a log file, which is then used for propagating the update to the slave nameservers. Upon the restart of the master nameserver, the new zone file is constructed with the updated information stored in the log file. An update operation is always processed by the primary master nameserver.

The Handle protocol in contrast, offers a much richer set of operations with management operations forming an essential part of the protocol from the beginning on. Also for the Handle System, a management operation is always accomplished in a transactional procedure.
In the reference implementation of the Handle protocol, a single management operation involves two transactions: First, the update is transactionally appended to an internal database, which is used to provision the replication system. This is followed by an access into the attached main database, which ensures the updates to be persistently written to disk.
The Handle protocol also defines its own authentication and authorization mechanism. By using the Handle protocol an authenticated user can manage any Handle Record stored at any LHS in the world, provided that the user is also authorized. In contrast to that, in the DNS system there is no standardized common mechanism for authentication and authorization, which can considered as an indication that the DNS system has not been conceived for data management. This also underlines that the Handle System, although it appears to be a DNS imitation, is a globally distributed data management system rather than a data lookup registry such as the DNS system.

5.2.5 Workload

The workload of the DNS system is mainly characterized by resolution operations. On the other hand, the registration procedure is often still accomplished manually and hence, it usually involves a relative long delay until a domain name is updated and globally propagated. Moreover, as already mentioned, these large amounts of registration requests are often still issued by individual human users. Note that also in DNS there are machine agents issuing requests against DNS registration agencies. An example for that are the virtualization (Cloud) technologies, in which DNS entries are registered upon the instantiation of new virtual machines automatically. However, for an individual user, the instantiation time of a virtual machine is usually not highly critical since the actual focus is on the performance of the virtual machine itself, rather than on the management of the instance.

In the Handle System, the registration operation is increasingly issued by machine agents belonging to research data repositories. Usually, the requests of an individual research data repository can be grouped into batches sharing common information pattern corresponding to individual research datasets. However, each of these requests is usually processed in a dedicated, isolated transactional administration procedure. This often causes a serious performance degradation at the respective research data repository.
In addition, the communication pattern between an individual research data repository and a PID system, is usually characterized by a high frequency. This stems from the fact that PID record administration has become an integral part of the research data management workflow.

After contrasting the Handle System against the DNS system, in the following section, we provide an abstract description of our core idea to extend the Handle protocol with a bulk operation. The actual technical implementation of this idea then addressed in the subsequent section.

5.3 Handle Protocol Extension Approach

Based on the previous discussions, especially in Section 5.2.5, for the Handle protocol we propose an extension, which incorporates a bulk registration operation. This would enable large groups of research datasets sharing a common descriptive information content to be administered in a common transactional procedure.
Such a bulk registration is considered successful, when all PID records of the data batch have been processed successfully. In case of a failure, since the bulk operation is issued by an individual research data repository, only that individual registrant has to be notified with an error response message. This individual registrant can then react accordingly on the aborted registration operation, which for example could be another bulk registration operation attempt.
Generally, the acceleration by means of the bulking concept is well-known in the area of data management systems. However, our objective is primarily on the conception of a bulk registration operation for the Handle protocol. This can also be considered as a prerequisite to implement further bulking strategies into the internal procedures of the Handle server.
A bulk registration operation would greatly improve the ingest workflow of an individual research data repository, which is heavily communicating with a PID system.

Although administration requests belonging to the same batch are often issued iteratively, another approach could be to issue parallel administration requests from the research data repository against the PID system. However, at one point in the registration workflow of a PID system, all these concurrent requests have to be synchronized again to finally ensure transactional processing. If the synchronization mechanism is used to ensure that an, in terms of processing time, expensive station of the registration workflow is only populated by exactly one request at a time, which means to achieve an isolated processing in this station, then this could quickly cause the response time to growth rapidly due to the increasing queuing time. This effect was already analyzed in the previous Chapter 4.

5.4 Implementation

In this section, we consider the technical conception and implementation of a bulk registration operation into the Handle protocol. This endeavor covers two parts: First, the specification of such a bulk operation and secondly, the actual implementation into the Handle server. Therefore, we start with a basic insight into the Handle protocol and investigate the specification of the original registration operation, including the corresponding registration workflow in the reference imple-

mentation. Based on this analysis, we deduce a technical conception for a new bulk operation.
So far, we have only provided abstract explanations about the Handle protocol, in this section, however, we describe the Handle protocol more technically by addressing its composition and the corresponding encoding and decoding mechanism.

5.4.1 Handle Data Model

The registration process of an individual research dataset at the Handle System can be considered as a data model mapping, where the research dataset is mapped into a Handle Record. As previously described, each Handle Record is identified by a Handle (≡ Handle-PID), which is an immutable and globally unique identifier. Each Handle Record in turn, consists of a set of Handle Values and each Handle Value is composed of a set of fields (cf. Figure 5.4). The data field is used to hold the actual content of an individual Handle Value, whereas the type field is used to specify the syntax for the content. Since a Handle Record is composed of multiple Handle Values, the index field enables to specially address one of them.
In the usual case, all these Handle Values are then returned upon a resolution request for the corresponding Handle-PID. However, the index field also enables the selection of a certain set of Handle Values to be returned at the resolution.

5.4.2 Handle Protocol

In this section, we provide insight into the most relevant aspects of the current specification of the Handle protocol. For a more detailed and comprehensive explanation we refer to the official specification [95].

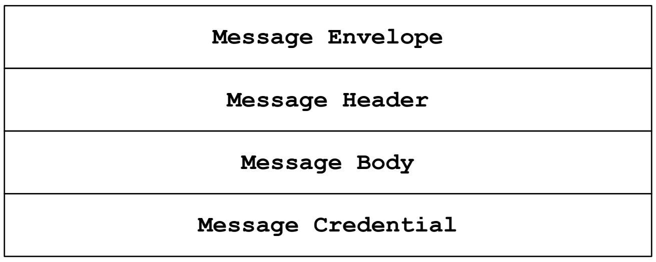


Figure 5.5: Handle Message Format

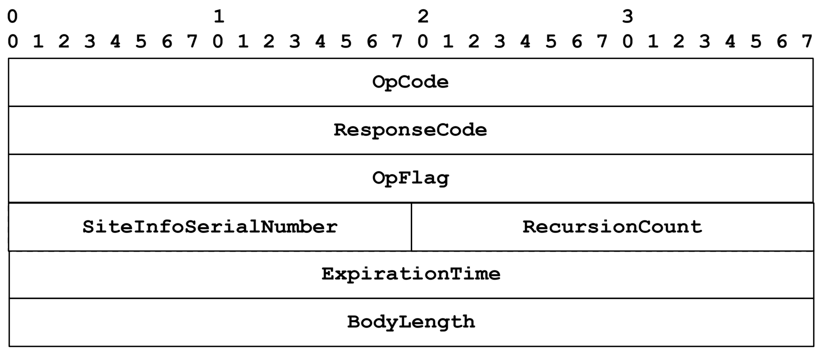


Figure 5.6: Handle Message Header Format

The Handle protocol defines its own messaging format used to enable the communication within the global Handle System overlay network. Each transmitted message consists of four sections, each fulfilling a dedicated function. Figure 5.5 provides an illustration of the logical composition of a single message exchanged in the Handle System.
Each exchanged message begins with a 20-byte sized section, the *Message Envelope*, whose primary function is to enable the message reassembly at the Handle server. The overall purpose of this section is limited to the message delivery.
The actual message begins with the *Message Header* with a fixed size of 24 bytes. Figure 5.6 provides a depiction of the Message Header. It holds the common data fields for all messages, with the operation code (`OpCode`) and the response code (`ResponseCode`) as the most important once.
The `OpCode`, a 4-byte unsigned integer, identifies the operation, which has to be executed by the Handle server. A subset of the operations offered by the original Handle protocol is listed in Table 5.1. In principle, the whole set of operations can be categorized into five groups:

OpCode	**Symbolic Name**	Remark
1	`OC_RESOLUTION`	Handle Resolution
100	`OC_CREATE_HANDLE`	Create new Handle
101	`OC_DELETE_HANDLE`	Delete existing Handle
102	`OC_ADD_VALUE`	Add Handle Value
103	`OC_REMOVE_VALUE`	Remove Handle Value
104	`OC_MODIFY_VALUE`	Modify Handle Value
1000	`OC_GET_NEXT_TXN_ID`	Get next transaction id
1001	`OC_RETRIEVE_TXN_LOG`	Retrieve current transaction id

Table 5.1: Subset of original Handle protocol operations. The operations highlighted with a grey background are used to administer Handle Records.

Handle Query: The Handle query operations enable a global resolution of any Handle-PID stored at the GHR or at any LHS. The corresponding resolution algorithm has already been discussed in Section 3.3.2.4. The result of a successful query operation is always a Handle Record.

Handle Management: The Handle management operations enable the administration of individual Handle Records. The currently offered management operations are shown in Table 5.1 with a grey background.

Authentication: Since only authorized users are permitted to perform managing operations, the Handle protocol defines its own mechanism for authentication. Only when an authenticated user also complies with the permission set of an individual Handle Record, the user is able to perform administration operations on that Handle Record.

Replication: To ensure the database of the mirror Handle servers to be consistent with the database of the primary Handle server, the Handle protocol additionally offers appropriate operations. The `OC_GET_NEXT_TXN_ID` operation (cf. Table 5.1) is one such an operation. It is used internally by the primary Handle server to increase a particular internal counter, which tracks the number of all so far performed administration operations. This number in turn, is retrieved by the mirror Handle server, which compares it with its corresponding internal counter. If its counter is smaller than the number retrieved from the primary, it initiates the replication of the missing updates. The counter retrieval from the mirror Handle server is accomplished with the OC_RETRIEVE_TXN_LOG operation. This operation is used by the mirror Handle server to periodically poll the primary Handle server.

Handle Service Administration: Another group of operations enables to administer the LHS itself. These operations include the adding or removing of prefixes, which are controlled within the LHS.

The ResponseCode, also a 4-byte unsigned integer, is used by a Handle server to inform the requestor about the status of its requested operation. Any ResponseCode other than zero is considered to be unsuccessful.
The BodyLength, a 4-byte unsigned integer, determines the number of bytes contained in the following *Message Body* section.
The Message Header is followed by the Message Body, which contains data specific to each operation. The data is encoded according to the OpCode and/or ResponseCode in the Message Header. Hence, the core of any message exchanged within the Handle protocol is composed by the Message Header and Message Body.
The last section of a Handle message is the *Message Credential*. It provides a mechanism for transport and security for any exchanged message within the Handle protocol.

Until this point, we have described the current (original) Handle protocol and its operations used to communicate with Handle servers. In the following, we first explain the current registration operation with its serialized representation and the corresponding processing procedure within the Handle server. This is important to understand the derivation our conception of the new bulk registration operation, which is finally proposed following the description of the current registration operation.

5.4.3 Current Registration Operation

As already indicated, in the current specification of the Handle protocol, new Handle Records are registered by means of the OC_CREATE_HANDLE operation. Note that this operation also enables to overwrite (update) existing Handle Records, which is achieved by a particular flag denoted as MSG_FLAG_OVRW. This flag is contained in the Message Envelope.

The actual content for this operation is contained in the message body, for which a particular serialization structure is defined, as shown in the following subsection. Note that the serialized representation is necessary to carry out the message transmission within the particular network.

5.4.3.1 Message Body Serialization

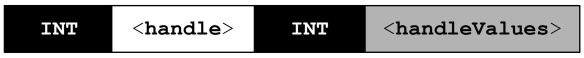


Figure 5.7: Serialized Message Body of already built-in OC_CREATE_HANDLE request.

The serialized representation of the corresponding message body is depicted in Figure 5.7. This representation describes the message body as a bit-stream representation used to specify the encoding/decoding procedure within the receiving end (client / Handle server). Each black INT field denotes a 4-byte unsigned integer, which are used to specify the number of elements to be read from the following field. For the <handle> field, it determines the number of characters,

INT <index>	INT <ts>	INT_1 <ttl_type>	INT <ttl>	INT_1 <perms>	INT	<type>	INT	<data>	INT <refs>

Figure 5.8: Already implemented Handle Value serialization.

which form the Handle-PID. Whereas for the `<handleValues>` field, it specifies the number of Handle Values, which are transmitted with the message body. Each individual Handle Value itself, is encoded according to the serialization scheme depicted in Figure 5.8.
In contrast to the black fields, the white `INT` fields, denote 4-byte unsigned integers, which contain values specific to the respective Handle Values fields. Note that a `INT_1` field occupies only one byte of memory.
The `<type>`, `<data>` and `<handle>` fields are each preceded by a 4-byte unsigned integer specifying the number of the contained elements composing the respective content.

With this structure, the message is sent to the primary Handle server.

5.4.3.2 Server-Side Registration Algorithm

When the primary Handle server receives a message submitted by a client, the message body is decoded back based on the information contained in the message header. In the case of a `OC_CREATE_HANDLE` operation, the message body is decoded according to the serialization scheme described in the previous subsection resulting a Handle-PID and a set of Handle Values. The message decoding procedure is then followed by the internal registration procedure.
As illustrated in Figure 5.9, the overall registration algorithm includes several steps. The thicker black lines represent the main path, which is executed for an ordinary successful Handle Record registration. The black boxes, ”Insert Transaction ID” and ”Write To Database”, represent the most important steps of the entire registration algorithm.

These black boxes are usually also the most expensive steps in the registration algorithm within the primary Handle server. In each of these steps, a transactional operation is performed. In step ”Insert Transaction ID”, certain information is written transactionally to an internal database used to ensure the replication system to be provisioned appropriately, including the increase of the previously explained counter tracking the number of all so far accomplished administration operations (cf. Section 5.4.2, `OC_GET_NEXT_TXN_ID`). In step ”Write To Database”, the actual Handle Values are finally persistently stored into the attached main database.
Therefore, these steps are often responsible for a significant performance loss of the overall registration algorithm, which is then ultimately propagated to the registrants.

However, also the green box, ”User Authorization”, can often be considered as a performance-relevant step. In this step the authorization process is executed. The internal algorithm of this process is dedicatedly depicted in Figure 5.10.
Depending on the implemented authorization scheme, it might cause a considerable overhead in the overall registration algorithm.
The authorization procedure in the Handle System always starts with the prefix Handle Record. The prefix Handle Record is queried in order to determine the authorized users, which are specified by `HS_ADMIN`-typed Handle Values. As an example, the prefix Handle-PID of the TextGrid repository is `0.NA/21.11113`, with the corresponding prefix Handle Record depicted in Figure 5.11. As can be seen (grey Handle Values), for the TextGrid prefix there are four groups of permitted

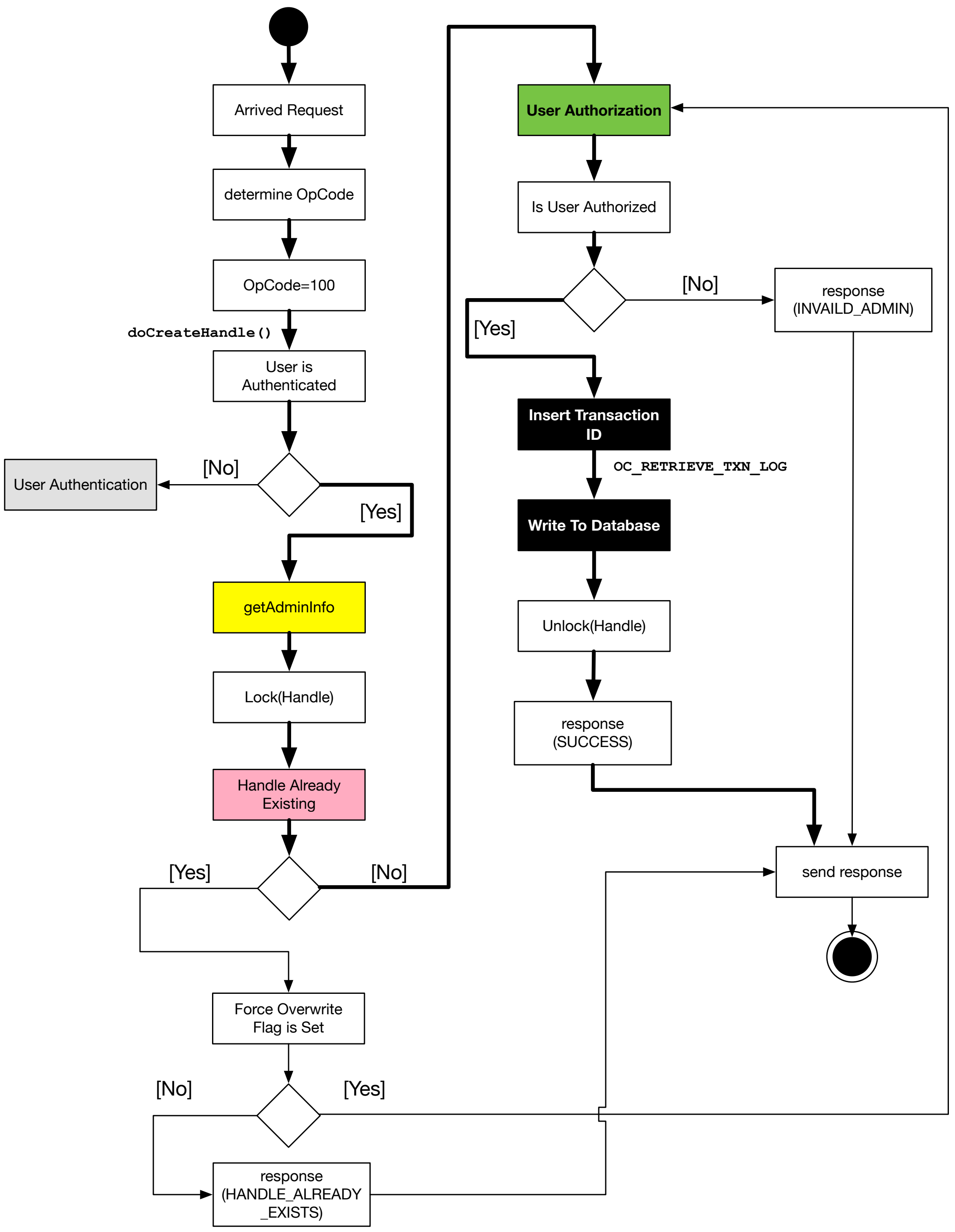


Figure 5.9: Algorithm for `OC_CREATE_HANDLE` operation, which is already implemented into Handle servers.

users. One of these is the group listed in the `21.11113/USERLIST`-Handle Record (Handle Value with index=102 in TextGrid prefix).

The list of permitted users is retrieved prior to the actual authorization process at the step represented by the yellow box, "getAdminInfo" (cf. Figure 5.9). This list is then used in the authorization process, green box, to determine whether the (authenticated) user is also a member of the set of authorized users.
However, exactly this process can cause a considerable overhead in the registration procedure, based on additional resolution queries as explained by the following two paragraphs:

Administrative User: Often, a Handle Record managing user is also an administrator of the Handle server itself. Such a user is directly authorized by a configuration file (config.dct) of the Handle server. The authorization workflow of this user concept is represented by the thicker black lines in Figure 5.10.

Non-Administrative User: Another concept is to authorize users based on individual Handle Records allowing to establish a hierarchy among the managing users. However, the disadvantage of this concept is that they entail additional resolution queries against the Handle server, which finally involve additional load on the attached main database. This is represented by the blue paths in Figure 5.10. As we have already seen in the previous chapter, an increased request population at a component can easily lead to an increased processing time of the component. This user concept, therefore, leads to more load at the database of the Handle server, which finally could cause a further increase of the overall response time.

Ultimately, when all the steps of the registration algorithm, including the authorization procedure, are accomplished, the Handle Record finally becomes registered or updated.
Inspired by the principles behind the original registration operation, in the following subsections, we finally provide our bulk operation extension for the Handle protocol.

5.4.4 New Bulk Registration Operation

We propose to extend the current Handle protocol by a new bulk registration operation with a symbolic name defined as:

`OC_CREATE_HANDLES_BULK`.

This new operation is identified by the operation code `OpCode`=900 in the message header and enables the registration of multiple Handle Records by one request.
The core difference of this operation is that all the contained Handle Records are processed in one transactional procedure. The request is considered successful when all Handle Records have been registered successfully. Otherwise, the request is considered to be failed.
As it is the case for the original `OC_CREATE_HANDLE` operation, also with the new bulk operation, existing Handle Records can be updated by means of the `MSG_FLAG_OVRW` flag.

The corresponding data model of our new operation is introduced in the next subsection.

5.4.4.1 New Bulk Data Structure

The message body of our new operation contains the data for the batch of the Handle Records to be registered. However, due to efficiency reasons, we have also extended the Handle protocol by an efficient and appropriate data structure, which means that it does not comply with the data

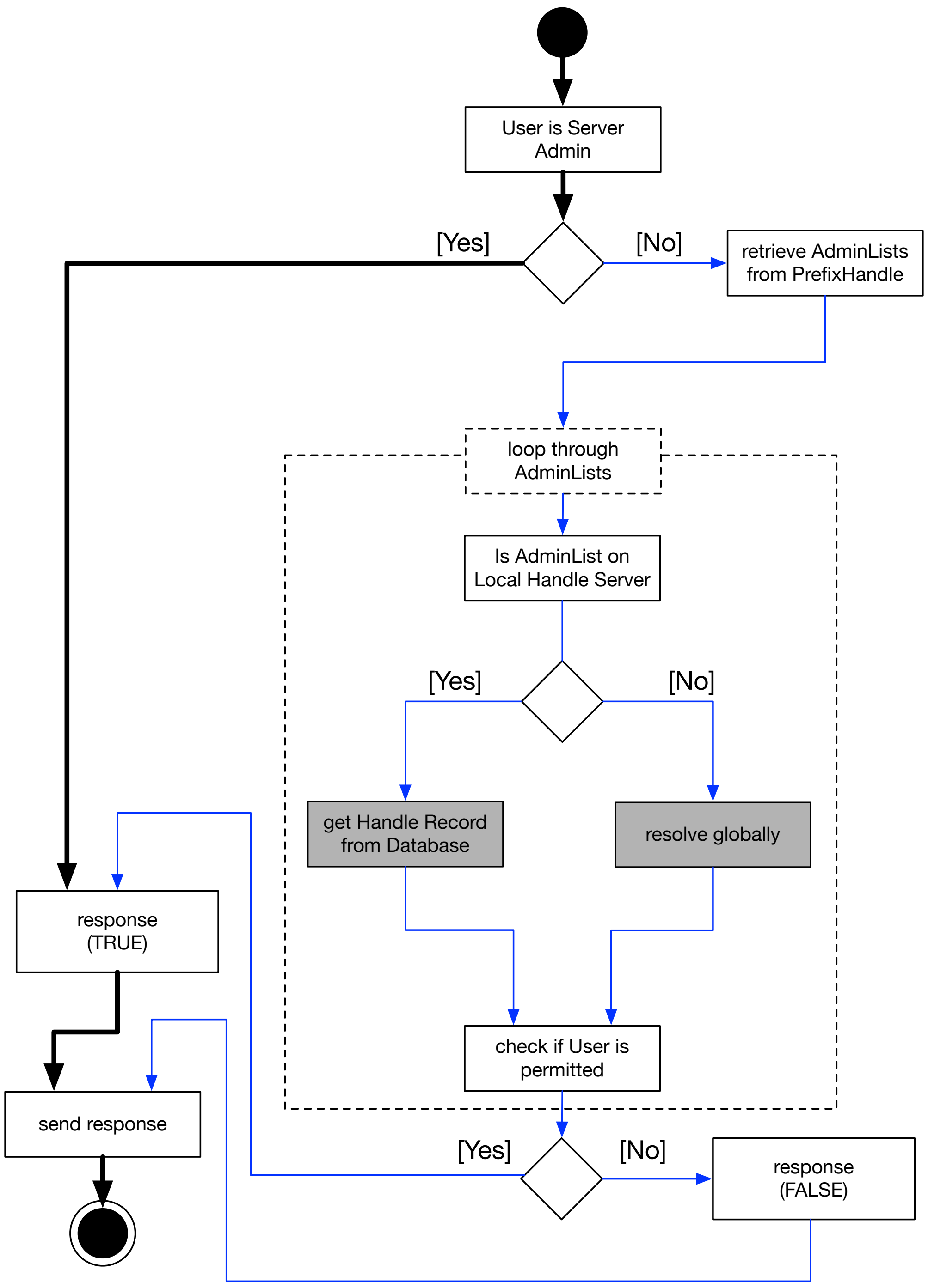
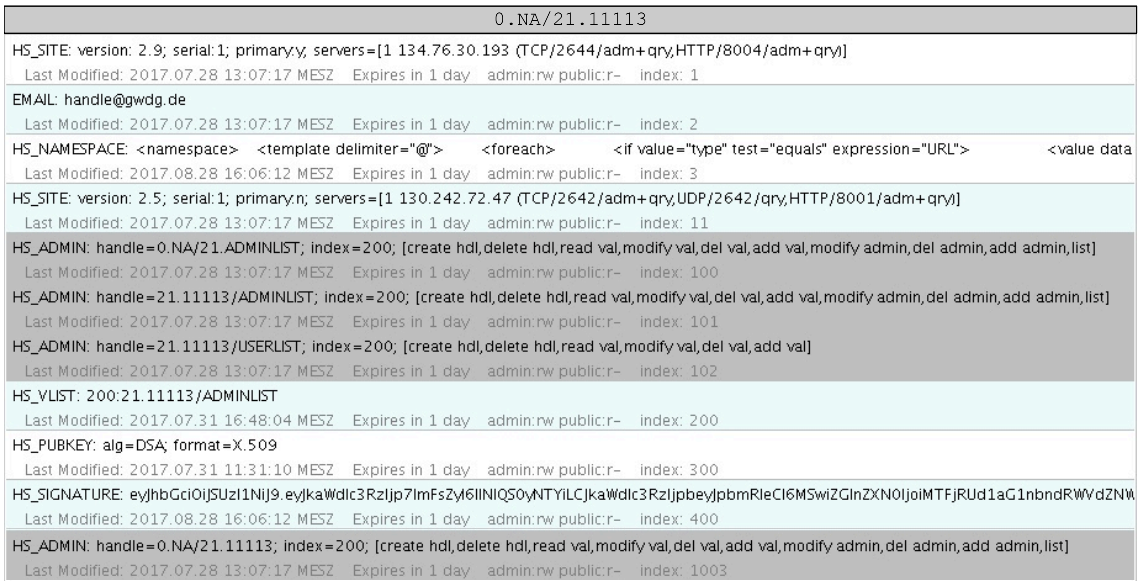


Figure 5.10: Original user authorization algorithm. The thicker black path represents the authorization process of an administrative user. The blue path represents the authorization process of a non-administrative user.



```
0.NA/21.11113
HS_SITE: version: 2.9; serial:1; primary:y; servers=[1 134.76.30.193 (TCP/2644/adm+qry,HTTP/8004/adm+qry)]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 1
EMAIL: handle@gwdg.de
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 2
HS_NAMESPACE: <namespace>   <template delimiter="@">      <foreach>         <if value="type" test="equals" expression="URL">          <value data
  Last Modified: 2017.08.28 16:06:12 MESZ   Expires in 1 day   admin:rw public:r-   index: 3
HS_SITE: version: 2.5; serial:1; primary:n; servers=[1 130.242.72.47 (TCP/2642/adm+qry,UDP/2642/qry,HTTP/8001/adm+qry)]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 11
HS_ADMIN: handle=0.NA/21.ADMINLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 100
HS_ADMIN: handle=21.11113/ADMINLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 101
HS_ADMIN: handle=21.11113/USERLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 102
HS_VLIST: 200:21.11113/ADMINLIST
  Last Modified: 2017.07.31 16:48:04 MESZ   Expires in 1 day   admin:rw public:r-   index: 200
HS_PUBKEY: alg=DSA; format=X.509
  Last Modified: 2017.07.31 11:31:10 MESZ   Expires in 1 day   admin:rw public:r-   index: 300
HS_SIGNATURE: eyJhbGciOiJSUzI1NiJ9.eyJkaWdlc3RzIjp7ImFsZyI6IlNIQS0yNTYiLCJkaWdlc3RzIjpbeyJpbmRleCI6MSwiZGlnZXN0IjoiMTFjRUd1aG1nbndRWVdZNW
  Last Modified: 2017.08.28 16:06:12 MESZ   Expires in 1 day   admin:rw public:r-   index: 400
HS_ADMIN: handle=0.NA/21.11113; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2017.07.28 13:07:17 MESZ   Expires in 1 day   admin:rw public:r-   index: 1003
```

Figure 5.11: Prefix Handle Record of `21.11113`. The grey Handle Values specify the users or user groups, which are permitted to perform administration requests against the primary Handle server.

serialization structure of the original registration operation depicted in Figure 5.7. This extension is based on the following fact:
Handle Records belonging to the same batch of administration requests issued by an individual registrant are often characterized by a common Handle Record structure.
Whereas the actual data contained in the data fields is usually unique within the batch of Handle Records, the overall set of used Handle Value types can often be reduced to a much smaller Handle Value type set. For example, the type set of a batch of Handle Records belonging to the TextGrid repository can be reduced to the following set of Handle Values types:
{URL, METADATA_URL, FILESIZE, CHECKSUM, PUPDATE, CREATOR, INST, HS_ADMIN} (cf. Figure 5.2).

To exploit such a structure, we have conceived the

`HandleRecordsBulk`

data structure. It is first of all a container for a Handle Records batch. Moreover, a `HandleRecordsBulk` is composed of a set of `RecordTypes` along with a set of `RecordDataSets`.
A `RecordType` includes all unique type identifiers, meaning that the number of `RecordTypes` to be less than or equal to the number of Handle Records contained in the batch. The structure of a `RecordType` is depicted in Figure 5.12. In principle, it is an ordinary Handle Value without the data field.
For TextGrid Handle Records, this would result in eight `RecordTypes`.
A `RecordDataSet` in turn, is composed of an unique suffix along with a set of `TypeDataPairs`. Hence, the number of `RecordDataSets` equals to the number of Handle Records in the overall `HandleRecordsBulk`.
Ultimately, a `TypeDataPair` is composed of a data field and a type index field. The type index corresponds to a certain `RecordType` contained in the set of `RecordTypes`.
An example for this is provided by Figure 5.13. It provides a representation for the TextGrid

Handle Record, already depicted in Figure 5.2, complying with our new data model.

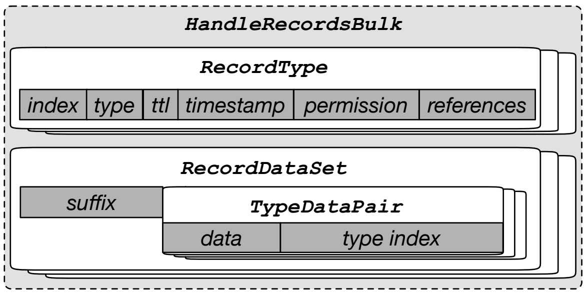


Figure 5.12: Our new `HandleRecordsBulk` Data Structure.

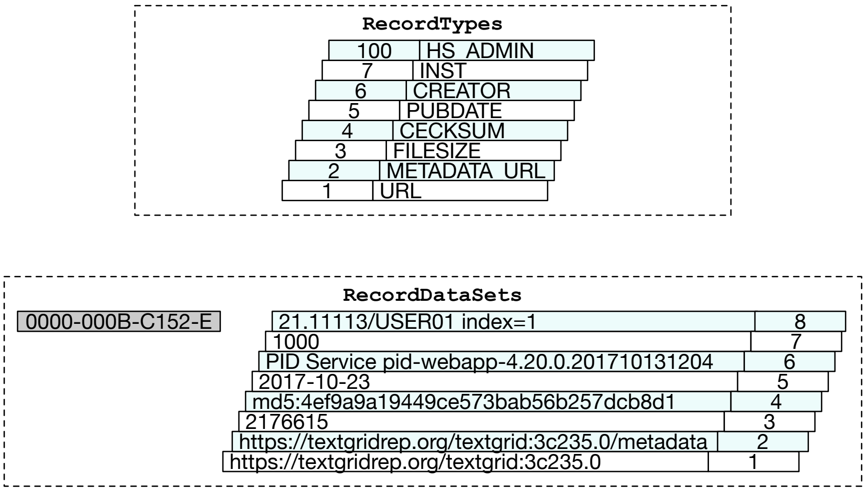

RecordTypes
100 HS_ADMIN
7 INST
6 CREATOR
5 PUBDATE
4 CECKSUM
3 FILESIZE
2 METADATA_URL
1 URL
RecordDataSets
0000-000B-C152-E
21.11113/USER01 index=1 8
1000 7
PID Service pid-webapp-4.20.0.201710131204 6
2017-10-23 5
md5:4ef9a9a19449ce573bab56b257dcb8d1 4
2176615 3
https://textgridrep.org/textgrid:3c235.0/metadata 2
https://textgridrep.org/textgrid:3c235.0 1

Figure 5.13: `RecordTypes` and `RecordDataSets` for TextGrid Handle Record from Figure 5.2.

As for the original administration operation, also for this new bulk operation, there is a particular structure definition for the message body transmission required. This is tackled in the following subsection.

5.4.4.2 Message Body Serialization

Figure 5.14 illustrates the serialization, including the prefix, of the overall message body of our new bulk operation. As already indicated in Section 5.4.3.1, the serialized representation determines the encoding/decoding algorithm within the receiving end (client / Handle server) upon message transmission/reception.
At the client-side, after the instantiation of a `HandleRecordsBulk` data object, it is filled with Handle Records by means of the following method:

```
add(String suffix, HandleValue [] values)
```

The prefix is specified at the `HandleRecordsBulk` data object instantiation.
When this batch Handle Records container is finally provisioned with all the to be administered Handle Records, it becomes automatically serialized while initiating the sending to the Handle server. Hence, an individual developer has basically not to deal with the internal `RecordTypes`

and `RecordDataSets`.
The serialization algorithm of these two internal data structures is in principle comparable with the algorithm for an ordinary Handle Value, which was already described by Figure 5.8.

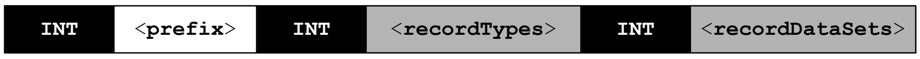


Figure 5.14: Serialized Message Body of our new `OC_CREATE_HANDLES_BULK` request.

Finally, the actual `HandleRecordsBulk` data object is sent with this representation to the primary Handle server.

5.4.4.3 Server-Side Registration Algorithm

After the retrieval of such a message by the primary Handle server, the message is again decoded back into a `HandleRecordsBulk` data object. This step is then followed by the corresponding registration algorithm, which is shown in Figure 5.15.
The main principle of this new registration algorithm is that the second transactional step, ”Write To Database (Bulk)”, is only initiated when the first transactional step, ”Insert Transaction (Bulk-Size)”, has been successfully accomplished for all Handle Records. At each of these transactional steps, each Handle Record of the `HandleRecordsBulk` is processed consecutively. After the successful processing of the last Handle Record, the overall transactional step is committed. Note that this new operation also requires the introduction of another operation used to increase the number of the internal transaction counter. Instead of incrementing the transaction counter one by one, with the new bulk registration operation, it is directly increased by the batch size. Therefore, we have additionally implemented the `OC_GET_NEXT_BULK_TXN_ID` operation with a `OpCode`=1009, which is executed within the first transactional step, ”Insert Transaction IDs (BulkSize)”.
The result of this new procedure is a reduced synchronization overhead between the processing of individual Handle Records.
Further research is required to evaluate various other possible server-side bulk registration algorithms with the goal to determine a more efficient internal bulk processing algorithm. One possible approach would be to implement a concurrent processing at each of these transactional steps. This is, however, beyond the scope of this thesis.

Finally, when all these steps of the new registration algorithm has been successfully accomplished, the administration of all Handle Records of the `HandleRecordsBulk` is finished. The operation is then accomplished with a single message sent to the client containing a success message.

5.5 Evaluation

In this section, we provide an evaluation of our extended version of the Handle protocol (current version: 2.10). Note that our extension [108] only includes modified or additional source files. To generate a deployable software package, these source files have to be compiled with the complete Handle software package provided by CNRI[15].

[15]http://www.handle.net/download_hnr.html

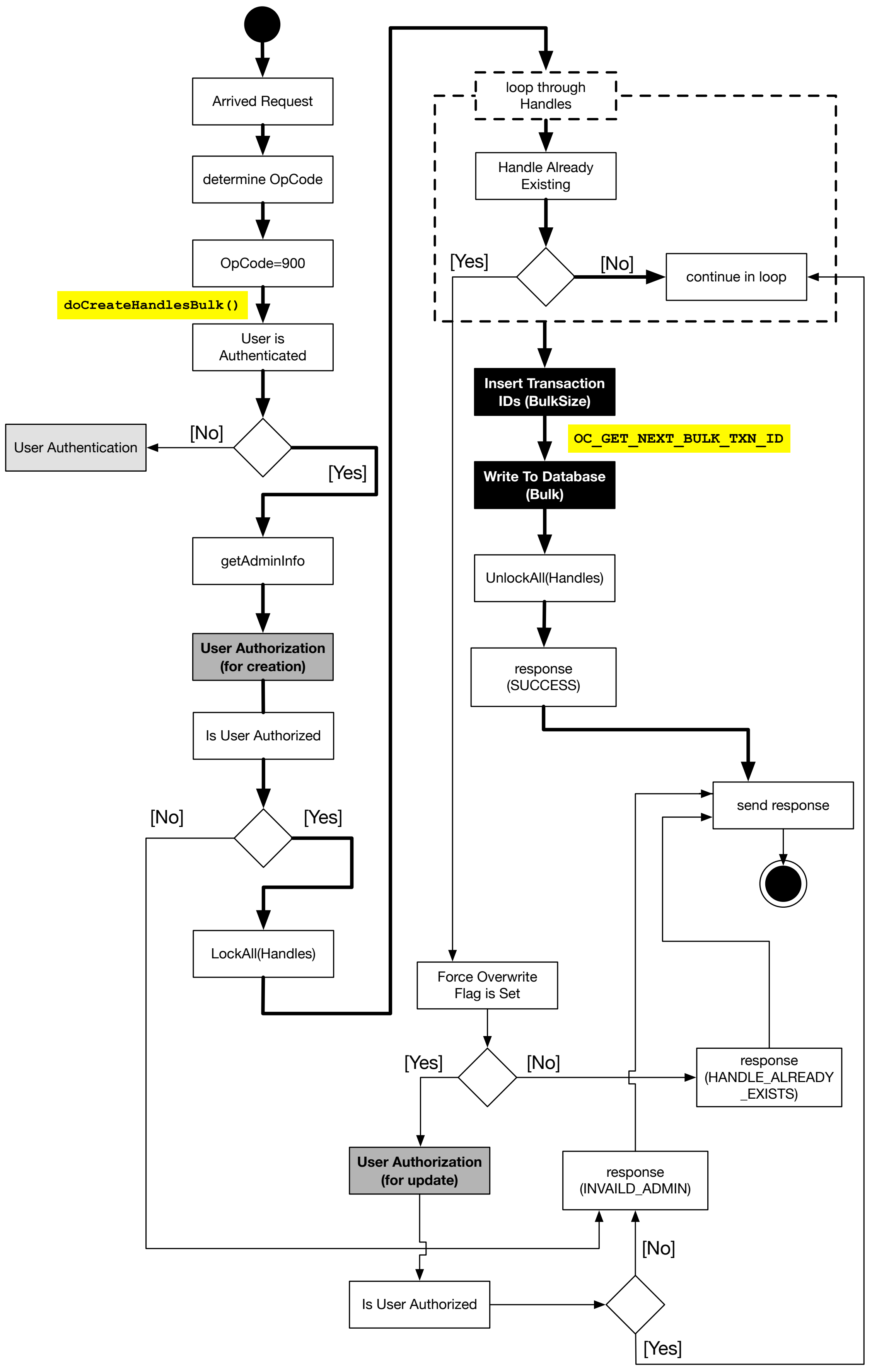


Figure 5.15: Algorithm for our new `OC_CREATE_HANDLES_BULK` operation, which has been implemented into our Handle protocol extension.

After describing our evaluation setup, we provide an analysis of the measurements, which were gathered from three different evaluation approaches based on three different workload types.

5.5.1 Evaluation Setup

For the evaluation, we established an experimental setup on the Amazon EC2 cloud as depicted in Figure 5.16. The setup was comprised of four ”c3.xlarge” instances. Among them, the primary Handle server, located in Frankfurt (FRA), constitutes the core component of the experimental setup.
The current Handle server implementation supports the attachment of two different databases: First, a Berkeley Database (BDB), which is an in-process key-value database. Secondly, a MySQL database, similar to the one used in the previous chapter. However, in contrast to the previous chapter, in this evaluation, the MySQL database server was installed on the same host (localhost) as the primary Handle server. Note that the storage engine of our MySQL database was again *InnoDB*, ensuring a transactional processing.
Another aspect with the *localhost* MySQL database is an accelerated insert time in comparison to an instance from the (non-localhost) GWDG MySQL Appliance cluster. This is mainly based on the reduced network latency between the Handle server and the database. But also due to a lower load on the hosting environment of the database itself. As part of the core GWDG infrastructure, the GWDG MySQL Appliance cluster is usually subjected to high loads. This finally means that the usage of the *localhost* database can be considered as an improvement of the database tier, tier T_3 of the setup from the previous chapter.

Finally, in our current evaluation procedure we made use of two databases, a BDB and a localhost MySQL database, to examine the differences in the administration performance.

To stress the primary Handle server with appropriate load, we placed two load-generator instances. One of the load-generators was, such as the primary Handle server, located in Frankfurt (FRA), where the other one in Ireland (IRE). The purpose of the latter was to determine the impact of the network latency.
Note that the source code of our load-generator can be found at [109].
Furthermore, a research data repository is usually connected to a PID registration agency, which is located in its relative geographic vicinity. Therefore, this setup can be considered to reflect the situation for repositories hosted in Europe.

The last instance, which is also located in Ireland, is a mirror Handle server to replicate the primary's database.

All these instances were equipped with our extended Handle protocol implementation, which is based on the reference implementation of the Handle server software with version 8.1.0.

5.5.2 Workload Generation

The load-generator was implemented to produce three kinds of workload types:

(WL-A) Increasing Concurrency Workload: This workload type was used to investigate the response time decomposition of the Handle server for the original `OC_CREATE_HANDLE` registration operation. By increasing the amount of registration requests, which are concurrently processed in the primary Handle server, this workload type enables the analysis of the impact of each step of the registration algorithm depicted in Figure 5.9.

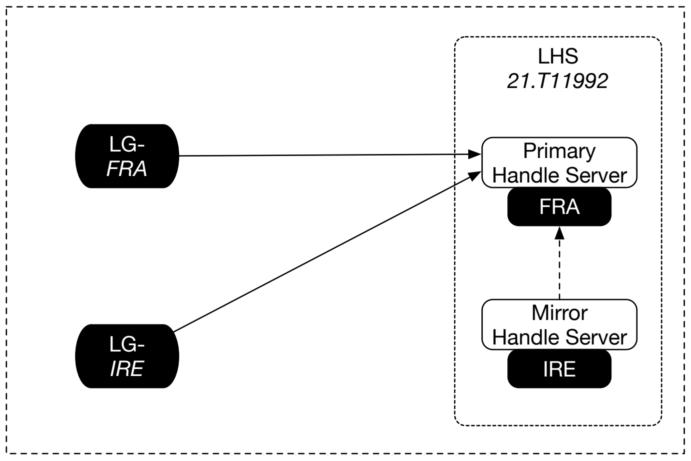


Figure 5.16: Amazon EC2 Evaluation Environment

In addition to the two different databases, MySQL and Berkeley DB, we also used two different authorized users. The first user was an administrative user of the primary Handle server, whereas the second was a non-administrative user. By means of these two users, it is possible to determine the performance difference for the different authorization mechanisms already described in Section 5.4.3.2.
For this workload, only the behavior of the primary Handle server was of interest. Therefore, only the load-generator located in Frankfurt was used.
Finally, we also provide insight into the corresponding throughput measures to reveal the maximum concurrency level supported by the Handle server.

(WL-B) Isolated Workload: This type of workload was used to examine the performance of the registration operations without being interfered by any other concurrent operation. In this workload, dataset batches of three different sizes (1000, 5000, 10.000) were generated and registered by means of three different approaches:
In the first approach, each dataset was registered iteratively with the original `OC_CREATE_HANDLE` operation. In the second approach, each dataset batch was divided into five sub-batches of equal size. All these sub-batches were then processed concurrently, where each dataset in a sub-batch was the registered in the same manner as in the first approach (iteratively). The number of five sub-batches is based on the throughput analysis of workload type (WL-A). As we will see in Section 5.5.3.1, it turned out that the maximum throughput of the Handle server is achieved for the concurrency level of about $n = 5$.
In the third approach, the whole batch was registered by means of the new `OC_CREATE_HANDLES_BULK` operation.
For each dataset batch, this procedure was repeated five times. Moreover, in this workload type, the performance of the registration from client point of view was of major interest. Therefore, both load-generators were involved.

(WL-C) Productive Workload: Finally, in this workload type, the primary Handle server was subjected to a request mixture based on the productive workload. This is especially useful to investigate the impact of the new `OC_CREATE_HANDLES_BULK` operation onto the system's overall stability. Therefore, only the load-generator located in Frankfurt was used.
Note that for generating productive workload, we analyzed the workload of the PID systems offered by GWDG.

It should be noted, that in each workload type, the primary Handle server was stressed with real world datasets extracted from the databases of GWDG PID systems.

5.5.3 Measurements

In the following, we investigate the measurements gathered in each of three aforementioned workload type runs.

5.5.3.1 Workload Type WL-A

The response time decomposition of the primary Handle server for the `OC_CREATE_HANDLE` operation is depicted in Figure 5.17. The bars with a red hue denote the measurements with the administrative user "`0.NA/21.T11992:index=300`". In contrast to that, the bars with a yellow/orange hue correspond to the measurements with the non-administrative user "`21.T11992/USER01:index=1`", which belongs to the group of authorized users specified by the Handle "`21.T11992/USERLIST:index=200`".
Hence, to authorize the user "`21.T11992/USER01:index=1`", it is necessary to retrieve the Handle "`21.T11992/USERLIST`" stored in the database of the primary Handle server.
As can be seen in Figure 5.19, for our prefix used for the evaluation, there are four groups of authorized users (`HS_ADMIN`-typed Handle Values). Thus, the authorization algorithm starts with "`0.NA/21.ADMINLIST`" and since the user "`21.T11992/USER01:index=1`" is member of "`21.T11992/USERLIST`", it terminates at this group Handle.

From Figure 5.17, we can see that the response time of the Handle server is increasing with the concurrency level. We can also note that, except the response times in the orange colored bars, all remaining response times are dominated by the processing times of the replication system and the database. Remember that these steps are highlighted as black boxes in the registration algorithm illustrated in Figure 5.9.
The response times with the non-administrative user "`21.T11992/USER01:index=1`" are always longer than with the administrative user "`0.NA/21.T11992:index=300`". However, with the BDB, the difference is only minimal. The response times with the BDB are also always shorter than with the MySQL database, which is based on the fact that it is running under exactly the same process as the primary Handle server, wherefore the BDB is also called a "in-process" database.

In contrast to the measurements from the previous chapter, in this measurements, it is the transactional step to provision the replication system of the primary Handle server which increasingly dominates, again except the orange bars, the response time. In the previous chapter, it was the database.
We assume that this is mainly based on the *localhost* MySQL database. The accelerated processing time of the *localhost* MySQL database leads to an increased load concentration in the Handle server itself, causing its internal processing time to increase rapidly.

However, the response times of the registration requests submitted with the non-administrative user to the Handle server attached to the MySQL database (orange bars) are increasingly dominated by the processing times of the authorization procedure. Note that the contribution of the authorization procedure is marked as double hatches. Also the step to check whether the to be registered Handle-PID is already existing in the database (area with black stars), becomes increasingly significant with the non-administrative user in the MySQL database attached Handle server. This step corresponds to the box with the red hue in Figure 5.9. For the BDB measurements, the contribution of this step is barely recognizable on the respective bars, which is also true for the

MySQL measurements with the administrative user.
A comparison of the response times with the MySQL database (administrative and non-administrative), reveals that with the administrative user (red bars), the processing time of authorization procedure becomes negligible, whereas the processing times of the replication system become dominating. With the non-administrative user (orange bars), the processing times of the replication system are significantly shorter than with the administrative user. This is based on the load damping effect of the authorization procedure with non-administrative user, which leads the load to be shifted from the replication system into the authorization procedure.

Finally, we have seen that in particular the expensive transactional steps (replication system and database), lead to a serious performance degradation of the original registration operation `OC_CREATE_HANDLE`. With the non-administrative user, the response time of the MySQL database attached Handle server (orange bars) is also mainly composed of the remaining database involving steps (existence check, authorization) in addition to the contribution of the replication system and the writing into the database.

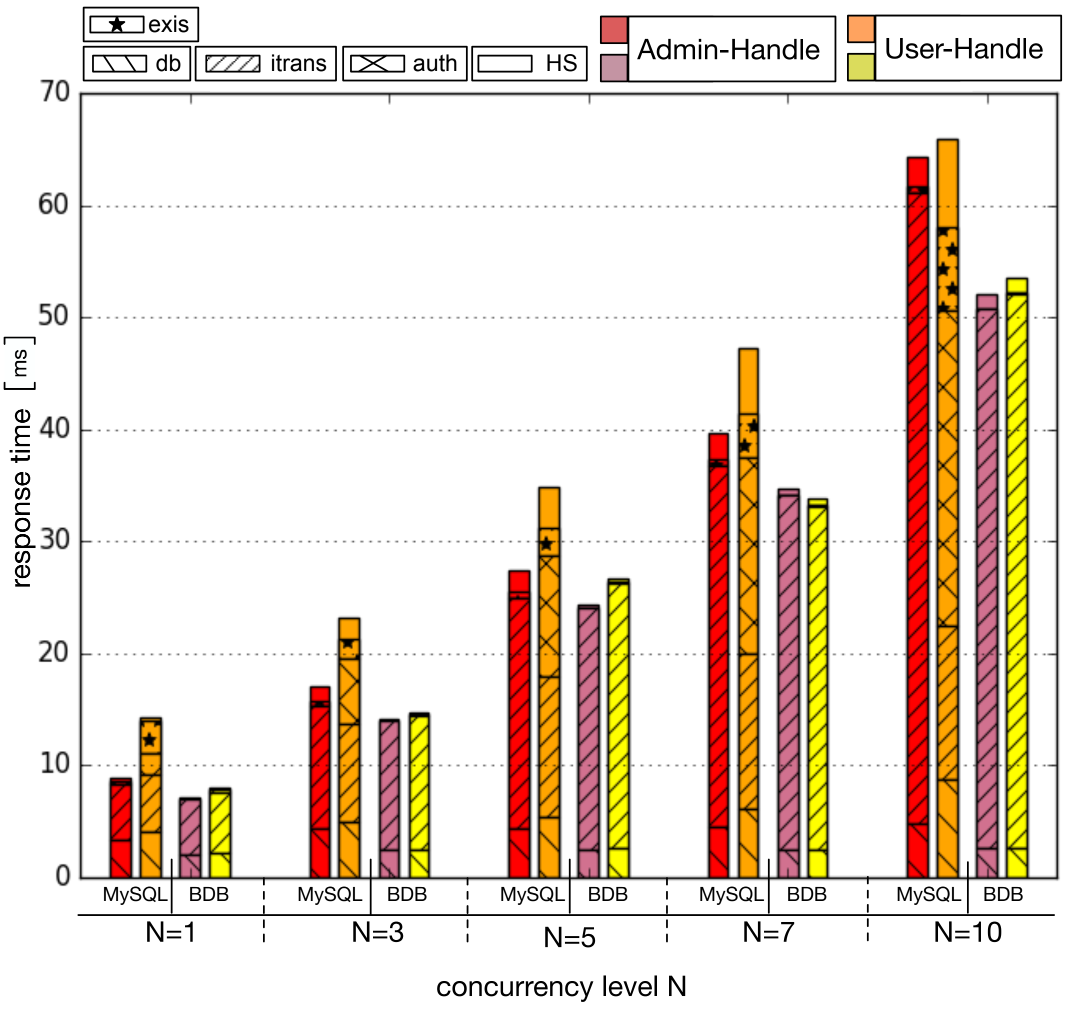


Figure 5.17: Response time decomposition for increasing concurrency level for the original `OC_CREATE_HANDLE` operation. The bars are composed of the following contributions: database (*db*), replication system (*itrans*), authorization (*auth*), existence check (*exis*), Handle server overall response time (*HS*).

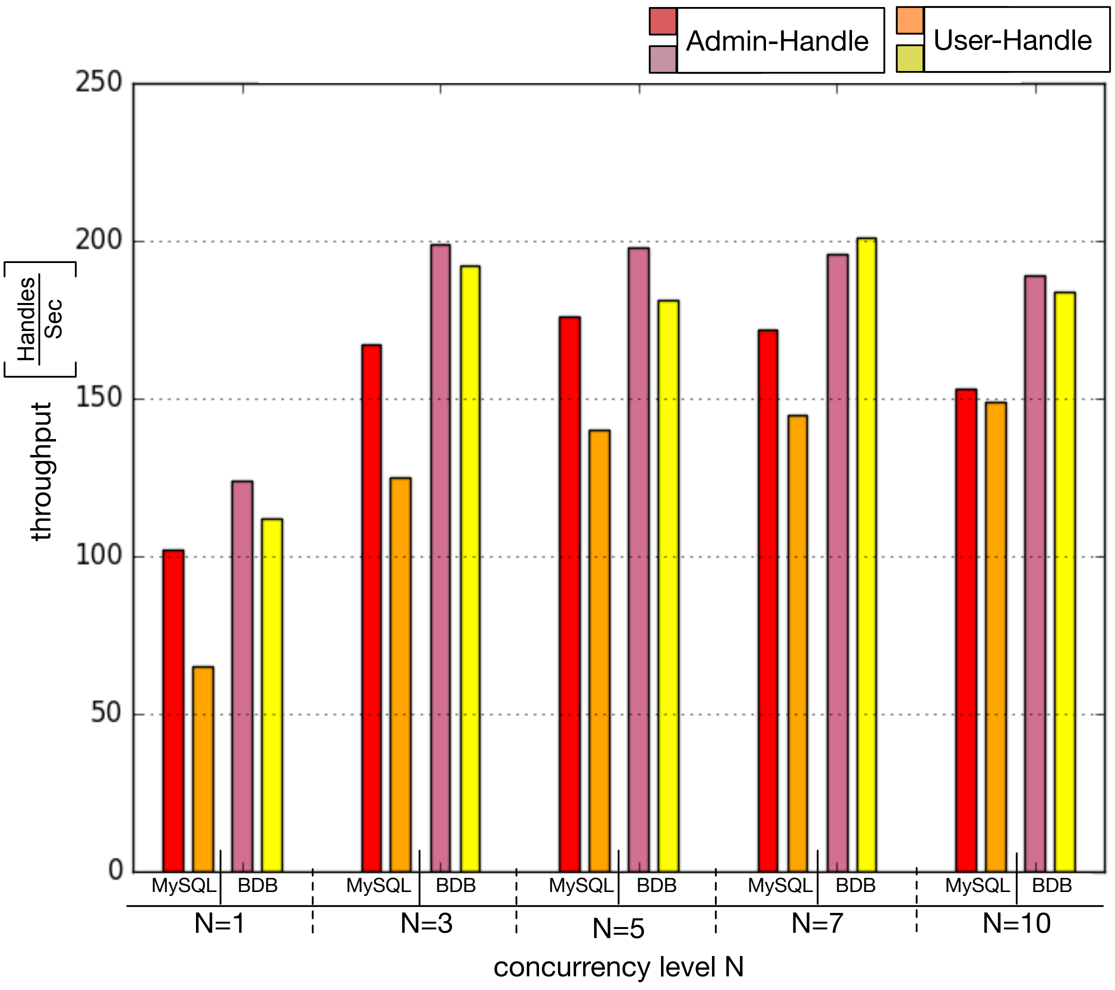
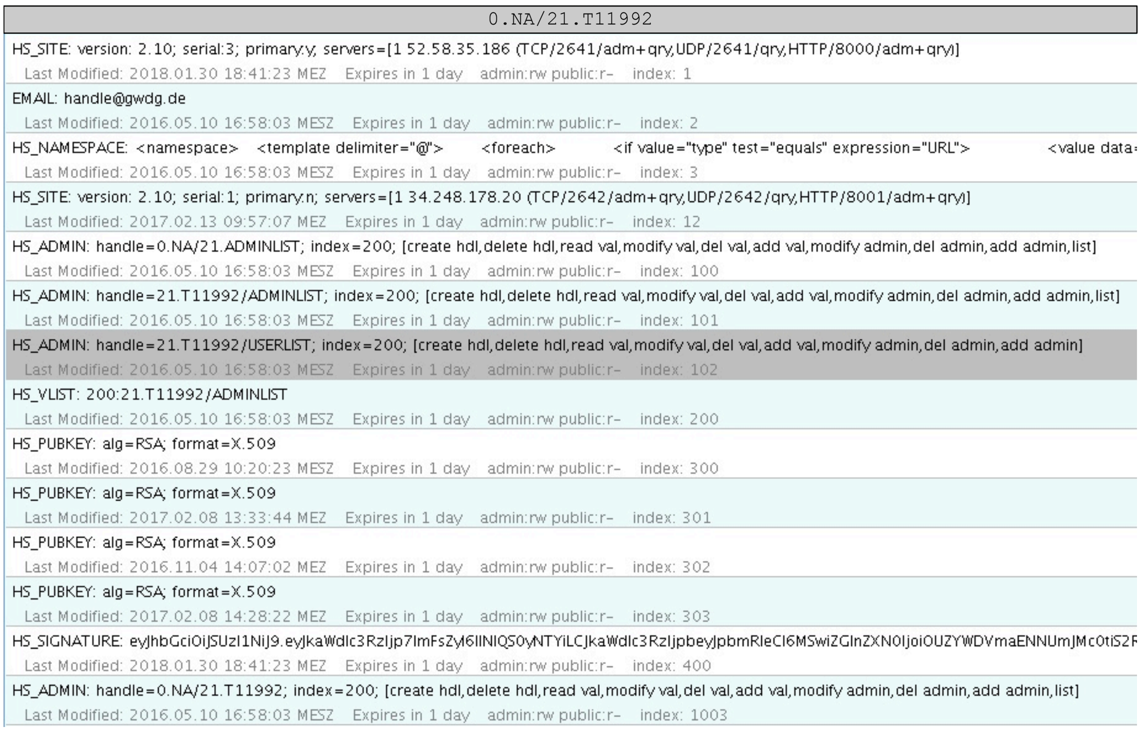


Figure 5.18: Throughput for increasing concurrency level for the `OC_CREATE_HANDLE` operation.



```
0.NA/21.T11992
HS_SITE: version: 2.10; serial:3; primary:y; servers=[1 52.58.35.186 (TCP/2641/adm+qry,UDP/2641/qry,HTTP/8000/adm+qry)]
  Last Modified: 2018.01.30 18:41:23 MEZ   Expires in 1 day   admin:rw public:r-   index: 1
EMAIL: handle@gwdg.de
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 2
HS_NAMESPACE: <namespace>   <template delimiter="@">   <foreach>   <if value="type" test="equals" expression="URL">   <value data=
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 3
HS_SITE: version: 2.10; serial:1; primary:n; servers=[1 34.248.178.20 (TCP/2642/adm+qry,UDP/2642/qry,HTTP/8001/adm+qry)]
  Last Modified: 2017.02.13 09:57:07 MEZ   Expires in 1 day   admin:rw public:r-   index: 12
HS_ADMIN: handle=0.NA/21.ADMINLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 100
HS_ADMIN: handle=21.T11992/ADMINLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 101
HS_ADMIN: handle=21.T11992/USERLIST; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin]
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 102
HS_VLIST: 200:21.T11992/ADMINLIST
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 200
HS_PUBKEY: alg=RSA; format=X.509
  Last Modified: 2016.08.29 10:20:23 MESZ   Expires in 1 day   admin:rw public:r-   index: 300
HS_PUBKEY: alg=RSA; format=X.509
  Last Modified: 2017.02.08 13:33:44 MEZ   Expires in 1 day   admin:rw public:r-   index: 301
HS_PUBKEY: alg=RSA; format=X.509
  Last Modified: 2016.11.04 14:07:02 MEZ   Expires in 1 day   admin:rw public:r-   index: 302
HS_PUBKEY: alg=RSA; format=X.509
  Last Modified: 2017.02.08 14:28:22 MEZ   Expires in 1 day   admin:rw public:r-   index: 303
HS_SIGNATURE: eyJhbGciOiJSUzI1NiJ9.eyJkaWdlc3RzIjp7ImFsZyI6IlNIQS0yNTYiLCJkaWdlc3RzIjpbeyJpbmRleCI6MSwiZGlnZXN0IjoiOUZYWDVmaENNUmJMc0tiS2F
  Last Modified: 2018.01.30 18:41:23 MEZ   Expires in 1 day   admin:rw public:r-   index: 400
HS_ADMIN: handle=0.NA/21.T11992; index=200; [create hdl,delete hdl,read val,modify val,del val,add val,modify admin,del admin,add admin,list]
  Last Modified: 2016.05.10 16:58:03 MESZ   Expires in 1 day   admin:rw public:r-   index: 1003
```

Figure 5.19: Prefix Handle Record of `0.NA/21.T11992`

Method	BulkSize:1,000		BulkSize:5,000		BulkSize:10,000	
	MySQL	BDB	MySQL	BDB	MySQL	BDB
iterative	97	116	104	124	99	126
parallel	186	182	188	205	168	200
bulk	722	16726	755	24639	678	17362

Table 5.2: **FRA→FRA Measurements**: Throughput $[(Handles)/(Sec)]$ comparison between Handle servers attached to MySQL and BDB.

The corresponding throughput values are depicted in Figure 5.18. We can see that for the BDB attached Handle servers the maximum throughput is about 200 $[(Handles)/(Sec)]$, which was achieved for the concurrency level $n \approx 5$. Whereas for the MySQL attached Handle server, the maximum throughput is approximately 175 $[(Handles)/(Sec)]$, which was also achieved for $n \approx 5$.

Finally, our new bulk operation enables to significantly improve the throughput of the Handle server, which is demonstrated in the following subsection.

5.5.3.2 Workload Type WL-B

In this section, we investigate and compare the performance of the new `OC_CREATE_HANDLES_BULK` operation with the original registration operation offered by the Handle protocol. Note that in the remainder of this chapter, we only consider measurements gathered with the administrative user.

The achieved speedup factors for the load-generator located in Frankfurt (FRA→FRA) are depicted in Figure 5.20, whereby the underlying measurements are shown in Figure 5.21 and in Table 5.2.
If we only consider the ratio between the throughputs of the bulk and parallel registration method (Bulk/Parallel), it reveals that by means of the new `OC_CREATE_HANDLES_BULK` operation, we have achieved speedup factor up to 120. With the MySQL database, however, the speedup factor is only about 4, which, nevertheless, still constitutes a significant speedup.

Usually, research data repositories register new PIDs in an iterative manner. The speedup factor for this situation is even up to 200 (Bulk/Iterative).
Furthermore, while Figure 5.21(a) illustrates the throughput, Figure 5.21(b) reveals the per Handle Record response time and its corresponding decomposition in logarithmic scale.

We can see that the tremendous difference in the speedup factors between MySQL and BDB for the new bulk operation is caused by the processing times of the databases. With the MySQL attached Handle server, the processing times of the database (left blue bars) is always within the range of 1×10^0 and 2×10^0 milliseconds. Whereas with the BDB attached Handle server (right blue bars), they are clearly always only in the range of 1×10^{-2} and 2×10^{-1} milliseconds.

In order to determine the impact of network latency onto the registration performance, we also performed the same workload type from the second load-generator, which was located in Ireland. In the following, due to the superior performance with the BDB, we only consider measurements with the BDB attached primary Handle server. The following measurements therefore represent a comparison between the performance of the load-generator located in Frankfurt and Ireland.
Figure 5.22 shows that for the bulk size $BZ = 1,000$, there is a significant performance loss in the throughput due to the network latency between the load-generator located in Ireland and the primary Handle server located in Frankfurt. Moreover, from the corresponding response time

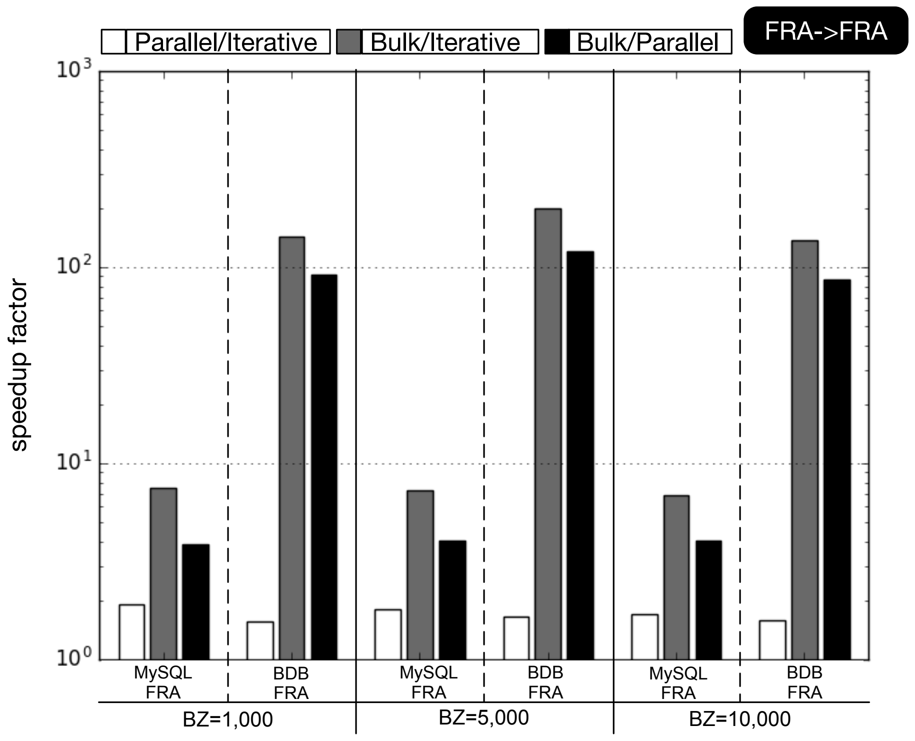


Figure 5.20: **FRA→FRA** Measurements: Speedup factors in logarithmic scale.

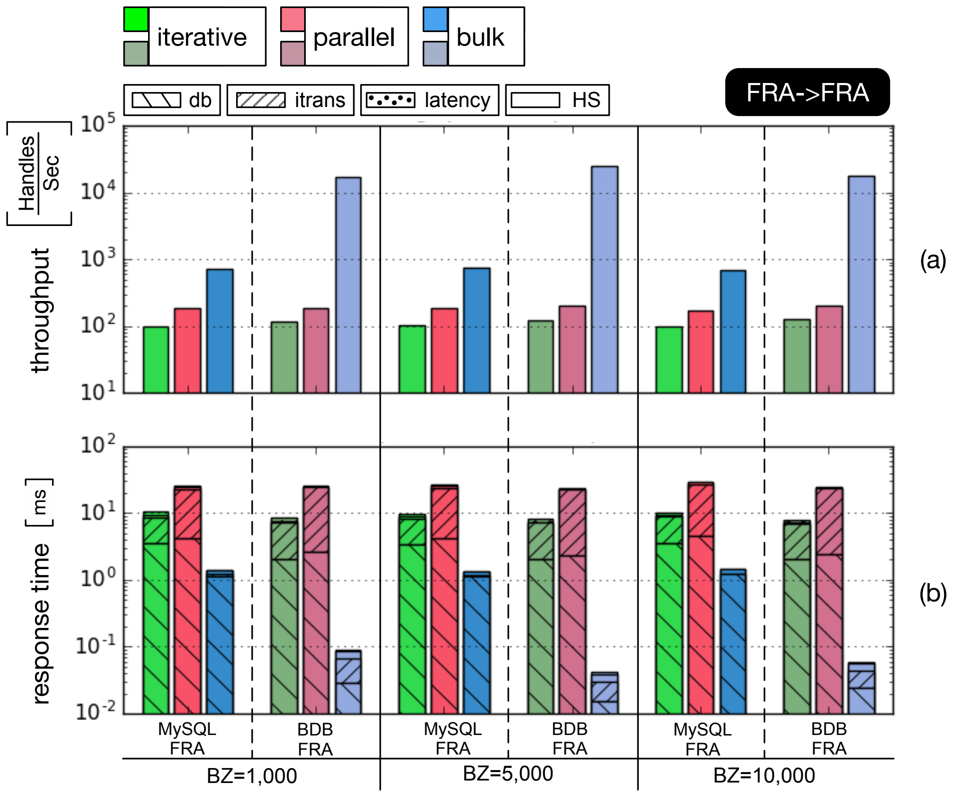


Figure 5.21: **FRA→FRA** Measurements: (a) Throughput in logarithmic scale for different bulk sizes, registration methods and databases attached to Handle server. (b) Corresponding response time decomposition per Handle Record creation, with the following contributions: database (*db*), replication system (*itrans*), *latency*, Handle server (*HS*).

Method	BulkSize:1,000		BulkSize:5,000		BulkSize:10,000	
	IRE	FRA	IRE	FRA	IRE	FRA
iterative	19	116	19	124	19	126
parallel	95	182	95	205	95	200
bulk	5220	16726	12879	24639	15569	17362

Table 5.3: **IRE→FRA Measurements**: Throughput $[(Handles)/(Sec)]$ comparison between Handle servers attached to BDB and different client locations

measurements depicted in Figure 5.23(b), we can clearly see that the response times for the iterative and parallel registration method get superposed with the network latency. The impact of the network latency even dominates the overall response times.
Interesting is that the response times of the iterative and parallel method for the (IRE→FRA) measurements are always almost identical. In contrast to that, for the load-generator located in Frankfurt, the response times of the parallel registration method are always longer than for the respective iterative method.
In comparison to the FRA load-generator, requests which are concurrently sent by the IRE load-generator have usually a significantly longer network path traversal until they arrive at the primary Handle server in Frankfurt. A longer distance usually also implies a wider network path variety with different round trip times. Therefore, we assume that for our applied concurrency level, there was no concurrent processing at the primary Handle server due to the subsequent arrival times of the submitted requests. In other words, our assumption is that concurrently submitted requests arrived subsequently at the primary Handle server, which means that the network also functioned as a load damper.

However, for the new bulk operation, the impact of the network latency became negligible for increasing bulk size. This has resulted the speedup factors (Bulk/Iterative) and (Bulk/Parallel) to become even larger. The maximum speedup between the new bulk and the original registration operation is about 800, whereas the ratio between (Bulk/Parallel) has its maximum at about 160.
A comparison of the throughput in Figure 5.23(a), however reveals that there is also a significant slowdown due to the network latency. The maximum throughput for the FRA load-generator is 24,639 $[(Handles)/(Sec)]$, while for the IRE load-generator, it is only 15,569 $[(Handles)/(Sec)]$. Thus, due to network latency, the throughput of the new bulk operation was approximately halved. The slowdown factor for the iterative registration methods is about six and, for the parallel registration methods it was also halved.

Finally, we have seen that our new bulk operation has enabled to achieve enormous performance gains.

5.5.3.3 Workload Type WL-C

In this section, we analyze the behavior of the primary Handle server equipped with our extended Handle protocol implementation for productive workload.

The analysis of the workload of the various PID systems provided by the GWDG revealed the following frequency of operations.

- single `OC_RESOLUTION` operation: every 1.5 seconds
- single `OC_CREATE_HANDLE` operations: every 2 seconds

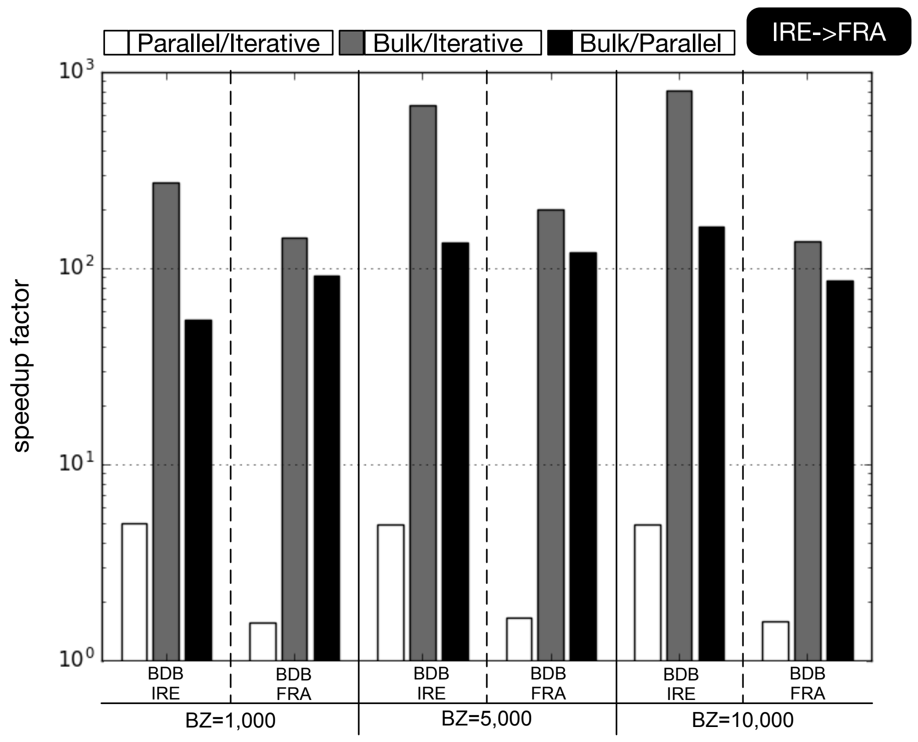


Figure 5.22: **IRE→FRA** Measurements: Speedup Factors in logarithmic scale.

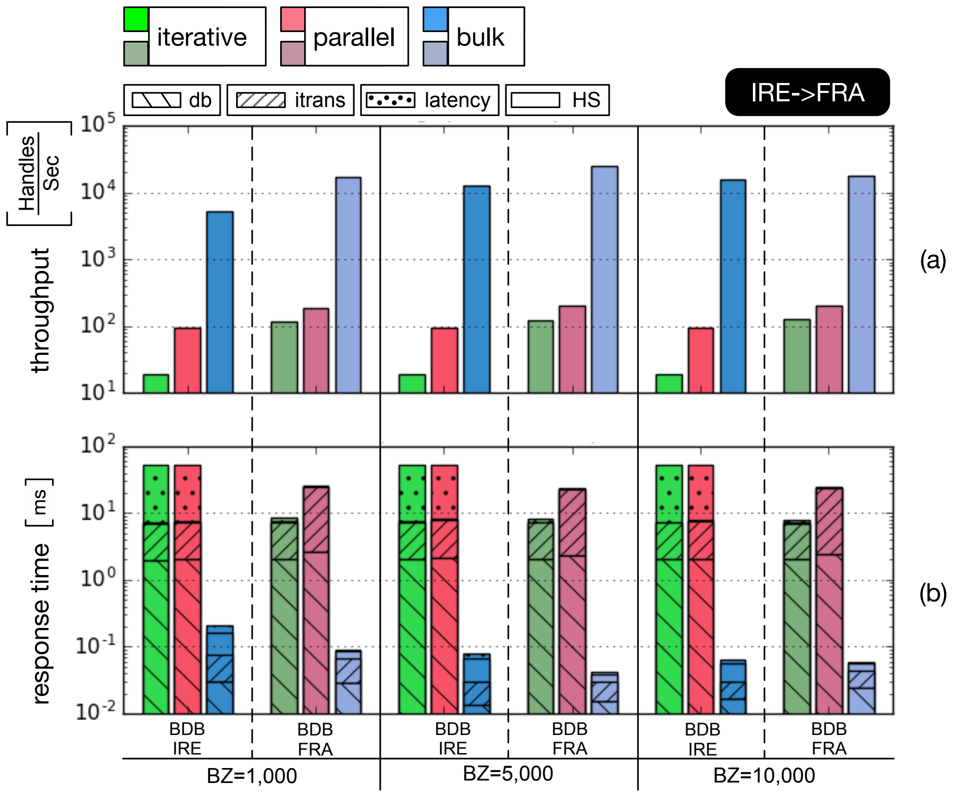


Figure 5.23: **IRE→FRA** Measurements: (a) Throughput in logarithmic scale for different bulk sizes, registration methods and databases attached to Handle server. (b) Corresponding response time decomposition per Handle Record creation, with the following contributions: database (*db*), replication system (*itrans*), *latency*, Handle server (*HS*).

- batch of OC_CREATE_HANDLE operations: These are regular registration operations, which can be grouped into one batch. For these requests, the batch size is often in the range from 800 and 10,000 handles. However, in most cases the batch size is about 1,000.
  Very often such a group of registration operations can be grouped into an one-hour-phase. Within such an one-hour-phase, between two batches of registration operations, there is often an interval of 20 seconds in which only single operations (OC_RESOLUTION, OC_CREATE_HANDLE) appear.
  However, often there are also several one-hour-phases, in which only single operations occur, which can not be grouped into a particular batch.
  Finally, the concurrency level of the single operations is usually around five.

Based on this analysis, in our load-generator, the frequency of a resolution and a single registration operation was set accordingly. In order to determine the stability of the system with the new bulk operation, approximately every ten seconds a OC_CREATE_HANDLES_BULK operation with batch size of 1,000 Handle Records was issued. In addition, another OC_CREATE_HANDLES_BULK operation with a randomly varying batch size ranging from 1,000 to 10,000 Handle Records was issued every five to 40 seconds.
Ultimately, the primary Handle server was additionally frequently stressed with parallel registration requests.

The objectives for such an operation mixture were to examine the following metrics:

- Response time behavior of single registration operations,
- Response time behavior of resolution requests,
- Throughput behavior of the new bulk registration operation.

In our measurements, gathered from this workload type, we could not observe any significant changes in the response time for the resolution operation. The overall response time for the resolution operation was always around a single millisecond, which is also confirmed by the measurements for the BDB attached primary Handle server in Figure 5.17. As can be seen, the response times for the non-administrative user (yellow bars) are only minimally longer than for the administrative user (lighter red bars), which is based on the additional reading access required to authorize the non-administrative user. Hence, this reading access can be considered as to be representative for a resolution request.

In contrast to that, for the registration operations, there were significant effects, when a registration operation got interfered by another such an operation.
Figure 5.24(a) shows a boxplot of the response times for the OC_CREATE_HANDLE operation. Whereas Figure 5.24(b) depicts a boxplot of the throughputs for the new OC_CREATE_HANDLES_BULK operation. In both boxplots, the boxes in the greyed area represent a subset of the respective ”interfered” box to the left.
Before we proceed with the actual analysis of these boxplots, we first provide a brief description of them:

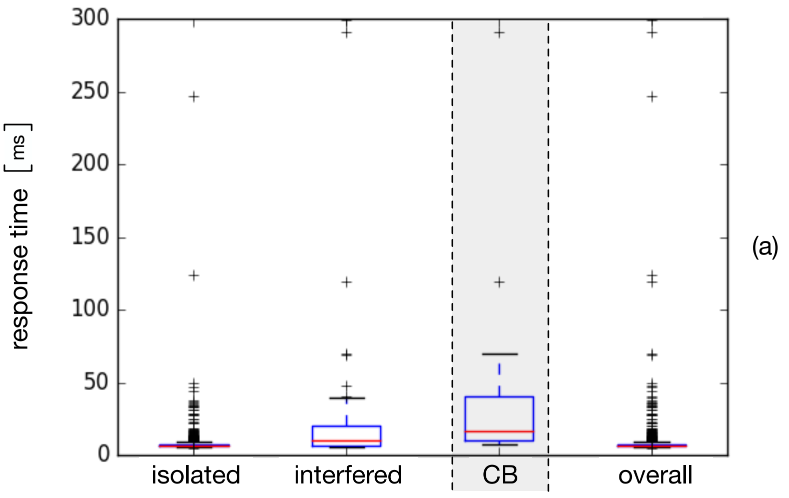
ISOLATED: The boxes with this label, contain the response times / throughputs, when the system was processing only a single OC_CREATE_HANDLE / OC_CREATE_HANDLES_BULK operation.

INTERFERED: These boxes contain the response times / throughputs, when the system was processing multiple operation concurrently.

CB: These boxes contain the response times / throughputs, when the system was concurrently processing a single registration and a single bulk registration operation.

BB: Boxes labeled like this, contain the throughputs, when the system was processing two bulk registration operations.

OVERALL: These boxes, contain all response times / throughputs measured in the one-hour-phase measurement phase.



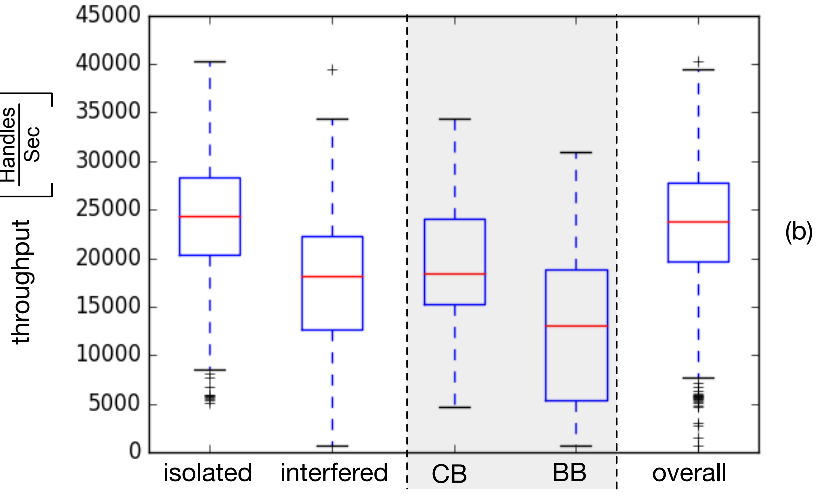


Figure 5.24: Operation interference measurements:
(a) Response time behavior of `OC_CREATE_HANDLE` operation.
(b) Throughput behavior of new `OC_CREATE_HANDLES_BULK` operation.

Finally, we continue with the analysis of the measurements from Figure 5.24.

In comparison to the respective ”isolated” state, there was a significant impact on the response time of the `OC_CREATE_HANDLE` operation and on the throughput of the new `OC_CREATE_HANDLES_BULK` operation, when such an operation was interfered by another administrative operation.
Even in the particular case, when a single registration operation was processed concurrently with a bulk registration operation, labeled as ”CB”, for both operations, there was a significant slow-

down.
In that particular state, for the response time of the single registration operation, the slowdown factor was in average about five. In contrast to that, for the throughput of the bulk operation, it was in average around 1.4.
In the state labeled as ”BB”, in which two bulk operations were processed concurrently, the slowdown factor for the throughput was about 2.

Although, during our one-our-phase, there were significant outliers, especially for the `OC_CREATE_HANDLES_BULK` operation, the extended primary Handle server was still operating properly. This means that by subjecting the primary Handle server with a load level exceeding the productive workload, we could not achieve a destabilization of the bulk-enabled Handle server. This is also confirmed by Figure 5.25, which shows 1,800 successful single registration operations during our one-hour-phase. Remind that the single registration operation was issued in a 2-second frequency, hence $2\text{sec} \times 1800 = 3600\text{sec} \mathrel{\widehat{=}} 1h$.

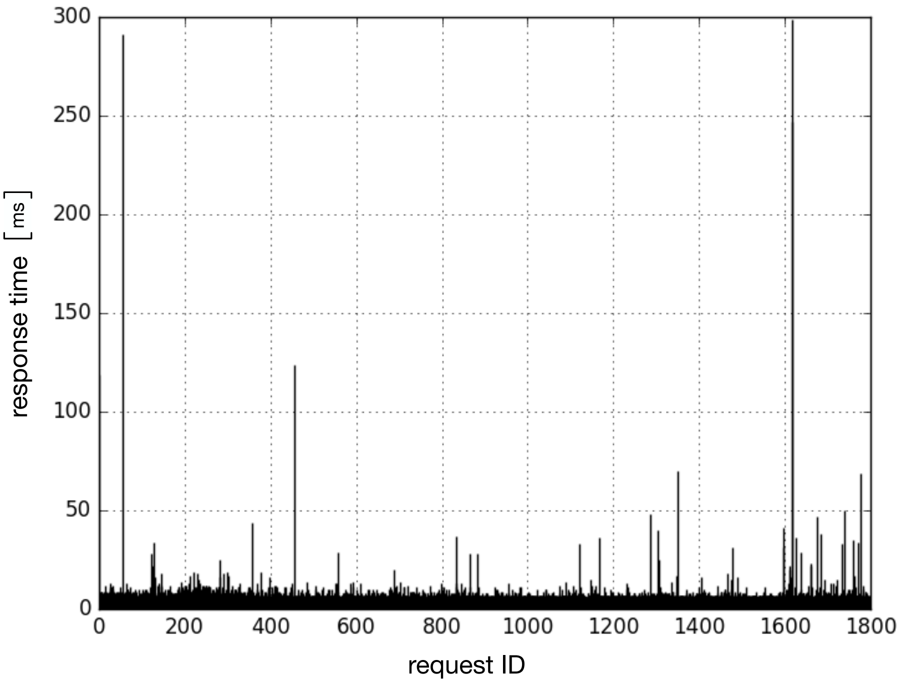


Figure 5.25: Response times of `OC_CREATE_HANDLE` operations during the one-hour-phase.

5.5.4 Summary

In Section 4.5.7 of the previous chapter, we have assumed the Handle server and its working principle with the underlying database to pose a bottleneck. This assumption has been confirmed by the measurements of this chapter. By extending the Handle protocol and the Handle server implementation with a bulk administration operation, we have achieved a speedup factor up to 160 for Handle Record administration.

Although it turned out, contrary to the previous chapter (Chapter 4), that in the setup of this chapter, the response time for administering Handle Records was dominated by the processing time of the Handle server itself (cf. Figure 5.17), the importance of the underlying database has been again revealed during the evaluation section, especially with the tremendous difference of the speedup factors with the MySQL and BDB attached bulk-enabled Handle server. Whereas with BDB the maximum speedup factor was 160, with the MySQL database, it was only about 4, which, however, can still be considered as an enormous improvement. And exactly this performance difference demonstrates again the impact of the database onto the PID administration performance. Nevertheless, the performance analysis of the original Handle server, conducted in this chapter, has also revealed that specific internal components of the Handle server itself, can also cause a significant performance degradation of the overall PID record administration. Therefore, for future research efforts, we propose to revise the internal processing mechanism of the Handle server, which can also lead to a further performance gain with our new bulk operation.

Finally, with our new bulk administration operation, for one million PIDs, we have the following extrapolation:

$$\begin{aligned} max(\text{IRE} \rightarrow \text{FRA}) &\Rightarrow 15569\ [(Handles)/(Sec)] \\ max(\text{FRA} \rightarrow \text{FRA}) &\Rightarrow 24639\ [(Handles)/(Sec)] \end{aligned}$$

$$\Longrightarrow 1{,}000{,}000 \text{ PIDs} \mathrel{\widehat{=}} 40 - 64 Sec$$

In contrast to that, with the parallelized single registration operation, we would have:

$$\begin{aligned} max(\text{IRE} \rightarrow \text{FRA}) &\Rightarrow 95\ [(Handles)/(Sec)] \\ max(\text{FRA} \rightarrow \text{FRA}) &\Rightarrow 205\ [(Handles)/(Sec)] \end{aligned}$$

$$\Longrightarrow 1{,}000{,}000 \text{ PIDs} \mathrel{\widehat{=}} 4878 - 10525 Sec$$

Hence, our Handle protocol extension provides a tremendous acceleration of PID record management. This extension can also be beneficial for other important PID systems, which are based on the Handle System, such as DOI and ePIC.

Chapter 6

High-Performance Persistent Identifier Resolution

So far, in this thesis the objective was mainly on the acceleration of the administration of PIDs. In contrast to that, in this chapter, the focus is devoted on the performance improvement of PID resolution.
To achieve an improvement of the resolution time of Handle-PIDs, we propose to resolve Handle-PIDs through DNS traffic. This is based on the fact that number of DNS proxy resolvers is much higher than the count of Handle proxy resolvers. This huge number of DNS proxy resolvers enables to reduce the latency between a requesting application and the resolver (cf. Figure 6.1).
Starting point of our discussion is a description of the resolution time composition of Handle-PIDs. This is followed by an outline of well-established techniques applied in the DNS system for resolution acceleration. Although, these techniques are also applied in the Handle System, due to the small number of Handle proxy resolvers, they are less efficient than in the DNS system. Therefore, we present a solution, which enables a faster Handle-PID resolution through DNS proxy resolvers.

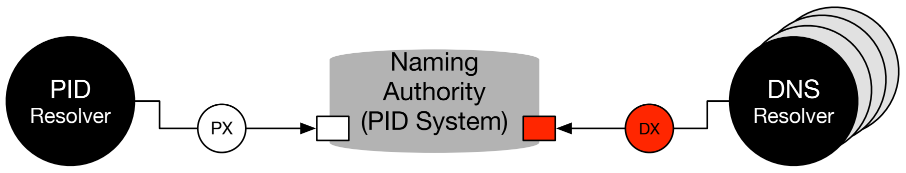


Figure 6.1: Focus in this chapter: Accelerating the resolution of PIDs by using DNS proxy resolvers. This is based on the vast number of globally distributed DNS proxy resolvers. For this, a PID system has to be extended in order to be able to interpret DNS queries (**DX**), which is therefore tackled in this chapter.

The base of our approach has been published in a conference paper at the 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), in Prague, Czech Republic [110]. In this thesis, however, we provide a revised and extended version.

6.1 Resolution Time Composition

This section describes the composition of the resolution times for DNS domain names and Handle-PIDs.

Due to the global distribution and the hierarchical architecture, the resolution of PIDs is usually a latency problem. As it is the case for DNS, the resolution is accomplished by a node traversing, starting from the root node and terminating at a certain responsible node. Figure 6.2 and Figure 6.3 provide an illustration of the resolution procedures for domain names and Handle-PIDs respectively:
To resolve a domain name into its IP-address, an application usually starts by querying its operating system's *stub resolver*, depicted as a green box in Figure 6.2. The stub resolver redirects the resolution request to a pre-configured DNS proxy resolver. Such a proxy resolver then traverses the hierarchical global DNS system until it reaches a DNS server, which can provide the corresponding DNS Resource Record.
To resolve a Handle-PID, an application has to submit the resolution request to the *Global Proxy Resolver* (GPR) (*hdl.handle.net*), which then traverses the hierarchical architecture of the Handle System. Remind that the resolution procedure for the Handle System was already discussed in Section 3.4. Also an improvement of the resolution time was addressed in that section. However, the discussion was constrained on the resolution of the specific group of *Research Data Silo Identifier* (DSID)-prefixed PIDs. In contrast to that, in this chapter, we address a much broader group of PIDs.

The resolution time composition in both systems, DNS and Handle, includes the communication overhead between the involved nodes. For the purpose of clarity, in Figure 6.2 and Figure 6.3, each communication overhead is depicted as directed, numbered graph, while the nodes are represented by the grey boxes.
The communication overhead itself is usually mainly composed of the network latency between the communicating nodes. A resolution procedure involving nodes, with high inter-communication overheads, therefore obviously results in a longer overall resolution time. At the same time, the contributions of the actual processing times of the involved nodes, usually become negligible. This implies, that an improvement at the individual nodes would have only a minimal impact on the overall resolution time.

Another important aspect is that for domain name and Handle-PID resolution, an application has to involve a particular proxy resolver. The difference between DNS and Handle resolution is the fact that for DNS there a myriad of publicly available DNS proxy resolvers, while the GPR currently only consists of five globally distributed proxy servers.

Ultimately, in this section we have pointed out that the resolution of domain names and PIDs is a network latency problem, rather than a data processing time problem. This means that the improvement of individual servers will not lead to a significant acceleration of the resolution time. Instead, only a combination of different techniques can lead to a notable improvement. These techniques are therefore discussed in the following section.

6.2 Resolution Time Acceleration Techniques

In this section, we provide the base for our idea to resolve Handle-PIDs through DNS proxy resolvers. Therefore, we outline several techniques, which are applied to accelerate domain name resolution. In addition, we examine the efficiency of these techniques to improve the resolution time of Handle-PID. We will see that their efficiency in the DNS system is essentially based on the high density of DNS proxy resolvers.

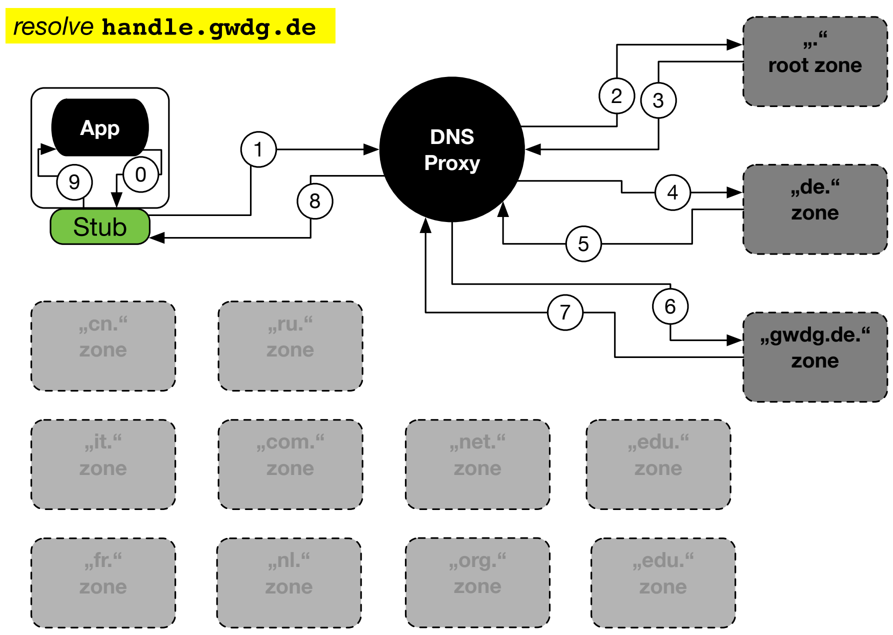


Figure 6.2: DNS Resolution Procedure

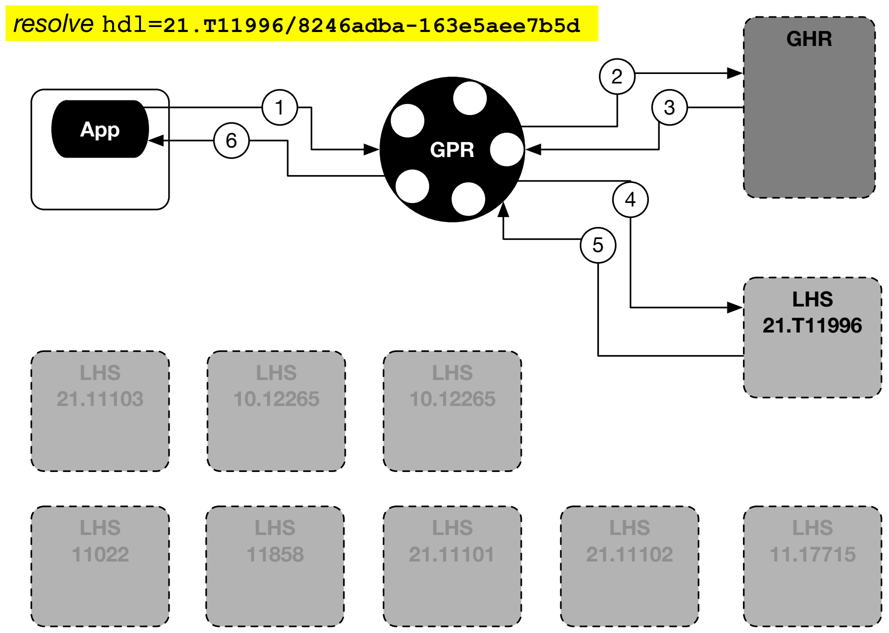


Figure 6.3: Handle Resolution Procedure

6.2.1 Caching

Caching is primarily applied to hold data in faster, and usually also smaller, storage. However, in the particular case of DNS, it is applied to shorten the traversal path. This is achieved by temporarily store resolution responses from previous requests at the traversed nodes. A later resolution request for the same domain name can then be answered directly from a particular node, without the need to traverse until the responsible nameserver.
Thus, for example, caching at a DNS proxy resolver can enable the resolution request to be directly responded by the proxy resolver itself. The resolution time then would be composed of the communication overhead between the requestor and the proxy resolver (Figure 6.2: step ① and ⑧) including an internal processing time.

Since caching is a well established technique for several use cases, also in the Handle System it finds its usage.

6.2.2 Load Balancing

In order to ensure high availability and robustness, an individual DNS zone usually consists of multiple nameservers, on which the incoming resolution requests are distributed on. The application of this technique usually results in the reduction of the internal processing times of the nameservers, which is based on the reduced load. Note that we already discussed the load-dependent processing times in Chapter 4.

Also this technique is applied in the Handle System, since an individual *Local Handle Service* (LHS) is often composed of a primary Handle server and multiple mirror Handle servers.

6.2.3 Anycasting

The technique of *anycast* is often used to redirect the requests to the nearest possible DNS server. Anycast is especially suitable for zones, which consists of multiple geographically distributed nameservers. The *Top Level Domain*s (TLDs) zones are a typical example for that.
The ”de.”-TLD zone for example, consists of many nameservers, which are globally distributed. A request submitted by an application, which is hosted in Europe will be directed to the European cluster of nameservers responsible for the ”de.” zone. Whereas a request originating from the USA will be answered by the nameservers located in the USA.

In anycast, multiple nodes are reachable by exactly the same IP-address. As an example, the public Google DNS resolver is reachable by the IP-address 8.8.8.8. However, since the public Google DNS proxy resolver is globally distributed among the world-wide Google data centers, a request of an individual client will be directed to one of the nearest Google DNS resolvers.
The request routing decision in anycast is made at the network switch level, which choose the particular path, which corresponds to the shortest hop count (number of to be traversed switches) among a set of possible other paths. However, the efficiency of anycasting is also highly dependent on the global distribution of the nodes, which are addressed by a single IP-address.

In the Handle System, there is currently no significant use of anycasting. It is currently only applied within a testbed of the GPR. However, it is less efficient, because of the already indicated low number of GPR servers.

6.2.4 Content Distribution Networks

In *content distribution network*s (CDNs), however, a request routing solely relying on anycasting is usually not efficient enough. This is due to the fact, that static routing decisions based on the hop count, do not cover the dynamic load behavior in CDNs. Therefore, in CDNs, special DNS servers ([111]) have proven to provide a reliable request routing for redirecting an individual requestor to the current most efficient content server. These special DNS servers are usually equipped with probing information, collected from previous requests. Based on these probing information, these special DNS servers enable to apply various complex routing decision algorithms, which take far more parameters than only the sole hop count into account.

The Handle System is currently not deployed within a CDN. However, a CDN can also not be considered as a reasonable platform to improve the resolution time of Handle-PID. The request routing based on DNS would again cause a considerable overhead into the overall Handle-PID resolution time. In addition, the usage of a CDN is often associated with high costs.

6.2.5 Proxy Resolver Latency Reduction

Proxy resolver latency reduction means to reduce the network latency between a requesting application and the proxy resolver.
In DNS, there are myriads of publicly available proxy resolvers offered by specific service providers, which also apply anycasting for their cluster of proxy resolvers. Moreover, institutes belonging to a larger organization usually also have their own domain name zone including a DNS proxy resolver, which can be used by all the participants within the internal network infrastructure. A similar situation can be found for the private Internet connections, where individual users are connected with the Internet via the network provided by a specific *Internet Service Provider* (ISP). Also within such a network, there are multiple DNS proxy resolvers provided by the ISP.
This means that an individual application can usually choose a DNS proxy resolver in its close vicinity (in terms of network latency).

As already mentioned, the number of proxy resolvers within the Handle System is currently only five. Therefore, in contrast to the DNS system, an individual application can often not choose a Handle proxy resolver with a low network latency.
The result is that from the perspective of an individual application, in particular the technique of caching at the Handle proxy resolvers has a usually a much lower efficiency than with the DNS proxy resolvers, which is also true for the technique of anycasting.
Therefore, to globally reduce the communication overhead between an application and the GPR, it is first of all necessary to increase the number of globally distributed proxy resolvers. This could be done by a manual setup, which is theoretically possible but associated with high costs and administration efforts.
The next option would be to make use of an already publicly available and globally widespread resolution infrastructure: the DNS proxy resolvers. The great benefit of this option is that one can make use of the myriad of globally available public DNS proxy resolvers without additional costs and administrative overheads. The result would be an increased efficiency of the technique of caching at the proxy resolver for Handle-PID resolution. The downside, however, is that it requires a translation process between the Handle and DNS protocol.

Finally, in this chapter, we emphasize on:

- The realization of Handle-PID resolution through DNS proxy resolver.

Therefore, the following section discusses the possibilities to enable this approach.

6.3 Handle-PID Resolution over DNS Approach

In this section, we provide an abstract description of the two options, which enable the resolution of Handle-PIDs through DNS proxy resolvers. Note that a concrete realization is presented in the subsequent section, Section 6.4.

6.3.1 Option (A): Embedding DNS Proxy Resolvers into the Handle System

For this option, the public DNS proxy resolvers have to be extended with an additional module, which first detects resolution requests containing Handle-PIDs and secondly, implements the Handle protocol in order to be able to communicate within the Handle System to retrieve the corresponding Handle Record.
The main obstacle of this option is based on the fact that there are various different, and often proprietary implementations of DNS proxy resolvers. For the realization of our proposed idea, it is necessary that all these publicly available DNS proxy resolvers have to be upgraded with such a Handle protocol module. This might be theoretically possible but it is rather unlikely in practice.

6.3.2 Option (B): Embedding the Handle System into the DNS System

With this option, excluding the GPR, all the LHS and the *Global Handle Registry* (GHR), have to extended with a DNS protocol module, which enables to operate with DNS traffic.
In contrast to the DNS system, all components of the Handle System are based on a common software package: the reference implementation of the Handle protocol. Thus, by extending this software package, it is by far easier to achieve a global adoption of this option. Furthermore, there are no modifications in the DNS proxy resolvers required. Another advantage of this approach is that the GPR basically becomes obsolete, which is beneficial for the CNRI[16], the organization who controls the servers of the GPR. Instead, the CNRI could use the freed-up resources to increase the number of GHR servers.
Due to this major advantages with this option, in the remaining we focus on its technical implementation.

6.4 Implementation

In this section, we consider the implementation of option (B) for enabling Handle-PID resolution over DNS traffic. The realization of this approach covers the following four parts:

(1) Handle server with DNS interface,

(2) Mapping of Handle Values into DNS Resource Records,

(3) Representation of Handle-PIDs as domain names,

(4) Appropriate resolution procedure,

for which we provide insight in the following.

[16]http://www.cnri.reston.va.us

6.4.1 Handle Server with DNS Interface

Since our idea is to embed the Handle System into the DNS system, it is necessary to extend the Handle servers with a DNS interface.
It should be noted that the Handle software package already includes a DNS interface implementation, which can be enabled in Handle servers for listening on DNS traffic. However, this DNS interface implementation is intended to enable Handle servers to act as ordinary DNS nameservers. It does not enable the resolution of ordinary Handle Records over DNS traffic, which is our objective. This stems from the fact that a DNS Resource Record is limited to a specific set of permitted data types, whereas Handle Records can contain Handle Values with any type of data. A deeper discussion on this aspect is provided in the following subsections.
With this built-in DNS interface, original DNS Resource Records can be stored as Handle Values in the attached database of the Handle server. Only Handle Values containing real DNS Resource Records can be resolved through this interface. In contrast to that, our approach is to enable the resolution of any type of Handle Value.
The key challenge for this endeavor is to transform Handle Values containing arbitrary data types into Resource Records with a limited ability for data typing.

Nevertheless, we make use of the already built-in DNS interface for request and response encoding/decoding and transmission. The algorithm, however, for requests received at the DNS interface has to be modified in order to realize our proposed approach, which is also discussed in the following subsections.

6.4.2 DNS Resource Record Types

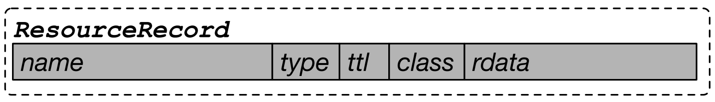


Figure 6.4: Generic DNS Resource Record

The composition of a DNS Resource Record is depicted in Figure 6.4. As already indicated in Section 5.2.3, a Resource Record can basically be reduced to a key-value pair, where the *type* and *rdata* fields represent the key and the data respectively.
Initially, DNS has been conceived for providing human-friendly addressing for computer hosts, which are actually addressed by IP-addresses. Therefore, most DNS Resource Records contain data, which is specifically related with computer hosts. Among them, the Resource Record with the type ”A” can be considered as the most common one, since its corresponding data field contain an IP-address.

Regarding the core function to store a key-value pair, a Handle Value is pretty similar to a Resource Record. However, the main difference between both data structures is that Resource Records are limited to the data type set [112], which is controlled by the *Internet Assigned Numbers Authority* (IANA), whereas Handle Values are not restricted on a specific data type set.
None of the permitted Resource Record types is specifically designated to hold descriptive information about digital datasets. This ability of providing meta information, however, would transform DNS from a simple resolution system into a real global data dissemination system.

To realize our approach, either the permitted set of Resource Records has to be extended by additional types for data dissemination, or it is required to exploit specific types of the permitted set. To core problem of extending the set of permitted types is based on the requirement that public DNS resolvers would also have to be upgraded to support these new types. This is rather challenging, since many public DNS resolvers even lack in the support of all the currently permitted types. Therefore, we emphasize on the exploitation of already permitted types. For that, we have identified a specific set of relevant Resource Record types:

NAPTR Resource Records: NAPTR typed Resource Records have been introduced to enable DNS to act as a rule database for the *Dynamic Delegation Discovery System* (DDDS) [113]. DDDS is basically an abstract concept for applications to enable resource access through applying rules on input strings. The rules are provided by the NAPTR Resource Records, which are then applied by the requesting DDDS applications to compose the locators of resources.
Hence, DDDS applications are compelled to implement a specific algorithm in order to apply the rules contained in NAPTR Resource Records. In contrast to that, in ordinary PID systems, the locator is retrieved directly through the resolution process. Furthermore, with this Resource Record type, it is not possible to retrieve individual metadata about the datasets.

SRV Resource Records: SRV Resource Records are used to describe the available services on hosts. They allow to specify the service, its protocols and the corresponding ports on which the service is listening on at the host. Hence, this Resource Record type does not at all enable to address individual datasets.

URI Resource Records: The URI Resource Record is an alternative to the SRV Resource Record. Although, it allows to hold the URI for a dataset, it lacks in the support of revealing the corresponding metadata.

TXT Resource Records: The TXT Resource Record is intended to transfer arbitrary textual information with DNS traffic. However, in principle it is also possible to transfer any kind of data encoded as a character string. As such, its pretty well suitable for transferring Handle Values.

Finally, among the listed types, the TXT Resource Record type constitutes the most appropriate option. Another beneficial aspect with this type is that it is widely supported by the public DNS proxy resolvers.
The idea of using TXT Resource Records to store key-value pairs is actually not new, as can be seen by [114]. However, so far it has been never considered in conjunction with PIDs.

Ultimately, in the following we provide the actual mapping from a Handle Value into a TXT typed Resource Record.

6.4.3 Mapping of Handle Values into DNS Resource Records

The working principle of our algorithm for mapping a Handle Value into a TXT type Resource Records is depicted in Figure 6.5. In addition, a schematization of the algorithm, which we have implemented into the Handle server is provided by Figure 6.6.

In principle, the mapping of Handle Values into DNS Resource Records constitutes the core part of our approach. Since we have identified the TXT typed Resource Record as the most reasonable option, each Handle Value is mapped into such a Resource Record. The resulting data field (*rdata*)

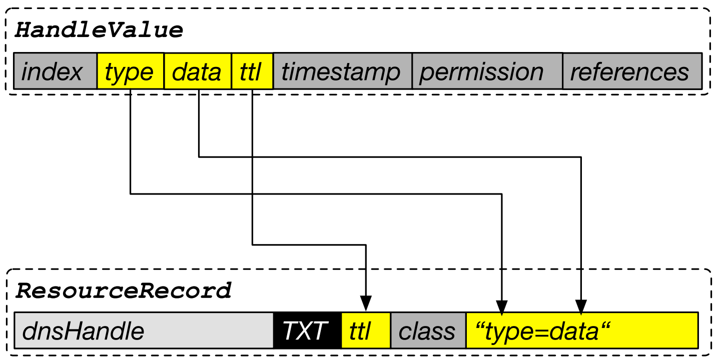


Figure 6.5: Working principle of our implemented mapping algorithm to map Handle Values into TXT typed Resource Records.

then incorporates both, the Handle Value type and its corresponding data field, which are separated by a equal character ”=”. This is illustrated in the right most (yellow) field in Figure 6.5.
The *ttl* field of a Handle Value is mapped into the corresponding *ttl* field of the Resource Record (cf. Figure 6.5).
An important aspect at this mapping algorithm is it that Handle Records may also contain Handle Values, which are not allowed for public consumption. All these restricted Handle Values, which are recognized by the *permission* field, are discarded at the mapping algorithm. Moreover, since a Handle Record can contain multiple Handle Values, the *index* field is actually used as an unique internal identifier within the Handle Record. This, however, is not foreseen for Resource Records and therefore, the index field is also not considered in the mapping algorithm.
The *name* field of the Resource Record is substituted by the domain name representation of the corresponding Handle-PID string. This is dedicatedly addressed in the next subsection.

6.4.4 Representation of Handle-PIDs as Domain Names

Another important aspect for the resolution of Handle-PIDs through DNS traffic is their representation as domain names. The working principle of our transformation algorithm is depicted in Figure 6.7. The *name* field of the corresponding Resource Record is denoted as *dnsHandle* (cf. Figure 6.5).
The deduction of this algorithm is based on the discussion of the following three subsections:

6.4.4.1 Character Set

A Handle-PID can consist of any character from the Unicode character set. In contrast to that, a domain name can only contain letters from A to Z, the digits from 0 to 9, a hyphen ”-” and a dot ”.”, which is used as a separator.
In order to represent each possible Handle-PID as a domain name, there is an additional encoding algorithm necessary. For internationalized domain names (IDNs), which also consist of Unicode characters, there is already such an algorithm available to convert them into regular domain names. This conversion algorithm could be extended to encode a general Handle-PID as a regular domain name. Since, Handle-PID strings are usually composed of known identifiers, such as UUIDs, which are representable as regular domain names without additional encoding efforts, in this thesis, we only focus on such Handle-PIDs. For general Handle-PIDs, there is future research

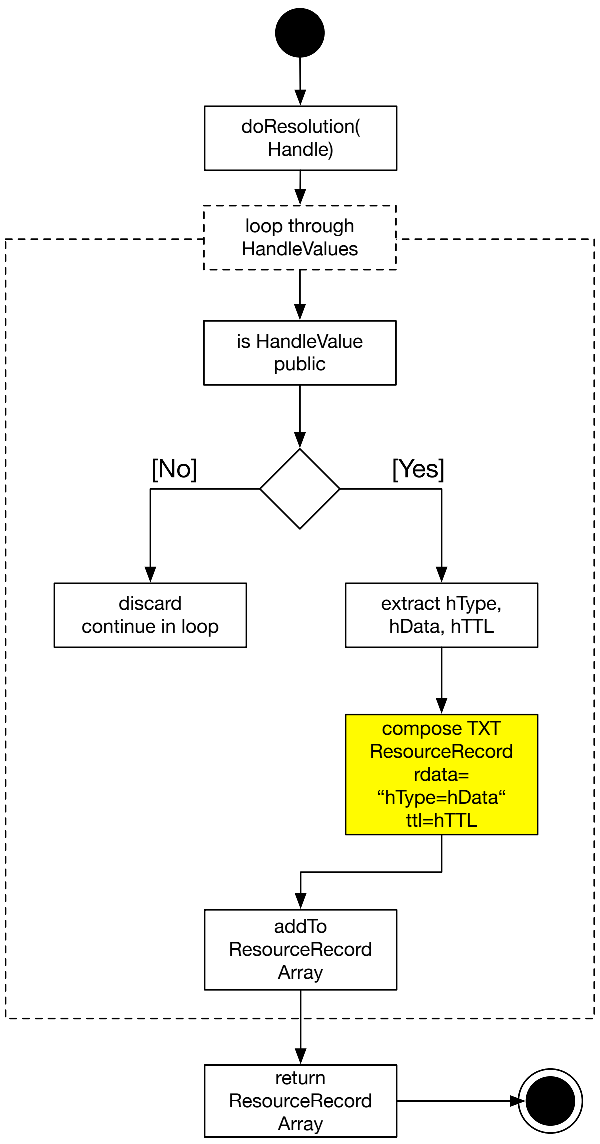

doResolution(Handle)
loop through HandleValues
is HandleValue public
[No]
[Yes]
discard continue in loop
extract hType, hData, hTTL
compose TXT ResourceRecord rdata= “hType=hData“ ttl=hTTL
addTo ResourceRecord Array
return ResourceRecord Array

Figure 6.6: Our mapping algorithm to generate TXT typed Resource Records from Handle Values implemented into Handle servers.

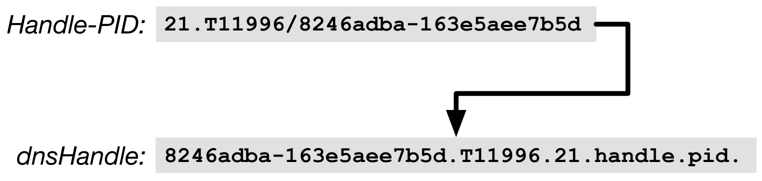

Handle-PID: 21.T11996/8246adba-163e5aee7b5d
dnsHandle: 8246adba-163e5aee7b5d.T11996.21.handle.pid.

Figure 6.7: Working principle of our implemented domain name representation algorithm for Handle-PIDs.

necessary in order to deduce an appropriate conversion algorithm.

6.4.4.2 Namespace Hierarchy

Domain names are composed as hierarchically dot separated labels. From an individual label point of view the label next to the right denotes its parent label. For example, in the domain name ”handle.gwdg.de.” the ”gwdg” label represents the parent of the ”handle” label.
In Handle-PIDs in turn, also the prefix incorporates hierarchically dot separated labels. In contrast to domain names, in the prefix, the rightmost label denotes the lowest level of the hierarchy. Moreover, the prefix is terminated by the slash character ”/”, which is followed by the suffix. For example, in the Handle-PID ”21.T11996/8246adba-163e5aee7b5d”, the ”21” label constitutes the parent for the ”T11996” label, where ”8246adba-163e5aee7b5d” represents the suffix.
In summary, this means that a Handle-PID incorporates two different separators: dots for the prefix and a slash to distinguish the suffix. In order to represent a Handle-PID as a domain name, it is first of all necessary to replace the slash by a character which is allowed for domain names. In our implementation, the slash is replaced by a dot.
A further aspect is that the resulting Handle-PID has to reversed to preserve the correct hierarchy at the resolution procedure through a DNS proxy resolver (cf. Figure 6.7).

6.4.4.3 Global Handle DNS Zone

For the resolution of both, domain names and Handle-PIDs, it is necessary to traverse to the responsible nameserver or LHS respectively.
The prefix of a Handle-PID is controlled at the GHR, which holds the address of the (next) responsible LHS for each individual prefix. Therefore, it is essential that the resolution procedure of Handle-PIDs through DNS proxy resolvers involves the GHR in order to determine the address of the LHS.
This means that the domain name representation of Handle-PIDs has to include a common DNS zone name specifying the GHR. This zone is then composed of the same servers, which form the GHR. We denote this zone as: the *Handle Zone*.
Currently there is only the ”hdl.handle.net.” zone, which is composed of the proxy resolvers of the GPR. For the realization of our approach it is instead necessary to register an additional zone for the GHR, e.g. ”handle.pid.”.

Finally, the resulting domain name of an individual Handle-PID is depicted in Figure 6.7.

6.4.5 Resolution Procedure

In this section, we explain the last part of the implementation of our approach, which is dealing with an appropriate resolution procedure, such as depicted in Figure 6.8.

To resolve a Handle-PID over DNS, the Handle-PID identifier has to be first transformed into its domain representation, which is accomplished in step ⓪. For a Java-based application, we have implemented a transformation module (*HandleDNSResolver*) [115] on top of an already existing Java DNS module (*dnsjava*). The method:

```
getTypesByName(String handle)
```

returns all type-data pairs as JSON string. Another method:

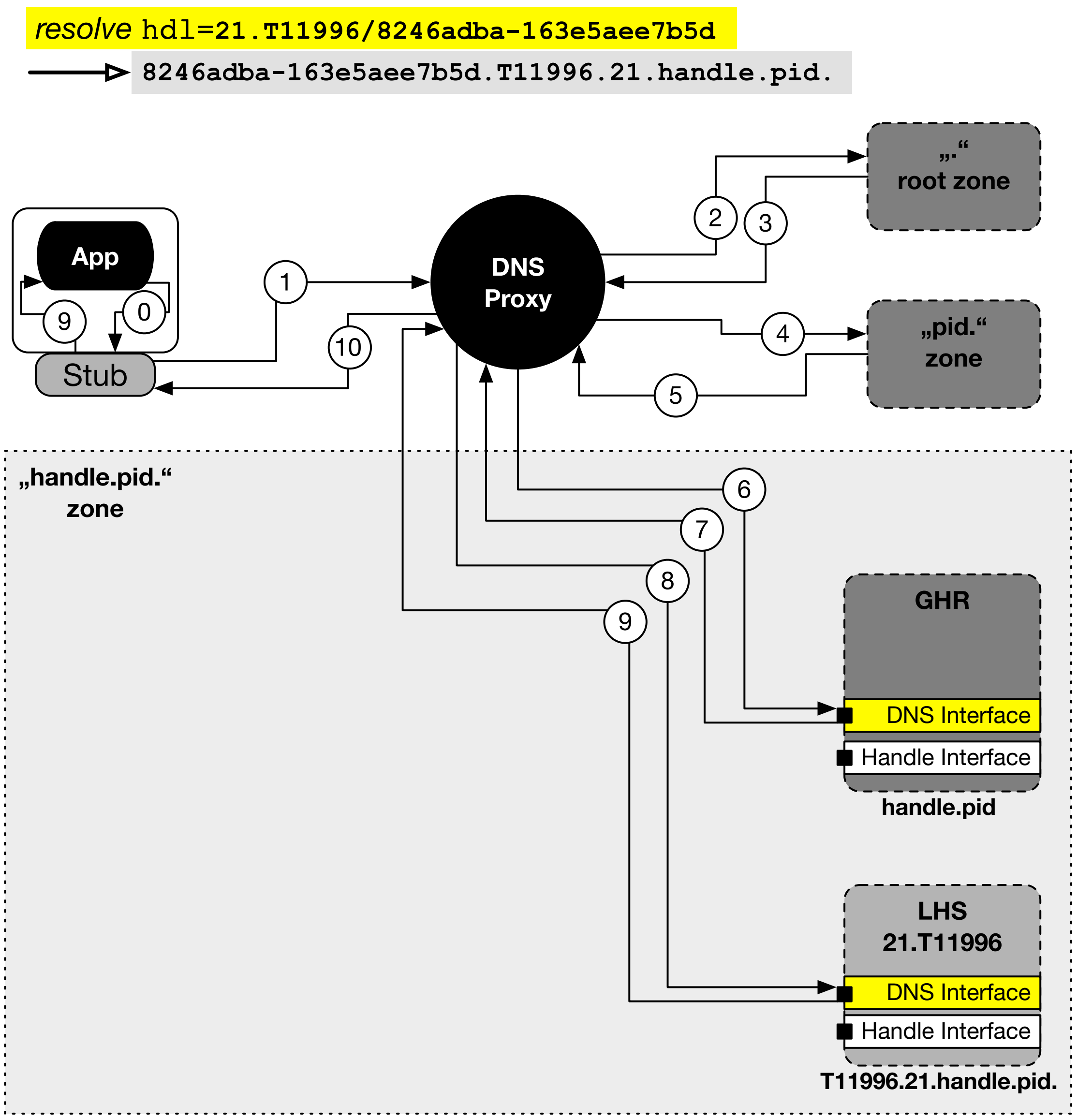


Figure 6.8: The resolution procedure for Handle-PIDs via DNS proxy resolvers.

```
getTypeValueByName(String handle, String type)
```

returns only the type-value pairs for a specified type.
The steps ① to ⑤, are required to be traversed until the actual Handle DNS Zone is reached. In these steps, the DNS proxy resolver communicates with ordinary DNS servers. At step ⑥, the DNS proxy resolver queries the GHR through its enabled DNS interface to determine the authoritative Handle servers for a specific prefix. The GHR then responds with a referral to the (next) responsible Handle server. At step ⑧, the DNS proxy resolver sends the resolution request to the DNS interface of the (next) responsible Handle server.
At the (next) responsible Handle server, the queried domain name is re-transformed into its Handle-PID representation, for which then the Handle Values are retrieved from the database.
After applying the mapping algorithm (cf. Figure 6.5) to generate DNS Resource Records from the retrieved Handle Values, the Handle server responds with the TXT typed Resource Records (step ⑨).
Finally, the Resource Records containing corresponding the Handle Values are sent back to the requesting application (step ⑩).

6.4.6 Overall Algorithm

Our complete algorithm [116] behind the DNS interface of the Handle server is schematized in Figure 6.9. In this illustration, the grey box with the label "getHandleRecord" represents the Handle Value mapping algorithm (cf. Figure 6.6).
For an overall example we refer to Figure 6.10 and Figure 6.11: Figure 6.10 illustrates a Handle Record associated with a TextGrid object. Whereas its representation as a set of DNS Resource Records is depicted in Figure 6.11.

6.5 Evaluation

In this section, we conduct an evaluation of different resolution approaches, including our proposed DNS-based Handle-PID resolution method. The evaluation procedure is comprised of two parts:
The first part covers two aspects:

- the examination of the response time composition of the regular resolution method,
- an analysis to reveal whether the productive proxy resolvers of the GPR are overloaded by the current resolution request load.

Whereas in the second part the objective is on:

- the analysis and comparison of the distinct resolution approaches for different geographical situations (*proximity situations*).

It should be noted that our extension [116] only includes modified or additional files. Therefore, for compilation, it has to be merged with the complete implementation provided by CNRI[17].

6.5.1 Evaluation Setup

For each evaluation part, we made use of a distinct evaluation setup:

[17]http://www.handle.net/download_hnr.html

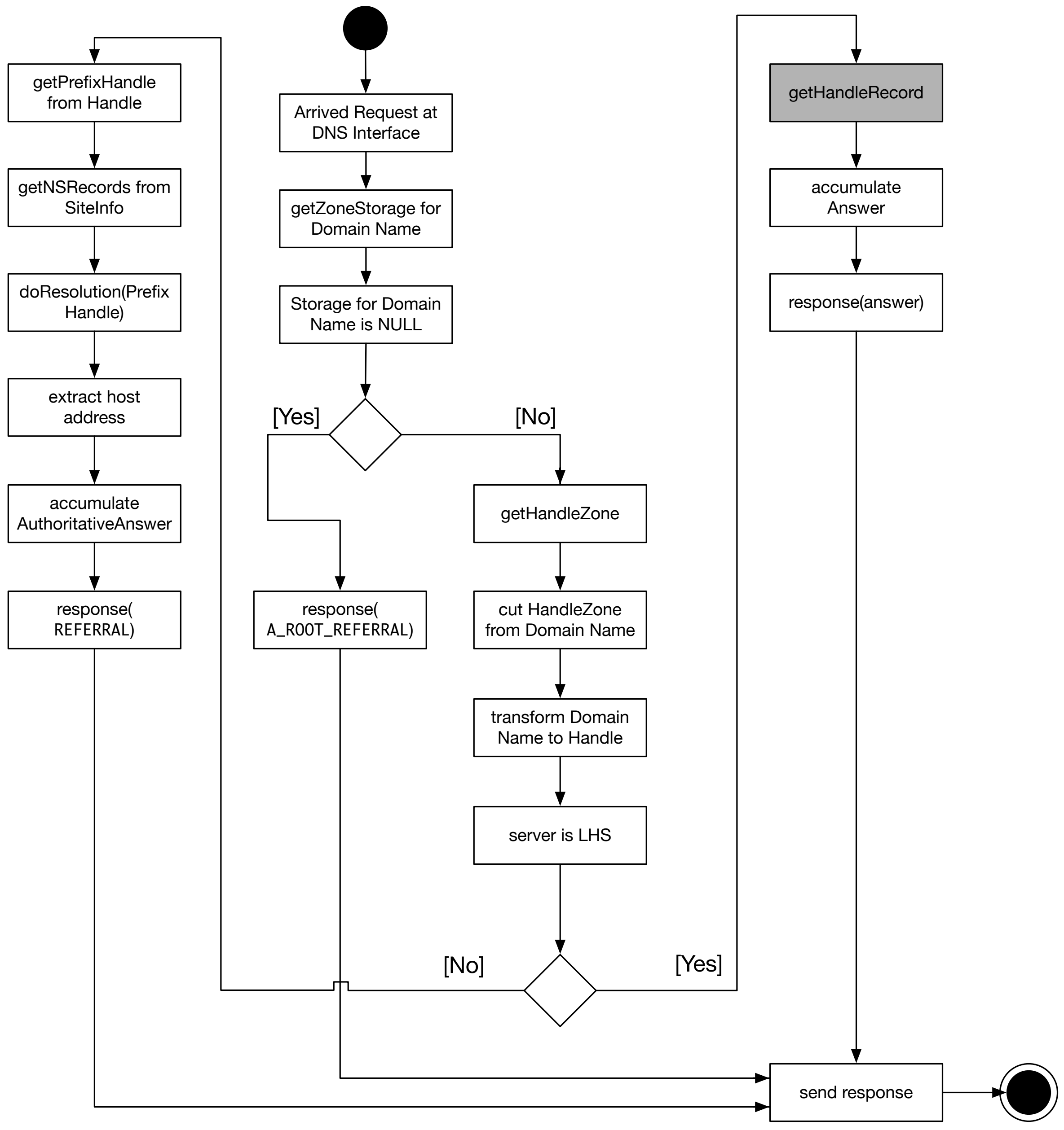
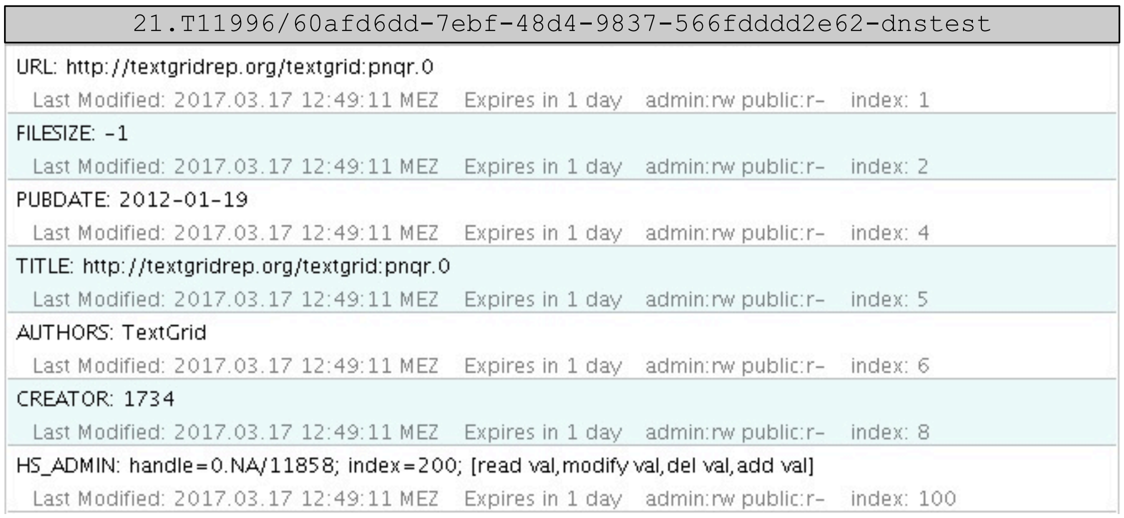

getPrefixHandle from Handle
getNSRecords from SiteInfo
doResolution(Prefix Handle)
extract host address
accumulate AuthoritativeAnswer
response(REFERRAL)
Arrived Request at DNS Interface
getZoneStorage for Domain Name
Storage for Domain Name is NULL
[Yes]
[No]
response(A_ROOT_REFERRAL)
getHandleZone
cut HandleZone from Domain Name
transform Domain Name to Handle
server is LHS
[No]
[Yes]
getHandleRecord
accumulate Answer
response(answer)
send response

Figure 6.9: Our algorithm at DNS interface of Handle server, which has been implemented to realize our approach.



```
21.T11996/60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest
URL: http://textgridrep.org/textgrid:pnqr.0
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 1
FILESIZE: -1
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 2
PUBDATE: 2012-01-19
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 4
TITLE: http://textgridrep.org/textgrid:pnqr.0
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 5
AUTHORS: TextGrid
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 6
CREATOR: 1734
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 8
HS_ADMIN: handle=0.NA/11858; index=200; [read val,modify val,del val,add val]
  Last Modified: 2017.03.17 12:49:11 MEZ   Expires in 1 day   admin:rw public:r-   index: 100
```

Figure 6.10: Example Handle Record consisting of seven Handle Values associated with a TextGrid object.

The first setup, depicted in Figure 6.12, is described by the following paragraphs:

Productive Proxy: We made use of the two GPR proxy resolvers (white circles) located in Europe: the first, hosted at GWDG and the second, hosted on an Amazon EC2 instance in Ireland, which is under the control of CNRI.

Testing Proxy: We placed an additional testing proxy resolver (grey circles) instance in close proximity to each of the two European productive proxy resolvers. These testing resolvers were not involved in the GPR for the resolution of real-world requests.

LHS: The resolution requests were made against `21.T11996`-prefixed Handle-PIDs, which are managed by a LHS hosted at GWDG. The LHS is composed of a primary and a mirror Handle server.

Load-Generator: The resolution requests were generated by a load-generator (black box) hosted on an Amazon EC2 instance located in Frankfurt. Note that the aforementioned testing proxy resolvers were only subjected to requests from our load-generator.

As already indicated, in the second evaluation part, we investigate the resolution approaches for different *proximity situations*. By proximity situation, we mean the relative proximity between a requesting application, a GPR proxy resolver and the target LHS. In principle, there are three possible constellations:

Proximity Situation (A): In this situation, the requesting application, the GPR proxy resolver and the target LHS are all close to each other.

Proximity Situation (B): In this situation, there is a GPR proxy resolver in the vicinity of the requesting application, while the target LHS is located significantly outside of this vicinity.

Proximity Situation (C): In this situation, all these three nodes are significantly distanced to each other.

```
; <<>> DiG 9.9.7-P3 <<>> 60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de ANY
;; Got answer:

;; QUESTION SECTION:
;60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. IN ANY

;; ANSWER SECTION:
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "URL=http://textgridrep.org/textgrid:pnqr.0"
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "CREATOR=1734"
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "AUTHORS=TextGrid"
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "PUBDATE=2012-01-19"
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "FILESIZE=-1"
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "HS_ADMIN=04700000000A302E4E412F313138353800000
60afd6dd-7ebf-48d4-9837-566fdddd2e62-dnstest.T11996.21.hx.gwdg.de. 86400 IN TXT "TITLE=http://textgridrep.org/textgrid:pnqr.0"

;; AUTHORITY SECTION:
T11996.21.hx.gwdg.de.   86400   IN  NS  gwirods-v-src03.gwdg.de.
T11996.21.hx.gwdg.de.   86400   IN  NS  gwirods-v-src02.gwdg.de.
```

Figure 6.11: Resource Record representation of the example Handle Record from Figure 6.10. The ANSWER SECTION is composed of the seven Handle Values, which have been mapped into TXT Resource Records.

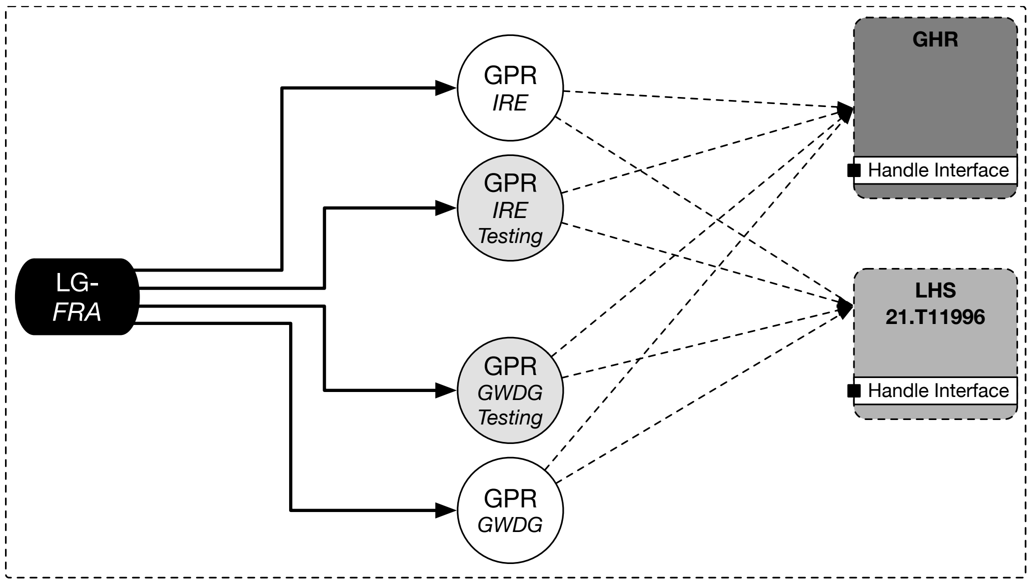


Figure 6.12: First Evaluation Setup

Furthermore, by the following listing, we provide an overview about the different used resolution methods:

- native Handle protocol resolution (`OC_RESOLUTION` operation, cf. Table 5.1)
- resolution through the GPR proxy resolvers
- resolution through pre-configured internal EC2 instance DNS proxy resolver
- resolution through public Google DNS proxy resolver

Finally, Figure 6.13 provides an overview about the second evaluation setup, where the individual components are described by the following paragraphs:

Testing GHR: We setup a testing instance of the GHR, which was equipped with the DNS interface. This testing GHR was composed of a primary server, hosted at GWDG, and a mirror server hosted on an Amazon EC2 instance located in US east. These servers were also the nameservers for our experimental Handle DNS Zone "hx.gwdg.de.", a sub zone under the GWDG zone "gwdg.de.".

LHS: As the target LHS, we used the DNS interface extended version of the LHS of the first evaluation setup. Since this LHS was responsible for Handle-PIDs under the `21.T11996` prefix, the Handle servers were at the same time the nameservers for the "T11996.21.hx.gwdg.de." zone.

Load-Generator: We positioned three load-generator instances at different geographical locations: The first, in Frankfurt (*LG-FRA*), the second, in US east (*LG-US*) and the third, in Singapore (*LG-AS*). All of them were hosted on an Amazon EC2 c3.xlarge instance at the respective region.
Each of the load-generators was equipped with three different modules: The first module was used to resolve Handle-PIDs through the native Handle protocol (`OC_RESOLUTION`), which is represented by the grey paths in Figure 6.13. The second module was used to resolve Handle-PIDs via the GPR, the current standard resolution approach, which is repre-

sented by the red paths in Figure 6.13. The third module made use of our *HandleDNSResolver* to resolve Handle-PIDs over DNS traffic, our proposed new resolution approach, which is represented by the blue paths in Figure 6.13.

GPR Proxy Resolver: To achieve the above defined proximity situations, each load-generator was configured to submit resolution requests against a specific set of proxy resolvers of the GPR.
The *LG-FRA* was configured to issue requests against the two European proxy resolvers. Since in this constellation the load-generator, the proxy resolvers and the target LHS can be considered to be close to each other (all hosted in Europe), this setup represents the proximity situation (A).
In contrast to that, the *LG-US* (US east) was configured to sent resolution requests for Handle-PIDs stored at a LHS hosted in Europe against the three remaining proxy servers located in the US east region, which therefore represents the proximity situation (B).
Finally, the *LG-AS* (Asia) was configured to resolve the Handle-PIDs stored at a LHS hosted in Europe via the regular DNS round-robin proxy resolver selection method. Hence, all resolvers of the GPR (US east / Europe) were involved. This setup can be considered as to be representative for proximity situation (C).

DNS Proxy Resolver: To resolve Handle-PIDs over DNS traffic, we made use of two different DNS proxy resolvers. The first one was the internal Amazon DNS proxy resolver, which was pre-configured in the EC2 instances.
The second one was the well-known public Google DNS proxy resolver, which is reachable via the 8.8.8.8 IP-address.

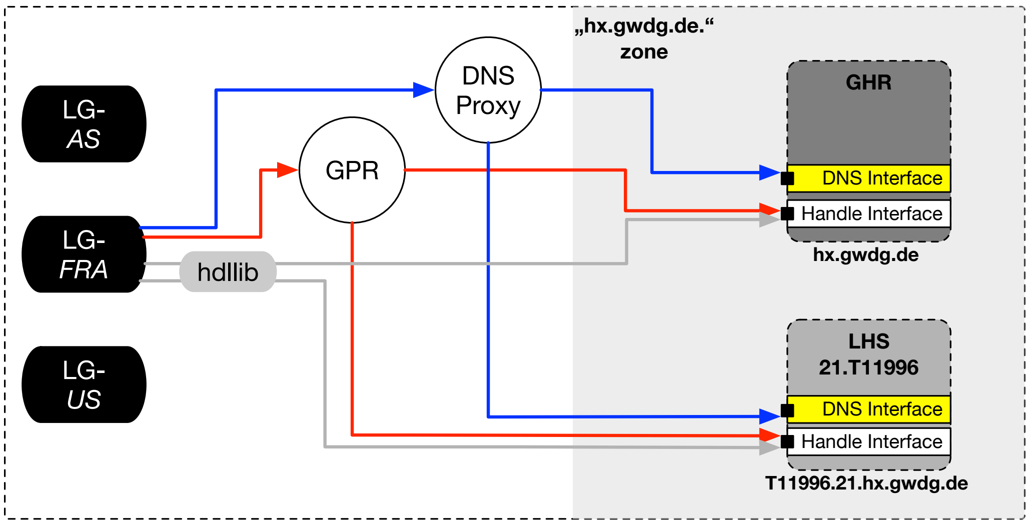


Figure 6.13: Second Evaluation Setup. Each colored graph denotes a distinct resolution method. Note that in addition to *LG-FRA* also the remaining load-generators were involved in this evaluation part.

6.5.2 Part 1: Measurements of the First Setup

In this section, we analyze the measurements gathered from the first setup:

In Figure 6.14, we can see, when we compare the resolution times of the European productive

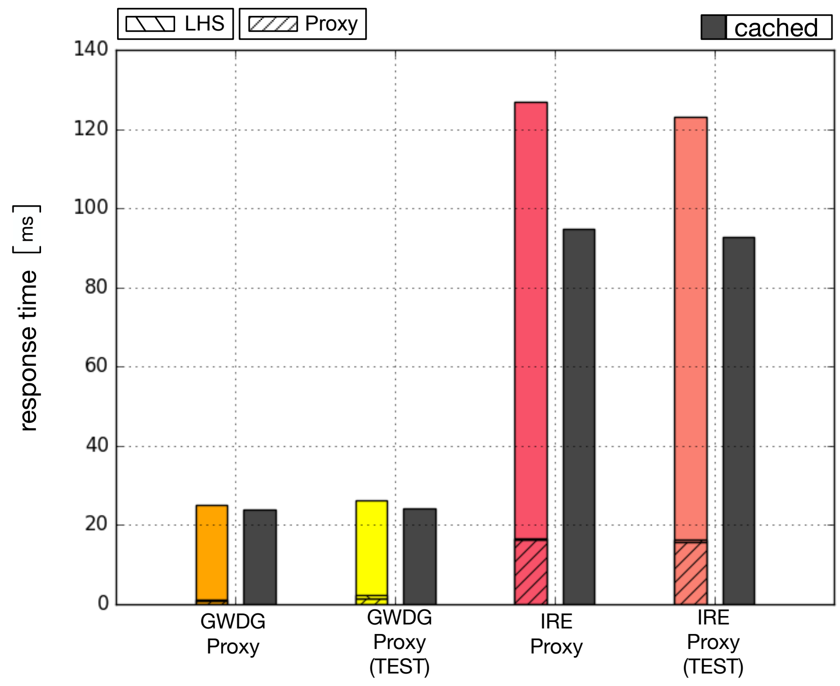


Figure 6.14: **First Setup:** Resolution times for European GPR proxy resolvers. The contributions of the LHS and the proxy resolver are represented as hatched areas on the bars. The black bars represent the resolution times for Handle-PIDs, which are cached at the proxy resolvers

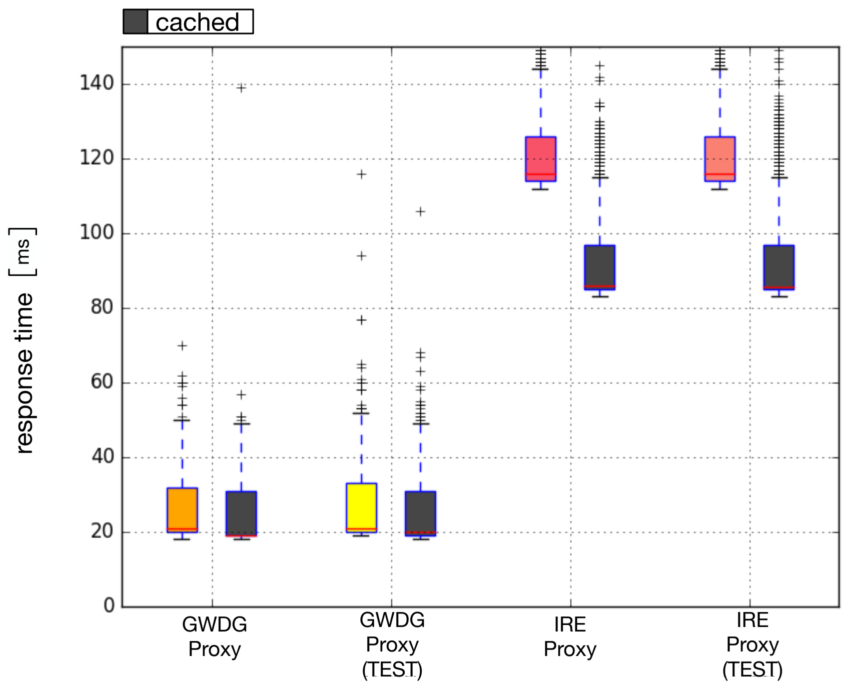


Figure 6.15: **First Setup:** Boxplot of resolution times for European GPR proxy resolvers.

proxy resolvers with the resolution times of the respective testing resolvers (orange bar with yellow bar and red bar with lighter red bar), that the productive GPR resolvers are basically not overloaded with the current request load. Remember that these testing proxy resolvers were only subjected to iterative resolution requests from the load-generators only. In contrast, the productive proxy resolvers are subjected to many resolution requests from various requestors.
Only for the proxy resolvers located in Ireland, the resolution time of the productive proxy resolver is slightly higher than for the respective testing resolver, which, however, can not be considered as an indication for overloading.
In addition, the boxplot in Figure 6.15 shows that the productive and the respective testing proxy resolvers have quite similar distributions in the resolution times, also confirming that the productive proxy resolvers are currently not overloaded.

The response time contributions are represented in Figure 6.14 by the hatched areas on the bars. And since the hatched areas represent the contribution of the proxy resolvers and the LHS, the remaining areas correspond to the latency between the load-generator and the respective proxy resolver. However, the contribution of the LHS is barely identifiable, which means that the actual database lookup has only a minimal impact onto the overall resolution time.

Ultimately, this means that by only positioning additional proxy resolvers at a specific region will not lead to an improvement of the resolution time for Handle-PIDs. Instead, it is necessary to place significantly more proxy resolvers around the world, wherefore we propose to utilize the DNS proxy resolvers for Handle-PID resolution.

6.5.3 Part 2: Measurements of the Second Setup

The actual evaluation of our proposed DNS approach is tackled in this section. In the following subsections, we analyze the measurements gathered for each of the defined proximity situations.

6.5.3.1 Measurements of Proximity Situation (A)

Figure 6.16 reveals that the resolution via the `OC_RESOLUTION` operation (labeled as ”HDLLIB”) is the fastest resolution method. Since this approach directly communicates with the GHR and LHS, there is no overhead caused by the involvement of any proxy resolver of the GPR. The disadvantage of this approach is the requirement for the implementation of the Handle software package into the respective application. This, however, could entail a significant redesign for the application due to appropriate handling of the requests and the responses through the Handle protocol.

In this constellation (proximity situation (A)), the resolution through the GPR proxy resolver located in Ireland has the longest resolution time (red bar), even for cached (black area) Handle Records.
The resolution via the GWDG proxy resolver consists mostly of the latency between the Amazon EC2 instance and the proxy resolver (black area on orange bar). Note that for cached Handle Records (at the proxy resolver), the resolution time mainly consists of the communication overhead between load-generator and the proxy resolver.
In contrast to that, the resolution time with the pre-configured Amazon DNS resolver primarily consists of the latency between the LHS and the DNS proxy resolver (green area on the ”EC2 DNS” labeled bar). Although the resolution times with the these two resolvers (GWDG PROXY and EC2 DNS) are similar, with caching, the resolution time via the EC2 DNS proxy resolver is significantly shorter.

We can also see that our DNS algorithm implementation does not cause any additional overhead into the resolution time. Otherwise, the resolution time with the EC2 DNS proxy resolver would be longer than with the GWDG Handle proxy resolver.
Also remarkable is that the resolution times with the EC2 DNS are more stable than with the proxy resolver hosted at GWDG, which can be seen from the deviation whiskers plotted on the bars.

The resolution time with the public Google DNS proxy resolver (blue bar) reveals to be the second slowest in this measurement. However, again with caching, for Handle Records which have been previously requested, the resolution time becomes significantly reduced. In our setup, apart from the resolution via the Handle protocol, the resolution time with the Google DNS resolver was even the second fastest method for cached Handle Records (black area on blue bar). From the deviation whiskers, we can also see that with this DNS proxy resolvers, the resolution times are quite stable.

6.5.3.2 Measurements of Proximity Situation (B)

The measurements of this proximity situation are illustrated in Figure 6.17.

With the internal EC2 DNS proxy resolver, the resolution time is only minimally longer than with the native Handle protocol resolution operation. In principle, this is also true for the remaining proxy resolver involving resolution methods ("HDL PROXY" and "Google DNS"). All corresponding resolution times are basically of similar magnitude. With "HDLLIB" and "EC2 DNS" the resolution time for cached Handle Records is barely recognizable, which underlines again the benefit of our DNS-based resolution idea. Although the resolution time for cached Handle Records with "HDL PROXY" and "Google DNS" are also of similar magnitude, with "Google DNS" the resolution times are more stable than with the former.

Ultimately, for this proximity situation, although there is again an improvement of the resolution time, it is not that significant than in proximity situation (A).

6.5.3.3 Measurements of Proximity Situation (C)

Figure 6.18 represents the measurements of this proximity constellation.

As can be seen, the resolution time of the proxy resolver involving resolution methods are significantly higher than with the native `OC_RESOLUTION` operation, which is especially true for the internal Amazon EC2 DNS proxy resolver. The resolution time with this proxy resolver is the longest and the most unstable among the remaining resolution time bars. In contrast to that, the regular GPR proxy resolvers and the Google DNS proxy resolver revealed to have similar resolution times, which is also true for the stability, which can be seen from the corresponding deviation whiskers.
However, the resolution times for cached Handle Records with both DNS proxy resolvers are significantly shorter than with the regular GPR proxy resolvers. Note that due to the small resolution times of cached Handle Records at the DNS proxy resolvers, compared to the remaining resolution times, the contribution of their resolution times ($\approx 10ms$) (black areas on green and blue bars) are barely identifiable in Figure 6.18.

This once more underlines the real benefit with the new DNS resolution approach for Handle-PIDs. With the DNS proxy resolvers, the caches are in the vicinity of the requesting applications, which in conjunction with the proximity principle can globally lead to a significant reduction of the Handle-PID resolution time.

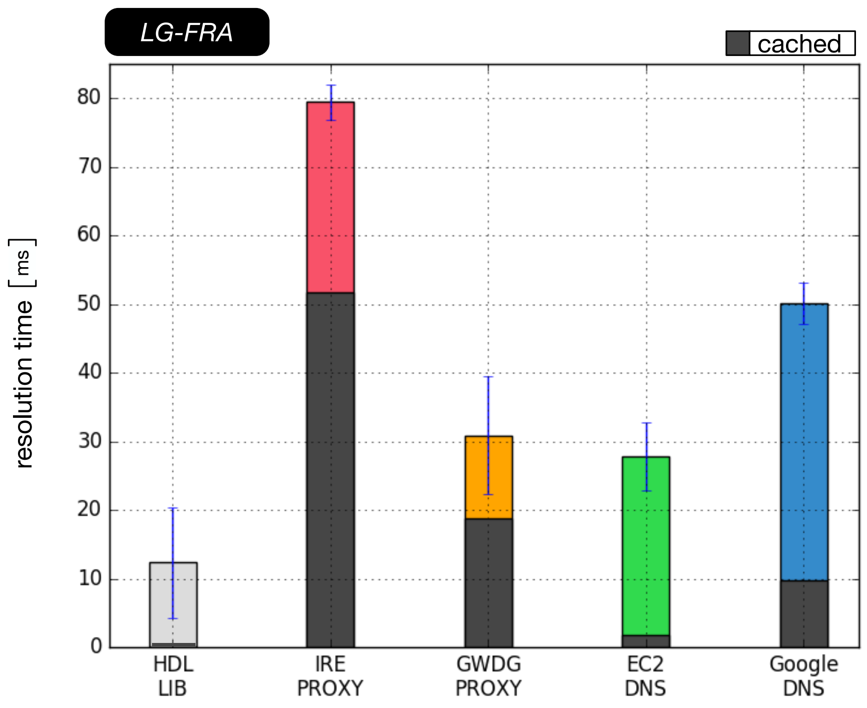


Figure 6.16: **Proximity Situation (A):** Resolution times for different resolution methods, recorded at the load-generator located in Frankfurt.

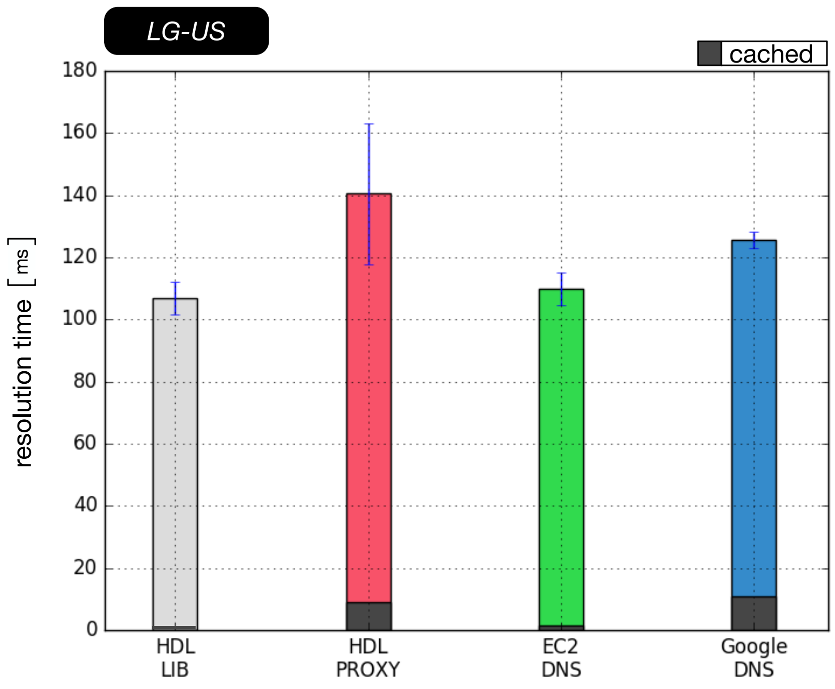


Figure 6.17: **Proximity Situation (B):** Resolution times for different resolution methods, recorded at the load-generator located in US east.

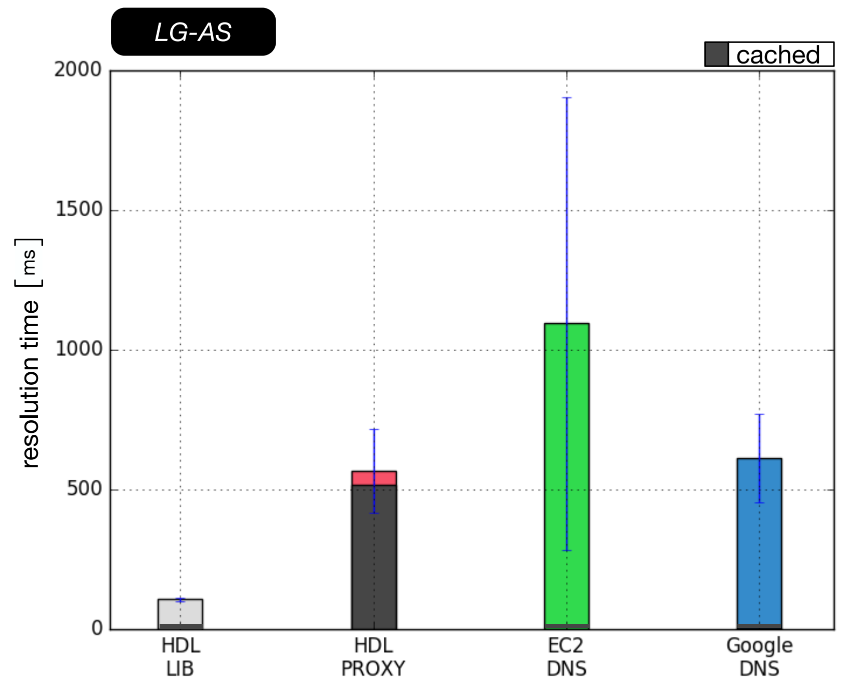


Figure 6.18: **Proximity Situation (C):** Resolution Times for different resolution methods, recorded at the load-generator located in Singapore. The resolution time of cached Handle Records with the current standard resolution method through the GPR (”HDL PROXY”), reveals the inefficiency of caching due to the small number of GPR proxy resolvers.

6.5.4 Summary

We have seen that the actual benefit of the resolution through DNS traffic is based on the reduced latency between a requesting application and the caches. This in turn is based on the myriad of globally distributed DNS proxy resolvers. This approach enables the technique of caching to be much more efficient than in the regular resolution approach. For important and frequently used Handle Records in a specific region, our DNS approach can profoundly improve the performance of the research data dissemination.
But also for Handle Records which not cached, our DNS-based has proven to useful, especially for the proximity situations (A) and (B). For proximity situations (C), the benefit highly depends on the proper choice of the DNS proxy resolver, which is actually not critical due to their high density distribution.
As an example, for a group of research data repositories hosted in Asia, which consume/exchange a common set of research datasets addressed by Handle-PIDs, our DNS approach can lead to a significant performance improvement. Often research data repositories process data in so called *sessions*, wherein a certain number of interlinked external research datasets are used.

Based on the measures in Figure 6.18, for a session requiring 100 interlinked external datasets, our DNS approach enables the following extrapolation for the overall processing time of the session:

- session time with Handle Records cached in GPR proxy resolver:
  - $100 \times 500ms = 50sec$
- session time with Handle Records cached in (Google) DNS proxy resolver:
  - $100 \times 10ms = 1sec$

This means that the overall session time would have been improved about $98\% (= \frac{50-1}{50} \times 100)$.

Furthermore, we have also seen that the current GPR proxy resolvers are not overloaded. This means that by simply adding a few more proxy resolvers to the GPR will not lead globally to a significant improvement of the Handle-PID resolution time.

Chapter 7

Discussion

In this section, we discuss the results of this thesis by answering the research questions defined at the beginning in Section 1.2. In addition, for each approach, we provide a discussion about the respective limiting aspects. In the final section of this chapter, we also provide a discussion about the future design of PID systems based on our results.

7.1 Answers to Research Questions Concerning the Concept of Persistent Identification

The deduced answers are based on the discussion in Chapter 3.

In summary, the concept of persistent identification is performance relevant, because it poses a particular additional maintenance overhead, which is composed of a certain number of expensive transactional administration operations (RQ 1).
In Section 3.1.2 we have identified the overall goal of the concept of persistent identification (RQ 1.1) to be persistent naming and persistent access. This is achieved by an overlay network on top of the current Internet, which is composed of specific naming authorities. Each naming authority in turn, is used to register and manage a particular set of PID-to-locator bindings.
Since with the concept of persistent identification, data mobility has to be continuously tracked at specific naming authorities, persistent access results in a continuous maintenance problem. The continuous maintenance of the PID-to-locator bindings, causes a particular overhead consisting of a number of expensive transactional administration operations. This constitutes the origin of the performance problem of persistent identification (RQ 1.2).
A reduction of this maintenance overhead (RQ 1.3) can be achieved by two approaches: The first approach is to reduce the number of these expensive transactional administration operations. The second approach in turn, is to improve the performance of the transactional administration operation itself.
In Chapter 3, we focused on the reduction of the number of the administration operations by introducing *Research Data Silo Identifier*s (DSIDs). A DSID allows to specify the data access interface, which is currently supported by the respective research data repository. By this approach, the overhead for administering individual PID records associated with research datasets can become eliminated. This solution is particularly appropriate for research data repositories which are only relying on the locator abstraction function of PIDs, rather than on individual metadata imposed into PID records.
The improvement of the performance of the transactional administration operations itself in turn, is tackled in Chapter 4 and Chapter 5. The focus in these chapters was to accelerate the respective procedures within the naming authorities.

A limitation of our DSID approach, is that it reduces PIDs to a sole redirection mechanism, which is only suitable for a specific group of research data repositories. This group usually only relies on the locator abstraction function of PIDs. For research data repositories, which also rely on the ability of PID records to hold semantic information about the identified research datasets, our proposed DSID approach can only be used in combination of individual PID records.
Another limiting aspect is that this approach requires a modification of the software package of the Handle System. First of all it is necessary to enable to distinct between a real individual Handle Record containing individual information and a dynamically generated Handle Record based on the registered composition rule. We have realized this by extending the native resolution operation of the Handle protocol with a "force" flag, which, when set to true, prevents the execution of the dynamic composition rule. Secondly, it is necessary to extend the pre-defined set of Handle Values by another Handle Value type (`HS_RDS_URL`) to realize the DSID approach without the need for a naming authority (*Local Handle Service* (LHS)).

7.2 Answers to Research Questions Concerning the Performance Analysis of Multi-Tier Internet Systems

Chapter 4 provides the basis for the deduced answers.

These naming authorities form specific, local PID systems, which are often regular multi-tier Internet systems. In addition to the overhead posed by these systems, their offered administration operations are usually accomplished transactionally. For a transactional system it is important to continuously improve its response time especially in the advent of technological progress. Therefore, the goal is to achieve a best possible benefit-effort ratio for the response time improvement. This can be supported by a model, which enables the analysis of effects caused by improvements in individual tiers onto the overall system's response time (RQ 2).
An appropriate model for a multi-tier Internet system (PID system) is queuing network of successively chained simple queuing systems (RQ 2.1), which was also proposed by other authors, in particular by Urgaonkar et al. [79].
Such a queuing network is then analyzed by the established *Mean Value Analysis* (MVA) algorithm. The fundamental performance behavior of the MVA algorithm is a monotonous behavior for the load on specific queues. In a queuing network, which includes a set of dominating service times corresponding to particular queues, with an increasing concurrency level, the weighting factors of these queues are monotonically increasing, while the weighting factors of the remaining queues are decreasing (RQ 2.2). This monotonous behavior is specifically addressed in Section 4.2.3.
An improvement at an individual tier leads to a redistribution of the load within the remaining tiers. The overall response time, therefore, highly depends on the load at the individual tiers. The improvement of the dominating tiers can result in a significant speedup of the response time. In contrast to that, improving a tier with a relative short service time, according to our model, contrarily results for a high concurrency level in no improvement (RQ 2.3).
The effect of an improvement endeavor is described by Theorem 4 in Section 4.3.3. It basically determines the load relation for an individual tier between the old and improved overall system (RQ 2.4).
Since an improvement at an individual tier leads to a redistribution at the remaining tiers, in the worst case, this can cause the capacities at one of the remaining tier to become exceeded. This effect is called *overloading* and is discussed in Section 4.5.5.6 (RQ 2.5). Overloading can contrarily even cause the response time to increase after an improvement effect, which is demonstrated in Figure 4.11.

Considering our contributions in Chapter 4, the main limitation is the restriction on $(T_i \equiv Q_i)$-multi-tier systems. Such multi-tier systems are characterized by the fact, that each tier can be modeled as a single queuing system $(T_i \equiv Q_i)$.
However, in a general multi-tier system, each tier often possess multiple resources, which incorporate their own queuing effects. The load within such a single tier is then often distributed among the internal queues of these resources, which leads our deduced formulas to yield inaccurate values. We have demonstrated this by evaluating the ePIC PID system on two different hosting environments: an ideal environment and a productive environment.
In the ideal environment, the assumption of the ePIC PID system to be a $(T_i \equiv Q_i)$-multi-tier system, proofed to be right, since our formulas provided accurate results. However, in the productive environment, additional internal queuing effects within the tiers have led the load within the overall multi-tier system to be distributed differently than expected. Therefore, in this environment, the values provided by our formulas could only be considered as a heuristic.
A further aspect is that our approach only considers tiers, which only offer a single operation, instead of a set of different operations. In a multi-tier system offering multiple different service operations, these operations, usually also incorporate distinct queuing effects into the overall processing time of an individual tier, which again lead to a different load distribution than expected. This finally, means that also for such systems, our formulas would yield inaccurate results.

7.3 Answers to Research Questions Concerning the High-Performance Persistent Identifier Management Protocol

Our answers to the corresponding research questions are based on Chapter 5.

Handle servers are often subjected to bulky administration requests originating from individual research data repositories. The fundamental performance problem is based on the fact that each of these administration requests, although they can usually be grouped into a common request batch, is processed in a dedicated transactional procedure within the Handle server. This often leads to significant performance losses at the administration of large amounts of PID records (RQ 3.1).
Therefore, to mitigate these performance losses, we have extended the Handle protocol by a bulk registration operation, which can be used to administer multiple Handle Records by a single request, which is then accomplished by a single transactional procedure within the Handle server (RQ 3.2).

The major limitation of this approach is that it requires the modification of the Handle protocol and its reference implementation, which impedes the global adoption. However, since our extension is fully backward-compatible, individual LHSs, equipped without extension, can directly operate within the current global infrastructure of the Handle System.
Another limiting aspect of our proposed bulk registration operation is that it entails the restriction of the maximum supported concurrency level threshold at a Handle server. Otherwise, it is possible that multiple concurrent bulk registration requests containing large amounts of Handle Records can lead to instabilities at the Handle server. Although, in our evaluation we had subjected the Handle server with workloads larger than the current productive load level, future loads can still cause the Handle servers, equipped with our extension, to become instable.

7.4 Answers to Research Questions Concerning the High-Performance Persistent Identifier Resolution

The answers to the corresponding research questions are originated from Chapter 6.

To achieve a high-performant PID resolution we have proposed to utilize the existing and established DNS system (RQ 4).
Since the resolution is accomplished by involving a specific proxy resolver, which is tasked with a node traversal within the hierarchical architecture, the resolution time is often mainly composed of the network latencies between the traversed nodes (including the latency between the requesting application and proxy resolver) (RQ 4.1). In DNS, the main technique to reduce these latencies is to heavily employ caching at the DNS servers. In addition, a requesting application can make use from one the myriads available DNS proxy servers, which ensures a relative low latency to the proxy resolver. However, for PID systems, such as the Handle System, there are only very few proxy resolvers globally available (RQ 4.2), which means that the latency between a requesting application and the PID proxy resolver has usually a contribution in the overall resolution time. This finally leads to an unsatisfactory resolution performance even when caching is employed.
To improve the PID resolution time, we have proposed to resolve PIDs through DNS traffic (RQ 4.3). For the Handle System, this is accomplished by extending the Handle servers with a DNS interface, which enables them to answer on incoming DNS resolution requests containing resolution requests for Handle-PIDs.

A disadvantage of our approach of resolving Handle-PIDs over DNS traffic is the need to run a Handle server, equipped with a DNS interface, in ”Kernel-Mode”, instead of ”User-Mode”. This stems from the fact that DNS traffic is always transferred through port 53 which is obviously below 1024. Remind that all processes listening on a port number lower than 1024 require to be run in ”Kernel-Mode”. This can pose additional security risks, which have to be taken into account. A further limitation is that the combination of {IP-ADDRESS}:{PORT} can only be used by a single process: Since a DNS nameserver always requires the port 53, it would be necessary to assign multiple Ip-addresses to a single host to enable the hosting of multiple Handle servers equipped with a DNS interface. Another hindering aspect is the need for common global zone for the entire Handle System used to delegate domain names containing Handle-PIDs to the servers of the Handle System. Note that have proposed to use the ”handle.pid.” zone. For our evaluation, however, we made use of the ”hx.gwdg.de.” zone, which is a testing sub-zone under the GWDG DNS zone.

7.5 Achieving High-Performance with Persistent Identification

We have proposed several approaches which can significantly improve the performance of PID administration and resolution. To underline our results, for each proposal, we have provided a publicly available implementation into the Handle System and a corresponding comprehensive evaluation.
All these results can be used in combination to achieve an overall high performing PID system based on the Handle System. To demonstrate such a combination, in the following we consider the PID service provided by the ePIC consortium[18]:

Within the ePIC consortium each research data repository is usually assigned a dedicated prefix of the form `21.XXXXX`. For the administration of the corresponding PID records, the consortium

[18]http://www.pidconsortium.eu

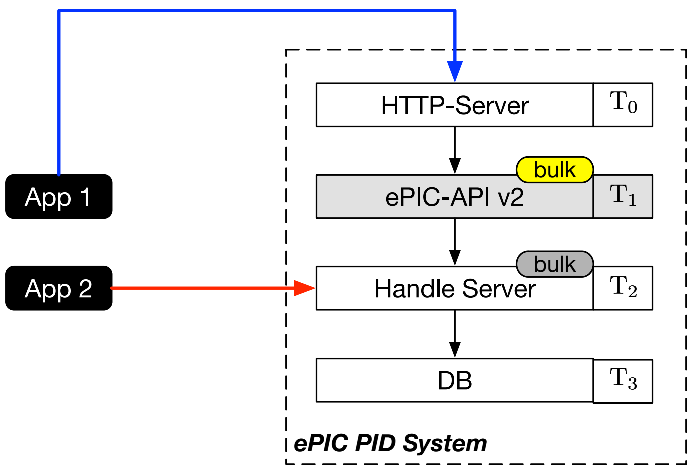


Figure 7.1: PID record administration interfaces of the ePIC PID system: (a) Through the ePIC-API v2 (blue path). (b) Directly through the bulk interface extended Handle server (red path), without involving the ePIC-API v2.

provides several instances of the ePIC PID system. Since this PID system is based on the Handle System, it includes a primary Handle server, which can be also directly used to manage the PID records stored in the underlying database. In Figure 7.1, this is represented by the red path. Although the blue path represents the usual way of involving the ePIC-API v2 for managing PID records associated with ePIC-PIDs, the direct communication (red path) is increasingly used by research data repositories due to the new REST interface of the Handle server (as of version 8.0). However, the blue path is especially useful to ensure the PID records to comply with a predefined structure.

By means of the dedicated prefix, for a research data repository it is appropriate to employ our DSID approach (**first approach**) for the group of datasets, for which only the locator abstraction function is relevant. All occurring changes in the data access syntax of the repository have to be directly adapted at the `HS_NAMESPACE` typed Handle Value in the prefix Handle Record. All PIDs for the remaining datasets, which require individual PID records, can be either be individually administered through the response time improved ePIC PID system (**second approach**) or when the number of individual PID records is large, the new bulk interface (**third approach**) of this PID system can be utilized.
While the first and second approach (remember that the improved version of the ePIC system is already in productive operation) are already applicable, the third approach requires an additional effort: The underlying Handle servers of the ePIC PID system instances can be easily equipped with our publicly available bulk extended version. Hence, the bulk administration through the "red path" (cf. Figure 7.1) requires only a little effort for productive support. However, the "blue path" requires a redesign in the ePIC-API v2 (T_1) to support the new bulk interface of the underlying Handle server (T_2).
On the other, the global resolution time of these PIDs can be significantly improved with our DNS-based resolution approach (**fourth approach**). However, also for this approach there is an additional effort necessary: First of all, our DNS-based Handle-PID resolution requires a common Handle System domain name (*The Handle Zone*), which is appended to the domain name representation of the PIDs. However, it is also possible that the ePIC consortium defines a dedicated DNS zone for its `21.XXXXX`-prefixed Handle-PIDs entailing a respective setup of nameservers

representing the root zone for the prefix `21.XXXXX`. Further information on this can be found in Section 6.5 about the setup of our experimental root zone ”hx.gwdg.de”. Secondly, the underlying Handle servers (T_2) have to be also upgraded with our publicly available special DNS interface, which is relatively easy to achieve. And thirdly, this requires also the requesting applications to issue Handle-PID resolution requests through a specific DNS proxy resolver. Since for DNS there are numerous software libraries available, also the support in the requesting applications is realizable at a maintainable effort. For Java-based applications, we have provided a publicly available implementation of such a DNS software library.

Finally, these four approaches, can enable the ePIC consortium to become a high-performance persistent identification service provider.

Also the *Digital Object Identifier* (DOI) system for example, can employ a set of our proposed solutions, to a achieve a significant performance improvement. Especially our bulk interface extended Handle server can lead to an enourmous speedup of the PID record management. However, as for the ePIC PID system, the new bulk interface requires a redesign in the components, which are communicating with the underlying Handle server.

Chapter 8

Conclusion

In this section, we conclude this thesis by providing a short summary of the derived results. In addition, for the outlook, we provide indications for possible future research aspects.

8.1 Summary

This thesis provides performance improvements for the concept of persistent identification. We have particularly focused on the Handle System, which can be considered as the most established and elaborated PID system.
We have shown that the concept of persistent identification is realized as an overlay network on top of the current Internet. Such a PID system overlay network is composed of naming authorities (local PID systems) used as distributed registries for holding PID records containing PID-to-locator bindings for individual research datasets. The administration of PID records at these naming authorities is usually accomplished by expensive transactional procedures. In addition, the PID concept requires a continuous maintenance of the PID records to keep them valid. This causes a certain overhead affecting the performance.

To improve the performance, our first approach is to reduce the maintenance overhead by reducing the number of the transactional management procedures required for PID record administration. This is achieved by a novel identification concept (*Research Data Silo Identifier* (DSID)), which directly identifies the research data repository instead of its research datasets. This results in the elimination of all the transactional management operations required for administering individual PID records. The DSID is associated with a locator composition rule specifying the currently supported data access syntax of the research data repository. This rule is then used to dynamically compose the corresponding locator of a PID upon a resolution request. However, this concept is usually only suitable for research data repositories which are only relying on the locator abstraction function of PIDs.
For research data repositories which also impose PID records with individual semantic information, the transactional administration operations to maintain the individual PID records are still necessary. Our second approach is therefore focused on the improvement of the response time of PID systems, which usually can be considered as ordinary transactional multi-tier Internet systems.

To improve the response time of PID systems in the advent of fast technological advance, we have provided a mathematical approach, which enables the analysis of the impact of a costly improvement endeavor. Our approach is based on the analysis of the asymptotic behavior of the established *Mean Value Analysis* (MVA) algorithm, which has enabled us to derive estimation

formulas, which can be used to support an improvement endeavor of multi-tier Internet systems by predicting the resulting effects.
In the course of this approach, we have achieved a speedup factor of approximately 1.77 for the response time improvement of the ePIC PID system. This improved version of the ePIC PID system is currently also deployed in the productive PID systems within the ePIC consortium.

In the third approach, we have analyzed the specific workload PID systems are usually subjected to. We have shown that PID systems are often subjected to bulky PID record management requests originating from individual research data repositories. To improve the maintenance overhead with such bulky requests, we have proposed PID systems to offer a bulk management operation, which processes multiple PID records in one transactional management operation. Therefore, we have conceived an extended version of the Handle protocol incorporating such a bulk management operation. With this extension, we have achieved a speedup factor up to 160 for the throughput of the Handle server for PID record administration.

In the last part of this thesis, the focus is devoted on the resolution time improvement of PIDs. Our approach is to enable PID resolution through DNS traffic. The idea is based on the fact there are myriads of publicly available DNS proxy resolvers, whereas the number of PID proxy resolvers is currently very limited, leading globally to long PID resolution times. The result of resolving PID through DNS proxy resolvers is the reduction of the latency between the requesting application and the proxy resolver. This is particularly beneficial for the resolution time of PIDs, which are cached at a particular DNS proxy resolver. For such PIDs, the resolution time basically only consists of the low latency between the requesting application and the DNS proxy resolver.

8.2 Outlook

Another potential of PIDs is to establish a central index enabling the search for research datasets. This means that PIDs systems have the potential to create a ”Google” for research datasets. This aspect was indicated in Section 3.2.7. Within the data-management group of the GWDG, there are currently efforts to provision a graph database with information gathered from the databases of the provided Handle servers. This is realized by a special mirror Handle server, which retrieves Handle Records from different primary Handle servers. The difference between this special and a regular mirror Handle server is based on the attached database: As already mentioned, the Handle server natively only supports the attachment of a Berkeley DB or a SQL database. The special mirror Handle server was been extended to also support a graphs database enabling complex queries on Handle Records. Currently this only considers databases of Handle servers provided by GWDG. Further research is required to establish a global search service on top of various Handle servers provided by various Handle-based PID system providers.

In the context of our performance analysis approach, we have only considered a multi-tier system wherein the individual tiers offer a single operation. Therefore, future work is necessary to investigate systems, which offer a diverse set of operations. Our approach can also be considered as an methodology, which emphasizes on the analysis of the load distribution within a specific group of multi-tier systems. The applicability on a broader group of multi-tier systems requires a respective analysis of the load distribution within such systems. After determining the asymptotic load distribution behavior, it is possible to derive estimation formulas, which can be used in practical situations to understand a system’s behavior and to apply according measures to optimize it. Finally, this means, that our approach provides a fundamental base for further research of multi-tier systems, which is based on a system’s asymptotic load distribution behavior.

Furthermore, our bulk registration operation extension is only focused on the message transmission aspect between a client/application and a Handle server. In the current implementation of our extended Handle server, the Handle Records transmitted with the new bulk operation are processed iteratively and committed in a single transaction within the Handle server. With the original single registration operation, each Handle Record is first processed separately and more importantly also committed individually causing a high overhead for increasing concurrency levels. We assume that further research can reveal an improved processing algorithm for our new bulk operation within the Handle server, which can lead to a further throughput increase. One idea would be that the received batch of Handle Records (via the new bulk interface), are processed in parallel but (again) committed in a single transaction. This is can be especially useful for the two transactional steps, in which first, the replication system is provisioned and second, the Handle Records are written to the attached database (cf. Figure 5.15). The result would be that in each of these two steps, the data insert time would be decreased, while the commit time would basically stay the same as for the initial version of our bulk operation. The ultimate result would be an additional increase of the throughput. In contrast to the original single registration operation, the parallel processing of the batch would not require a synchronization mechanism.
In addition, with the recent versions (>8.0) of the Handle server implementation, it is possible to compose a multi-primary setup consisting of a number $MM > 1$ of primary Handle servers. This opens a further possibility two improve the performance for Handle-PID record administration, especially in conjunction with our bulk operation. This can be realized with a special algorithm, which first, efficiently groups numerous Handle Records into multiple batches of a certain size and secondly, distributes these groups via the new bulk operation onto the multiple primary Handle servers. We expect a throughput speedup of factor MM with this setup.

As already indicated in Section 6.4.4, further research is also required to deduce a more general transformation algorithm which maps general Handle-PID identifier strings into DNS domain names.
Due to the huge number of globally distributed DNS proxy resolvers, it is often reasonable to consider the location of the proxy resolver as be to representative for the requesting application. This can be used to enable an efficient selection among a set of geographically distributed redundant datasets. Since, the number of Handle Records which include multiple locators is increasing, our DNS-based Handle-PID resolution can therefore also be used to determine the ”best” copy of an individual dataset. This requires further elaboration in the software of the Handle servers: A Handle server, receiving a resolution request for a certain Handle-PID at its DNS interface, can use the IP-addresses of the locators of the corresponding Handle Record together with the IP-address of the requestor (DNS proxy resolver) to initiate a special algorithm, which selects the ”best” locator. The simplest form of such a special algorithm can be based on a simple lookup for the originating countries or continents of the IP-addresses at a special IP-address database such as [117]. This would enable PIDs in conjunction with DNS proxy resolvers to provide an efficient data consumption in addition to persistent access and identification.

Bibliography

[1] Across all industries: World Economic Forum, Personal Data: The Emergence of a New Asset Class. `http://www3.weforum.org/docs/WEF_ITTC_PersonalDataNewAsset_Report_2011.pdf`. [Accessed: 2018-06-25].

[2] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[3] Robert Kahn and Robert Wilensky. A Framework for Distributed Digital Object Services. *Int. J. Digit. Libr.*, 6(2):115–123, April 2006.

[4] Diomidis Spinellis. The Decay and Failures of Web References. *Commun. ACM*, 46(1):71–77, January 2003.

[5] Reagan Moore. Towards a Theory of Digital Preservation. *International Journal of Digital Curation*, 3(1):63–75, December 2008.

[6] Ross Wayland Thornton Staples and Sandra Payette. The Fedora Project: An Open-source Digital Object Repository Management System. In *D-Lib Magazine*, volume 9, pages 135–247, Apr 2003.

[7] Bing Zhu, Richard Marciano, Reagan Moore, Laurin Herr, and Jurgen Schulze. Digital repository: preservation environment and policy implementation. *International Journal on Digital Libraries*, 12(1):41–49, Jul 2012.

[8] Yannis Marketakis and Yannis Tzitzikas. Dependency Management for Digital Preservation using Semantic Web Technologies. *International Journal on Digital Libraries*, 10(4):159–177, Dec 2009.

[9] Andreas D. Alexopoulos Theodore S. Papatheodorou Dimitrios A. Koutsomitropoulos, Georgia D. Solomou. Semantic Web Enabled Digital Repositories. *Int. J. Digit. Libr.*, 10(4):179–199, December 2009.

[10] Ronald Jantz and Michael J. Giarlo. Digital Preservation: Architecture and Technology for Trusted Digital Repositories. In *Microform & Imaging Review*, pages 135–247, Mar 2007.

[11] Kenneth Thibodeau. Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years. In *in CLIR and the Library of Congress, The State of Digital preservation: An International Perspective, April 2002. ISBN*, pages 1–887334. Press, 2002.

[12] Wallace Koehler. A longitudinal study of Web pages continued: a consideration of document persistence, 2004.

[13] Steve Lawrence, David M. Pennock, Gary William Flake, Robert Krovetz, Frans M. Coet-

zee, Eric Glover, Finn Arup Nielsen, Andries Kruger, and C. Lee Giles. Persistence of Web References in Scientific Research. *Computer*, 34(2):26–31, February 2001.

[14] Hans-Werner Hilse and Jochen Kothe. Implementing Persistent Identifiers. Overview of concepts, guidelines and recommendations. 2006.

[15] Jonathan D. Wren. URL decay in MEDLINE—a 4-year follow-up study. *Bioinformatics*, 24(11):1381–1385, 2008.

[16] Sam Sun, Larry Lannom, and Brian Boesch. Handle System Overview. `https://tools.ietf.org/html/rfc3650`, 2003. [Accessed: 2018-06-25].

[17] Norman Paskin. The digital object identifier system: digital technology meets content management. *Interlending & Document Supply*, 27(1):13–16, 1999.

[18] Norman Paskin. Digital Object Identifier (DOI®) System. *Encyclopedia of Library and Information Sciences, Third Edition*, 2009.

[19] The DOI Data Model. `https://www.doi.org/doi_handbook/4_Data_Model.html`. [Accessed: 2018-04-10].

[20] John A. Kunze. Towards Electronic Persistence Using ARK Identifiers, ARK motivation and overview, 2003.

[21] Nick Nicholas, Nigel Ward, and Kerry Blinco. Abstract Modelling of Digital Identifiers. `http://www.ariadne.ac.uk/issue62/nicholas-et-al`. [Accessed: 2018-06-25].

[22] Tobias Kuhn and Michel Dumontier. Making Digital Artifacts on the Web Verifiable and Reliable. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2390–2400, Sept 2015.

[23] Juha Hakala et al. Persistent Identifiers - an overview. `http://www.persid.org/downloads/PI-intro-2010-09-22.pdf`. [Accessed: 2018-06-25].

[24] Tim Berners-Lee. Cool URIs don't change. `https://www.w3.org/Provider/Style/URI`, 1998. [Accessed: 2018-06-25].

[25] Emma Tonkin. Persistent Identifiers: Considering the Options. *Ariadne*, (56):8, 2008.

[26] Nicola Nicolson Kevin Richards, Richard White and Richard Pyle. A Beginner's Guide to Persistent Identifiers. `http://www.gbif.org/document/80575`. [Accessed: 2018-06-25].

[27] Ruth E. Duerr, Robert R. Downs, Curt Tilmes, Bruce Barkstrom, W. Christopher Lenhardt, Joseph Glassy, Luis E. Bermudez, and Peter Slaughter. On the utility of identification schemes for digital earth science data: an assessment and recommendations. *Earth Science Informatics*, 4(3):139, Jul 2011.

[28] Herbert Van de Sompel, Robert Sanderson, Harihar Shankar, and Martin Klein. Persistent Identifiers for Scholarly Assets and the Web: The Need for an Unambiguous Mapping. *IJDC*, 9(1), jul 2014.

[29] Mark Nottingham. Web Linking. `https://tools.ietf.org/search/rfc5988`, 2010. [Accessed: 2018-06-25].

[30] Emanuele Bellini, Chiara Cirinnà, Maurizio Lunghi, Ernesto Damiani, and Cristiano Fugazza. Persistent Identifiers distributed system for Cultural Heritage digital objects. *iPRES 2008*, page 242, 2008.

[31] URN Syntax. `https://tools.ietf.org/html/rfc2141`, 1997. [Accessed: 2018-05-26].

[32] Emanuele Bellini, Cinzia Luddi, Chiara Cirinnà, Maurizio Lunghi, Achille Felicetti, Barbara Bazzanella, and Paolo Bouquet. Interoperability Knowledge Base for Persistent Identifiers Interoperability Framework. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, pages 868–875. IEEE, 2012.

[33] Tobias Weigel, Stephan Kindermann, and Michael Lautenschlager. Actionable Persistent Identifier Collections. *Data Science Journal*, 12(0):191–206, 2014.

[34] Barbara Liskov and Stephen Zilles. Programming with Abstract Data Types. In *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*, pages 50–59, New York, NY, USA, 1974. ACM.

[35] EUDAT Project. `https://eudat.eu`. [Accessed: 2018-05-26].

[36] Chi Harold Liu, Bo Yang, and Tiancheng Liu. Efficient naming, addressing and profile services in Internet-of-Things sensory environments. *Ad Hoc Networks*, 18:85 – 101, 2014.

[37] EU-China Joint White Paper on Internet-of-Things Identification. `http://www.miit.gov.cn/n1146312/n1146909/n1146991/n1648536/c3489529/part/3489530.pdf`, 2014. [Accessed: 2018-05-26].

[38] Lukasz Bolikowski, Aleksander Nowinski, and Wojtek Sylwestrzak. A System for Distributed Minting and Management of Persistent Identifiers. *The International Journal of Digital Curation*, 10(1):280–286, 2015.

[39] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system,. `http://bitcoin.org/bitcoin.pdf`. [Accessed: 2018-05-26].

[40] August E Evrard, Christopher Erdmann, Jane Holmquist, James Damon, and Dianne Dietrich. Persistent, Global Identity for Scientists via ORCID. *arXiv preprint arXiv:1502.06274*, 2015.

[41] Oliver Wannenwetsch and Tim A. Majchrzak. On constructing persistent identifiers with persistent resolution targets. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1031–1040, Sept 2016.

[42] BitTorrent. `http://www.bittorrent.org`. [Accessed: 2018-05-26].

[43] Ulrich Schwardmann. Automated schema extraction for PID information types. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3036–3044, Dec 2016.

[44] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys Tutorials*, 16(2):1024–1049, Second 2014.

[45] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-oriented (and Beyond) Network Architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 181–192, New York, NY, USA, 2007. ACM.

[46] Named Data Networking Project. `http://www.named-data.net`. [Accessed: 2018-06-25].

[47] FP7 PURSUIT Project. `http://www.fp7-pursuit.eu/PursuitWeb`. [Accessed: 2018-06-25].

[48] FP7 SAIL Project. `http://www.sail-project.eu`. [Accessed: 2018-06-25].

[49] FP7 COMET Project. `http://www.comet-project.org`. [Accessed: 2018-06-25].

[50] FP7 CONVERGENCE Project. `http://www.ict-convergence.eu`. [Accessed: 2018-06-25].

[51] MobilityFirst Project. `http://mobilityfirst.winlab.rutgers.edu`. [Accessed: 2018-06-25].

[52] Karen R. Sollins. Pervasive Persistent Identification for Information Centric Networking. In *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ICN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[53] Andreas Karakannas and Zhiming Zhao. Information Centric Networking for Delivering Big Data with Persistent Identifiers. `https://doi.org/10.5281/zenodo.889603`. [Accessed: 2018-06-25].

[54] Oliver Schmitt, Tim A Majchrzak, and Sven Bingert. Experimental Realization of a Persistent Identifier Infrastructure Stack for Named Data Networking. In *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pages 33–38. IEEE, 2015.

[55] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS Performance and the Effectiveness of Caching. *IEEE/ACM Trans. Netw.*, 10(5):589–603, October 2002.

[56] Edith Cohen and Haim Kaplan. Proactive Caching of DNS Records: Addressing a Performance Bottleneck. *Comput. Netw.*, 41(6):707–726, April 2003.

[57] Edith Cohen and Haim Kaplan. Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency. In *INFOCOM*, 2000.

[58] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. Authority server selection in DNS caching resolvers. *Computer Communication Review*, 42:80–86, 2012.

[59] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. On the Use of Anycast in DNS. In *Proceedings of 15th International Conference on Computer Communications and Networks*, pages 71–78, Oct 2006.

[60] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twen-*

tieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), volume 3, pages 1801–1810 vol.3, 2001.

[61] Jianping Pan, Y. Thomas Hou, and Bo Li. An overview of DNS-based server selections in content distribution networks. *Computer Networks*, 43(6):695 – 711, 2003.

[62] Janet L. Wiener and Jeffrey F. Naughton. OODB Bulk Loading Revisited: The Partitioned-List Approach. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 30–41, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[63] H. V. Jagadish, P. P. S. Narayan, S. Seshadri, S. Sudarshan, and Rama Kanneganti. Incremental Organization for Data Recording and Warehousing. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 16–25, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[64] Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava, Erik Vee, Ramana Yerneni, and Raghu Ramakrishnan. Efficient Bulk Insertion into a Distributed Ordered Table. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 765–778, New York, NY, USA, 2008. ACM.

[65] Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Vijay Gadepally, Matthew Hubbell, Peter Michaleas, Julie Mullen, Andrew Prout, Albert Reuther, Antonio Rosa, and Charles Yee. Achieving 100, 000, 000 database inserts per second using Accumulo and D4M. *CoRR*, abs/1406.4923, 2014.

[66] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5 edition, 2006.

[67] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991.

[68] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, New York, NY, USA, 2000.

[69] U. Narayan Bhat. *An Introduction to Queueing Theory: Modeling and Analysis in Applications*. Statistics for Industry and Technology. Birkhäuser Boston, 2008.

[70] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[71] Martin Reiser and S. Stephen Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *J. ACM*, 27(2):313–322, April 1980.

[72] Louis P. Slothouber and Ph. D. A Model of Web Server Performance. In *In Proceedings of the Fifth International World Wide Web Conference*, 1996.

[73] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat. Model-based Resource Provisioning in a Web Service Utility. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[74] Christopher Stewart and Kai Shen. Performance Modeling and System Management for Multi-component Online Services. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 71–84, Berkeley, CA, USA, 2005. USENIX Association.

[75] Christopher Stewart, Terence Kelly, Alex Zhang, and Kai Shen. A Dollar from 15 Cents: Cross-platform Management for Internet Services. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 199–212, Berkeley, CA, USA, 2008. USENIX Association.

[76] Abhinav Kamra, Vishal Misra, and Erich M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites. In *Twelfth IEEE International Workshop on Quality of Service, 2004. IWQOS 2004.*, pages 47–56, June 2004.

[77] Xue Liu, Jin Heo, and Lui Sha. Modeling 3-tiered Web applications. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 307–310, Sept 2005.

[78] Abraham Seidmann and Sarit Shalev-Oren. Computerized closed queueing network models of flexible manufacturing systems: A comparative evaluation. *Large Scale Systems*, 12:91–107, 1987.

[79] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An Analytical Model for Multi-tier Internet Services and Its Applications. *SIGMETRICS Perform. Eval. Rev.*, 33(1):291–302, June 2005.

[80] Sandjai Bhulai, Swaminathan Sivasubramanian, Rob Van Der Mei, and Maarten Van Steen. Modeling and Predicting End-to-end Response Times in Multi-tier Internet Applications. In *Proceedings of the 20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks*, ITC20'07, pages 519–532, Berlin, Heidelberg, 2007. Springer-Verlag.

[81] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting Nonstationarity for Performance Prediction. *SIGOPS Oper. Syst. Rev.*, 41.

[82] Jerry A. Rolia and Kenneth C. Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.

[83] John E. Neilson, C. Murray Woodside, Dorina C. Petriu, and Shikharesh Majumdar. Software bottlenecking in client-server systems and rendezvous networks. *IEEE Transactions on Software Engineering*, 21(9):776–782, Sep 1995.

[84] Greg Franks, Dorina Petriu, Murray Woodside, Jing Xu, and Peter Tregunno. Layered Bottlenecks and Their Mitigation. In *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, pages 103–114, Sept 2006.

[85] Minjang Kim, Pranith Kumar, Hyesoon Kim, and Bevin Brett. Predicting Potential Speedup of Serial Code via Lightweight Profiling and Emulations with Memory Performance Model.

[86] Val Donaldson, Francine Berman, and Ramamohan Paturi. Program Speedup in a Heterogeneous Computing Network. *J. Parallel Distrib. Comput.*, 21(3):316–322, June 1994.

[87] Yonathan Bard. Some Extensions to Multiclass Queueing Network Analysis. In *Proceedings of the Third International Symposium on Modelling and Performance Evaluation of*

Computer Systems: Performance of Computer Systems, pages 51–62, Amsterdam, The Netherlands, The Netherlands, 1979. North-Holland Publishing Co.

[88] John Zahorjan, Derek L. Eager, and Hisham M. Sweillam. Accuracy, Speed, and Convergence of Approximate Mean Value Analysis. *Perform. Eval.*, 8(4):255–270, August 1988.

[89] Hai Wang and Kenneth C. Sevcik. Experiments with Improved Approximate Mean Value Analysis Algorithms. In Ramon Puigjaner, Nunzio N. Savino, and Bartomeu Serra, editors, *Computer Performance Evaluation*, pages 280–291, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[90] Paolo Cremonesi, Paul J. Schweitzer, and Giuseppe Serazzi. A unifying framework for the approximate solution of closed multiclass queuing networks. *IEEE Transactions on Computers*, 51(12):1423–1434, Dec 2002.

[91] Dorina Petriu and Murray Woodside. Approximate mean value analysis based on Markov chain aggregation by composition. *Linear Algebra and its Applications*, 386:335 – 358, 2004. Special Issue on the Conference on the Numerical Solution of Markov Chains 2003.

[92] Fatih Berber, Philipp Wieder, and Ramin Yahyapour. A High-Performance Persistent Identification Concept. In *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–10, Aug 2016.

[93] Norman Paskin. *Components of DRM Systems Identification and Metadata*, pages 26–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[94] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[95] Handle System Protocol Specification. `https://www.ietf.org/rfc/rfc3652.txt`, 2003. [Accessed: 2018-06-25].

[96] Sarah Higgins. The DCC Curation Lifecycle Model. In *International Journal of Digital Curation 3.1*, volume 3, pages 134–140, Dec 2008.

[97] ePIC PID System. `http://www.pidconsortium.eu`. [Accessed: 2018-04-10].

[98] The Archival Resource Key. `https://tools.ietf.org/html/draft-kunze-ark-18`. [Accessed: 2018-04-10].

[99] THUMP Protocol. `https://tools.ietf.org/html/draft-kunze-thump`. [Accessed: 2018-04-10].

[100] Easy-Eye-Dee PID Service. `https://ezid.cdlib.org`. [Accessed: 2018-04-10].

[101] James Martin. *Managing the Data Base Environment*. A James Martin book. Pearson Education, Limited, 1983.

[102] RDS Resolver Software Package. `http://hdl.handle.net/11022/0000-0000-9B7C-7`. [Accessed: 2018-02-27].

[103] Fatih Berber and Ramin Yahyapour. Response Time Speedup of Multi-Tier Internet Systems. In *2017 IEEE 36th International Performance Computing and Communications Con-*

ference (IPCCC), pages 1–9, Dec 2017.

[104] Fatih Berber and Ramin Yahyapour. A High-Performance Persistent Identifier Management Protocol. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–10, Aug 2017.

[105] TextGridConsistencyCheck. `http://hdl.handle.net/11022/0000-0007-C2EA-6`. [Accessed: 2018-05-26].

[106] TextGrid Digital Object Management. `http://hdl.handle.net/11022/0000-0007-C6F3-7`. [Accessed: 2018-05-26].

[107] DNS Update Operation. `https://tools.ietf.org/html/rfc213`. [Accessed: 2018-02-26].

[108] Bulk Operation Extended Handle Protocol. `http://hdl.handle.net/11022/0000-0001-8127-1`. [Accessed: 2018-02-27].

[109] Load-Generator for HandleServer. `http://hdl.handle.net/11022/0000-0007-C6FA-0`. [Accessed: 2018-05-16].

[110] Fatih Berber and Ramin Yahyapour. DNS as Resolution Infrastructure for Persistent Identifiers. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1085–1094, Sept 2017.

[111] Ao-Jan Su, David R. Choffnes, Aleksandar Kuzmanovic, and Fabian E. Bustamante. Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections. *IEEE/ACM Transactions on Networking*, 17(6):1752–1765, Dec 2009.

[112] DNS Resource Record Types. `http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml`. [Accessed: 2018-02-26].

[113] Dynamic Delegation Discovery System (DDDS). `https://tools.ietf.org/html/rfc3401`. [Accessed: 2018-02-26].

[114] DNS TXT Resource Record. `https://tools.ietf.org/html/rfc1464`. [Accessed: 2018-02-16].

[115] HandleDNSResolver Source Code. `http://hdl.handle.net/11022/0000-0003-88B2-A`. [Accessed: 2018-02-27].

[116] HandleDNS Algorithm Implementation for Handle Server. `http://hdl.handle.net/11022/0000-0007-C63B-8`. [Accessed: 2018-03-05].

[117] MaxMind: IP intelligence provider. `https://www.maxmind.com`. [Accessed: 2018-06-06].