Universidad Carlos III de Madrid

HIGH-PERFORMANCE AND FAULT-TOLERANT TECHNIQUES FOR MASSIVE DATA DISTRIBUTION IN ONLINE COMMUNITIES

DANIEL HIGUERO ALONSO-MARDONES

Supervisors:

Jesús Carretero Pérez
Florin Isaila

Computer Science Department

Escuela Politécnica Superior

June 2013

HIGH-PERFORMANCE AND FAULT-TOLERANT TECHNIQUES FOR MASSIVE DATA DISTRIBUTION IN ONLINE COMMUNITIES

AUTOR: Daniel Higuero Alonso-Mardones
DIRECTORES: Jesús Carretero Pérez
Florin Isaila

	Nombre y apellidos	Firma
Presidente:		
Vocal:		
Secretario:		

En Leganés, de del 2013

ACKNOWLEDGMENTS

Finalmente, después de bastante más tiempo del que deseaba, me encuentro escribiendo estas líneas que finalizan la escritura de la tesis. Ha sido un proceso arduo y con distintos baches por el camino, pero que por fin tiene una meta a la vista. Me gustaría usar estas líneas para agradecer el apoyo recibido durante la realización de este doctorado.

Quiero dar las gracias a toda mi familia por el apoyo recibido, en especial de mis padres **Marysol** y **Fernando** y mi hermano **Raúl**, que siempre han estado ahí para cualquier cosa. Su apoyo ha sido fundamental para conseguir llegar hasta este punto y es por eso que me gustaría reconocer ese esfuerzo que han puesto durante estos años.

Gracias a **Ana**, cuyo ánimo y apoyo también han sido otro pilar fundamental para poder llegar hasta aquí. He de reconocer que he tenido suerte en encontrarte y por eso espero que sigamos juntos mucho tiempo, comenzando con esa aventura que acabamos de empezar llamada *casa*.

Muchas personas han tenido una influencia significativa en estos últimos años y me han hecho darme cuenta de muchos aspectos de la vida tanto académica como práctica. Primero me gustaría agradecer a mis directores su disposición para llevar este trabajo. A **Jesús**, por haberme dado una oportunidad de incorporarme a este gran grupo de investigación llamado ARCOS y por haberme enseñado e introducido en el mundo de la investigación. A **Florin**, por haberme enseñado todo lo que implica realizar una tesis. También quiero agradecer el apoyo recibido de **todos** los miembros de **ARCOS**, con una especial mención a **Alex** y **Javi (*Doc*)** por esas conversaciones interminables sobre temas variopintos, a **Javi** por ser un grandísimo compañero. Por supuesto, también quiero dedicar unas líneas a **Juanma**. Has sido una persona fundamental durante todo este camino, compañero de fatigas, absurdeces, bugs imposibles y con el que he podido compartir muchas cosas que me han hecho crecer como persona. Creo que sin tu apoyo esta tesis tampoco existiría y por eso te agradezco el haber estado ahí.

Finalmente, también quiero agradecer al resto de personas que de alguna forma u otra me han acompañado en este camino. A la gente de seguridad, en especial a **Chema**, **Lorena** y ***el Maestro***, con los me une una gran amistad. A **Andrea**, **Teresa** y **Mario** por haber compartido también este proceso. A los integrantes de **KerData**, en especial a **Gabriel** por haberme dado la oportunidad de hacer la estancia en Rennes y haberme tratado como un miembro más del grupo. A **Javierito**, por ser un gran amigo y seguir compartiendo vivencias de forma remota. Y por último a todas las personas que me han ayudado durante este camino. Las enumeraciones son peligrosas por el temor de olvidar escribir algún nombre, o crear algún orden implícito, por lo que gracias a todos los demás no nombrados, pero que han estado ahí.

A mi familia
A Ana

We're here to make coffee metal.
We will make everything metal!
Blacker than the blackest black, times infinity!

Nathan Explosion

Never build a dungeon you wouldn't be happy to spend the night in yourself. The world would be a happier place if more people remembered that.

Terry Pratchett - Guards! Guards!

ABSTRACT

The amount of digital information produced and consumed is increasing each day. This rapid growth is motivated by the advances in computing power, hardware technologies, and the popularization of user generated content networks. New hardware is able to process larger quantities of data, which permits to obtain finer results, and as a consequence more data is generated. In this respect, scientific applications have evolved benefiting from the new hardware capabilities. This type of application is characterized by requiring large amounts of information as input, generating a significant amount of intermediate data resulting in large files. This increase not only appears in terms of volume, but also in terms of size, we need to provide methods that permit a efficient and reliable data access mechanism. Producing such a method is a challenging task due to the amount of aspects involved. However, we can leverage the knowledge found in social networks to improve the distribution process. In this respect, the advent of the Web 2.0 has popularized the concept of social network, which provides valuable knowledge about the relationships among users, and the users with the data. However, extracting the knowledge and defining ways to actively use it to increase the performance of a system remains an open research direction.

Additionally, we must also take into account other existing limitations. In particular, the interconnection between different elements of the system is one of the key aspects. The availability of new technologies such as the mass-production of multicore chips, large storage media, better sensors, etc. contributed to the increase of data being produced. However, the underlying interconnection technologies have not improved with the same speed as the others. This leads to a situation where vast amounts of data can be produced and need to be consumed by a large number of geographically distributed users, but the interconnection between both ends does not match the required needs.

In this thesis, we address the problem of efficient and reliable data distribution in a geographically distributed systems. In this respect, we focus on providing a solution that 1) optimizes the use of existing resources, 2) does not requires changes in the underlying interconnection, and 3) provides fault-tolerant capabilities. In order to achieve this objectives, we define a generic data distribution architecture composed of three main components: community detection module, transfer scheduling module, and distribution controller. The community detection module leverages the information found in the social network formed by the users requesting files and produces a set of virtual communities grouping entities with similar interests. The transfer scheduling module permits to produce a plan to efficiently distribute all requested files improving resource utilization. For this purpose, we model the distribution problem using linear programming and offer a method to permit a distributed solving of the problem. Finally, the distribution controller manages the distribution process using the aforementioned schedule, controls the available server infrastructure, and launches new on-demand resources when necessary.

CONTENTS

1 INTRODUCTION 1
1.1 Motivation . . . 2
1.2 Objectives . . . 3
1.3 Structure and contents . . . 3
2 RELATED WORK 5
2.1 Data distribution systems . . . 5
2.1.1 Publish/subscribe architectures . . . 5
2.1.2 Data Grids . . . 8
2.1.3 Content delivery networks . . . 9
2.2 Modeling data distribution systems . . . 12
2.2.1 Approximation algorithms . . . 12
2.2.2 Mathematical models . . . 13
2.2.3 Efficient solving of large models . . . 14
2.3 Efficient data transfers . . . 15
2.4 Social network analysis . . . 19
2.4.1 Statistical study of social networks . . . 19
2.4.2 Community detection in social networks . . . 20
2.5 Summary . . . 21
3 A GENERIC DATA DISTRIBUTION SCENARIO 23
3.1 Decomposition of a data distribution process . . . 26
3.2 Applicability . . . 27
3.3 Summary . . . 28
4 ANALYSIS OF ONLINE COMMUNITIES 31
4.1 Existing methods to detect user communities . . . 32
4.1.1 Detecting online communities using item selection . . . 34
4.1.2 Detecting online communities using community detection algorithms 35
4.2 Iterative weighted community detection algorithm . . . 38
4.3 Evaluation setup . . . 41
4.3.1 Data set . . . 41
4.3.2 Metrics . . . 44
4.4 Experimental evaluation . . . 45
4.4.1 Modularity and clustering coefficient . . . 45
4.4.2 Number of assigned elements to a partition . . . 46
4.4.3 Commonality . . . 48
4.4.4 Effect of the algorithm selection . . . 50
4.4.5 Iterative community detection algorithm . . . 51
4.4.6 Effect of the weighting threshold . . . 54
4.5 Summary . . . 55
5 DATA TRANSFER SCHEDULING 57
5.1 Transfer scheduling problem . . . 58
5.1.1 Baseline model . . . 60

5.1.2 Baseline model discussion . . . 60
5.2 Alternative modeling of the transfer scheduling problem . . . 62
5.2.1 Reformulation as a feasibility problem . . . 62
5.2.2 Approximation heuristic . . . 65
5.2.3 Distributed transfer scheduling . . . 66
5.2.4 Merging partial schedules . . . 67
5.3 Improvements on the formulation . . . 68
5.3.1 Multiplexing server bandwidth . . . 68
5.3.2 Adding user-driven fault-tolerance capabilities with parallel downloads 70
5.4 Evaluation . . . 73
5.4.1 Implementation . . . 74
5.4.2 Computational complexity in practice . . . 74
5.4.3 Sensitivity analysis of scheduling solutions . . . 77
5.4.4 Evaluation of distributed transfer scheduling . . . 78
5.4.5 Energy saving considerations . . . 80
5.4.6 Performance impact of the underlying hardware . . . 80
5.5 Summary . . . 82
6 FAST DATA TRANSFERS 85
6.1 Download engine . . . 88
6.1.1 Design objectives . . . 89
6.1.2 Multiprotocol parallel downloads . . . 90
6.1.3 Internal Architecture . . . 92
6.1.4 Download lifecycle . . . 94
6.1.4.1 Security aspects . . . 97
6.1.4.2 Specifying download sources . . . 98
6.2 Evaluation . . . 99
6.2.1 Parallel downloads . . . 100
6.2.2 Analyzing fault tolerance in the event of mirror failures . . . 103
6.3 Summary . . . 106
7 FINAL REMARKS AND CONCLUSIONS 107
7.1 Contributions . . . 108
7.2 Open research directions . . . 109
7.3 Thesis results . . . 111
7.3.1 Publications . . . 111
7.3.2 Research internship . . . 112
7.3.3 Research grants . . . 112
7.3.4 Related research . . . 113

BIBLIOGRAPHY 115

LIST OF FIGURES

Figure 1 Generic publish/subscribe architecture and operations 6
Figure 2 Generic content delivery network 10
Figure 3 Blocks downloaded from each mirror depending on the co-allocation strategy used during the transfer process. 16
Figure 4 Data distribution graph 23
Figure 5 Determining file-server mapping (left) and user redirection (right) on the server infrastructure. 24
Figure 6 Number of request received by 10% of Wikipedia servers. . . 25
Figure 7 Generic data distribution architecture. 26
Figure 8 Community detection module 31
Figure 9 Explicit social network 32
Figure 10 Inferred social network 33
Figure 11 Detecting communities on declarative social networks using item selection. 34
Figure 12 Application of intersection and Jaccard metrics to produce a condensed graph, with $\mu_I = \mu_J = 0$. 36
Figure 13 Iterative community detection algorithm 40
Figure 14 Correlation between the intersection and Jaccard metrics . . . 43
Figure 15 Normalized standard deviation of the intersection and Jaccard metrics 43
Figure 16 Modularity and clustering coefficient 46
Figure 17 Relationship between number of assigned elements and number of vertices per partition for different graph configurations and algorithms. 47
Figure 18 Influence of the community algorithm in the commonality metrics. 49
Figure 19 Study on the commonality metrics for different community detection algorithms. 49
Figure 20 Evaluation of the iterative algorithm varying the $ratio_{VC}^{entities}$ and the $min_{VC}^{elements}$ parameters. 52
Figure 21 Different measures of the iterative algorithm. 53
Figure 22 Size of the condensed graph attending to the weight threshold value. 54
Figure 23 Number of communities and clustering coefficient attending to the weight threshold value. 55
Figure 24 Generation of a transfer schedule in a data distribution scenario using the requests assigned to virtual communities, and the current file to server allocation on the publisher organizations. 57
Figure 25 Elements involved in data distribution scenario. 59
Figure 26 Baseline model for scheduling transfers. 61

Figure 27 Expressions used in the feasibility problem formulation. . . . 63
Figure 28 Constraints used in the feasibility formulation of the transfer scheduling problem. . . . 64
Figure 29 Effect of reducing the α value in the calculation of T_{max}. . . 65
Figure 30 Distributed solving of the file transfer scheduling problem. . . 66
Figure 31 Merge of two file schedules to obtain a global schedule. 69
Figure 32 Different types of parallel downloads 71
Figure 33 Example schedule produced by the model with parallel downloads. . . . 73
Figure 34 Impact of varying T_{max} on the computational time 76
Figure 35 Impact of varying T_{max} on the schedule makespan 76
Figure 36 Effect of changing α in terms of computation time and schedule makespan. . . . 77
Figure 37 Schedule makespan in time units for the different configurations. . . . 78
Figure 38 Aggregate server miss rate. 79
Figure 39 Ratio of server utilization. 79
Figure 40 Computational time depending on the underlying hardware. . 81
Figure 41 Distribution controller module. 86
Figure 42 Detailed architecture of the Notification Module. 88
Figure 43 Maximum throughput estimation using parallel downloads . 92
Figure 44 Download engine components 93
Figure 45 Download states diagram 95
Figure 46 Secure notifications PKI 97
Figure 47 Parallel download performance using two servers 100
Figure 48 Download speed depending on the number of parallel connections and the chunk size for different servers. . . . 102
Figure 49 Theoretical number of connections depending on the chunk size. . . . 103
Figure 50 Fault tolerance on the event of transient failures 104
Figure 51 Fault tolerance on the event of permanent failures 105
Figure 52 Fault tolerance on the event of permanent failures with a backup server 105

LIST OF TABLES

Table 1	Number of edges in the evaluation graphs.	42
Table 2	Median values of each of the studied metrics per algorithm.	50
Table 3	Baseline model variables and parameters	59
Table 4	Feasibility model variables and parameters	63
Table 5	Example configuration to determine the number of servers (η_{src}) involved in a transfer using the server availability.	71
Table 6	Hardware characteristics	81

1 INTRODUCTION

The amount of digital information produced and consumed is increasing each day. This rapid growth is motivated by the advances in computing power, hardware technologies, and the popularization of user generated content networks. New hardware is able to process larger quantities of data in the same time, which permits to refine the results, and as a consequence more data is generated. In this respect, scientific applications have evolved benefiting from the new hardware capabilities. This type of application is characterized by requiring large amounts of information as input, generating a significant amount of intermediate data resulting in large files.

As the increase not only appears in terms of volume of data, but also in the size of data to be accessed, we need to provide methods that permit an efficient and reliable data access mechanism for the users. Producing such a method is a challenging task due to the amount of aspects that need to be taken into account. However, we can leverage the knowledge found in social networks to improve the distribution process. In this respect, the advent of the Web 2.0 has popularized the concept of social network. The power of a social network does not rely only on the capacity of connecting people, but it also provides valuable knowledge about the relationships between the users, and the users with the data. However, extracting the knowledge and defining ways to actively use it to increase the performance of a system remains an open research direction.

While the existence of social networks can provide benefits to the design of a new architecture, we must also take into account the existing limitations. In particular, the interconnection between different elements of the system is one of the most important aspects to be considered. The availability of new technologies such as the mass-production of multi-core chips, large storage media, better sensors, etc., contributed to the increase of data being produced. However, the underlying interconnection technologies have not improved with the same speed as the others. This leads to a situation where vast amounts of data can be produced and need to be consumed by a large number of geographically distributed users, but the interconnection between both ends does not match the required needs. In particular, we must consider that some of the user are not only interested in accessing some data, but they also require a minimum quality of service.

In this thesis, we address the problem of efficient and reliable data distribution in geographically distributed systems. In this respect, we focus on providing a solution that 1) optimizes the use of existing resources, 2) does not requires changes in the underlying interconnection, and 3) provides fault-tolerant capabilities. The remainder of this chapter is organized as follows. First, in Section 1.1 we describe the motivation behind this work. After that, in Section 1.2 we detail the objectives of this thesis. Finally, Section 1.3 presents the structure of the document.

1.1 MOTIVATION

In order to illustrate the magnitude of the aforementioned problems, we focus on two aspects: the amount and size of data being produced, and the networking capabilities of the users. The data being produced daily in the world infrastructure increases each day. We notice a turning point when the main challenge is not how to generate data (computing capacity is easily available for example by using cloud computing services), but how to provide access and maintain the data.

In this respect, scientific applications represent a suitable candidate for the applications facing the aforementioned challenges. An example of these applications is the LHC or the VLT. The LHC (Large Hadron Collider) project which it is expected to produce [1] around 15 petabytes of in- formation per year. The VLT (Very Large Telescope) survey is expected to produce [2] around 100 Terabytes of data per year. Moreover, the requirement of moving large volumes of data is not only limited to single applications. As an example, Akamai [3] serves 15-30% of the daily Internet traffic reaching more than 10 Terabits per second and over 2 trillion daily Internet requests.

With respect to the networking capabilities, a recent report [107] showed that the global average connection speed increased by 5.0% to 2.9 Mbps, and the global average peak connection speed grew by 4.6% to 16.6 Mbps when compared with previous year. As an example, the average time to transfer a 1 GiB file in this scenario is approximately 47 seconds in the best case. Notice however, that many countries are still below the 1 Mbps level.

We also must point out that these types of problems have been also identified and targeted by different agencies. For example, the European Commission [29, 30] specified in the research lines of the Seventh Framework Program that started after the definition of this thesis:

> "The research is expected to firmly establish digital libraries services as a key component of digital content infrastructures, **allowing content and knowledge to be produced, stored, managed, personalised, transmitted, preserved and used reliably, efficiently, at low cost and according to widely accepted standards**."

> "Digital content is today being produced in quantities that are deeply transforming the enterprise and the creative industries. **Conditions for production and consumption are also rapidly changing as more and more content is produced by users. Organisations, public and private, are faced with maintaining, managing and exploiting increasing amounts of data and knowledge, in environments that are continually changing**."

In the same period, the World Economic Forum [101] also pointed to the need of new data access infrastructures as research and development priorities by includ-

1 Large Hadron Collider expected data per year accessed on October, 2012. http://public.web.cern.ch/public/en/lhc/Computing-en.html

2 Very Large Telescope expected data per year accessed on October, 2012. http://www.eso.org/public/teles-instr/surveytelescopes.html

3 Accessed on May 2013. http://www.akamai.com/html/about/facts_figures.html

ing "Content digitization, digital preservation and access" in the "Digital content technologies" section of the report. Notice that even though the Seventh Framework Program is in its last years, the main objective remains active as of 2013 [31] (*Objective ICT-2013.1.6 Connected and Social Media*):

> "This objective focuses on the **development of advanced digital media access and delivery platforms and related technologies supporting innovation in the digital media sector**."

1.2 OBJECTIVES

The main goal of this thesis is **to define a new efficient and reliable data distribution architecture that leverages the knowledge extracted from the user community to improve the data movement performance**. To provide a detailed view, the main goal of the thesis can be decomposed into the following specific objectives:

OBJ1 **To analyze social networks in order to extract knowledge that can be leveraged in the data distribution process**: Social networks form complex structures that contain vast amounts of valuable information. In this work, we aim to detect the existing communities in a social network and employ that knowledge to define proxy server locations in a data distribution infrastructure.

OBJ2 **To provide a reliable and fast data transfer mechanism**: In a data distribution environment, one of the key aspects is the selection of the underlying protocols and management policies used to access data. In particular, we focus on defining a novel data access mechanism that provides reliable and fast data transfers leveraging the existing server infrastructure.

OBJ3 **To define a novel architecture that exploits social knowledge to improve the data distribution process**: The global objective of providing reliable fast data access and leveraging the social knowledge requires an underlying architecture to support it. In this sense, we target to propose a flexible architecture that is able to capture knowledge from the user communities and use that information to efficiently distribute data to the interested users. This thesis focuses on scientific applications, for which large amounts of data need to be transferred to known users.

1.3 STRUCTURE AND CONTENTS

This document presents the research work realized as part of the PhD dissertation. The remainder of this document is organized as follows:

- **Chapter 2** overviews the related work relevant for the thesis research topics. First, we overview the different types of data distribution systems: publish/subscribe architectures, data grids, and content delivery networks. Second, we present different works related with the modeling aspect of these systems such as approximation algorithms or mathematical models. We complement this information with several methods to efficiently solve those models in real life

scenarios. Afterwards, we explore different methods to provide efficient data transfers. Finally, we describe different approaches to obtaining information from social networks through statistical studies, information propagation analysis, evolution of social networks, and community detection algorithms.

- **Chapter 3** presents the generic data distribution architecture proposed in this thesis and describes the main challenges found in current architectures. First, we introduce a graph representation to model a data distribution problem. After that, we describe the main components of the architecture.

- **Chapter 4** defines two methods to extract information from social networks, and defines a technique to condense the information contained in a data distribution graph in order to be able to apply existing community detection algorithms. We continue by describing some of the most representative community detection algorithms found in the literature and provide an extensive evaluation of the performance. In order to measure the quality of the partitions, we introduce a set of metrics and compare the suitability of each of them in the context of data distribution graphs.

- **Chapter 5** describes our approach to provide efficient data transfer scheduling. We first present the common mathematical approach for solving this type of problems and describe its main limitations. Then, we propose a feasibility reformulation of the initial model that speedups the solving process and permits a distributed solving using the map-reduce paradigm. The distributed model is then evaluated using a realistic scenario and an extensive study of the quality of the resulting schedules is provided.

- **Chapter 6** introduces our reliable fast data transfer mechanism. We describe the main components involved and focus on the proposed download engine. Our engine permits the concurrent use of multiple protocols in order to transfer different pieces of data at the same time. The chapter presents different aspects of the engine concentrating on how it communicates and interacts with other components, how it manages the download lifecycle, how it manages the security aspects, and how it provides reliability and fault-tolerance capabilities. Finally, we provide an evaluation of the engine that demonstrates its main features in terms of performance and fault-tolerant capabilities.

- **Chapter 7** presents a summary of the contributions of the thesis, the main conclusions resulting from this work and future research lines opened by this PhD dissertation.

2

RELATED WORK

This chapter overviews the state-of-the-art and presents the basic concepts used throughout the thesis. The contents are organized into four main sections. First, an overview of different types of data distribution systems is presented. Second, we explore different approaches that had been proposed in the literature to model this type of systems. Third, we describe techniques that provide fast data transfers in a distribution infrastructure. Finally, we overview different aspects and characteristics of online social networks that can provide valuable knowledge in the modeling process.

2.1 DATA DISTRIBUTION SYSTEMS

With the increase in content in the Web 2.0 era, the need for distributing data to vast amounts of users is a basic requirement of many applications. Several generic architectures for this purpose exist in the literature, and it is a fundamental decision from the content provider point of view to choose the underlying architecture based on its specific requirements.

We focus on three major types of data distribution systems based on how data is discovered, stored and accessed. The following subsections describe the inner workings of publish/subscribe architectures, data grids, and content delivery networks.

2.1.1 *Publish/subscribe architectures*

Publish/subscribe architectures are a common solution for propagating events between loosely coupled entities on a distributed environment. Particularly, this type of architecture is especially useful when the content is created by a small fraction of entities and it needs to be distributed to a large number of users (e.g., producer-consumer problems).

The generic publish/subscribe [41] architecture is composed of three types of entities. Users or components interested in a particular type of data receive the name of *consumers*. Analogously, users or components that generate data in the system receive the name of *publishers*. The information to be distributed is denoted by the term *event*. To determine which users must receive a particular type of event, they are required to establish a *subscription* prior receiving any *notification* from the publishers.

New events created by the publishers are sent to the *event service*, which represents the mediation point between publishers and subscribers. The event service is the component of the system in charge of storing subscriptions and performing the matching against new events in order to determine which consumers will receive the notification.

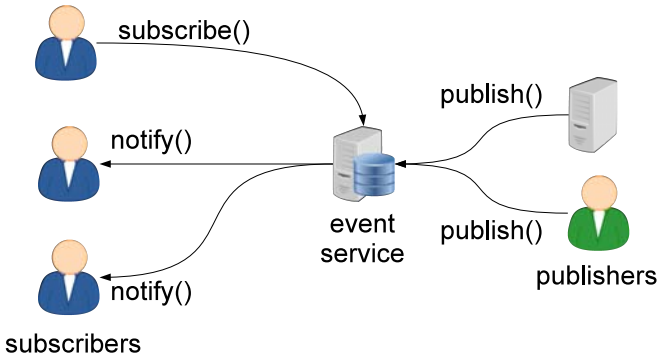


Figure 1: Generic publish/subscribe architecture and operations

Figure 1 shows a generic architecture for a publish/subscribe system. On the right side, the publishers generate new events by invoking the *publish(event)* operation on the *event service*. On the left side, a user is interested in a new type of events and executes the operation *subscribe(event)*. As described before, the publish operation triggers the matching mechanism in the event service and as a result a callback to the *notify()* operation in the interested users is executed. The main characteristics offered by the publish/subscribe paradigm are:

- **Space decoupling**: The publish/subscribe architecture introduces an intermediate point in the communication paradigm. The event service allows the publishers to avoid storing information about their subscribers, which also anonymizes the subscribers from the publishers.
- **Time decoupling**: The event service manages a set of event queues that allow the publishers to asynchronously send notifications to subscribers. This characteristic makes not necessary for both entities to be online at the same time in order to send and receive notifications. The use of queues allows subscribers to retrieve pending events when they come online.
- **Synchronization decoupling**: Related with the time decoupling characteristic, the asynchronous behavior introduces another desired functionality. As the events are sent first from the publishers to the intermediate event service and then transferred to subscribers, publishers are not blocked by the publish operation as the event is asynchronously notify to the interested users.

Publish/subscribe systems can be categorized [70, 41] attending to two characteristics: how subscriptions are managed, and how the underlying architecture is deployed. Users of publish/subscribe systems need to subscribe to the type of content they are interested in. Depending on the level of detail of the subscription, and subsequently, the internal matching algorithm on the event service, we distinguish between topic-based, content-based or type-based systems.

- **Topic-based**: This type of publish/subscribe systems is based on the notion of subject, category or topic. Each event published in the system is associated with a topic that represents the nature of the information contained in the

event. As using a flat categorization of events complicates the management of information, most topic-based systems support topic hierarchies to facilitate fine grained subscriptions.

- **Content-based**: As an evolution of topic-based systems, the content-based architectures offer users the possibility of subscribing to events with a particular content. In this type of publish/subscribe systems, the event service matches users subscriptions against the content of each event permitting a finer capability of filtering out undesired messages.
- **Type-based**: As the complexity of the distributed events grows, new mechanisms are necessary to offer a usable subscription system. Type-based subscriptions appear as an evolution of content-based matching algorithms. Based on Object Oriented Programming, events in the system are associated with classes or types, each having its own attributes. From the subscriber point of view, it is possible to subscribe to particular types of events, with a specific content.

The underlying deployment architecture is another differentiating characteristic of a publish/subscribe system. In this aspect, we can observe the evolution from the client-server approach to a distributed deployment of the involved components. Considering both the type of subscription employed, and the underlying architecture, we overview the relevant research in this direction.

Topic-based publish/subscribe systems offer a simple, yet powerful approach for a variety of use cases. Corona [92] proposes to use a topic-based system to address the problem of unnecessary polling on RSS web subscriptions. Users subscribe to a particular URL, and the distributed system built on top of the Pastry [98] distributed hash table computes the optimal way to poll for updates. In this aspect, the system optimizes the tradeoff between allocated bandwidth to a particular web and the resulting update latency. Scribe [22] as Corona also uses Pastry as the underlying peer-to-peer network. It defines a set of groups, each of them containing a subset of subscribed users. A multicast tree is then built to determine the notification process. Similar to Scribe, Bayeux [134] is built on top of Tapestry [132] and also defines a set of groups to manage the subscription process. Both works differ in the way the multicast tree is built, on Bayeaux all nodes have to maintain a large amount of information, as membership information is stored along the whole network. On contrast, Scribe distributes this type of information on the network.

Content-based systems provide additional capabilities to users to determine which content to subscribe to. While it is possible to introduce some type of hierarchy in topic-based system, their management becomes too complex and does not offer enough flexibility. Different proposals have been made to offer efficient content-based systems.

Terpstra et al. [106] propose a content-based publish/subscribe system based on Rebeca [78] and built on top of the Chord [103] distributed hash table. In this work, authors argue that using a tree structure for notifying users may introduce single points of failure, and therefore propose a routing algorithm built on top of a peer-to-peer graph. Tam et al. [105] propose a distributed content-based system built on top of Scribe and implemented on Pastry. The system requires an underlying schema to help during the matching process permitting to automatically identify general topics from the content of the subscriptions.

Triantafillou and Aekaterinidis [113] define a publish/subscribe system that leverages the advantages of the Chord DHT in order to support range predicates in the subscriptions. This type of content subscription benefits from the DHT overlay as queries targeting ranges in particular attributes will only involve querying contiguous nodes in the DHT. Sub-2-Sub [120] presents a collaborative self-organizing publish/subscribe system deployed over an unstructured overlay network. The author motivates the selection of an unstructured network due to the fact that popular content may overload some peers in the usual structured network approach. It uses an epidemic-based algorithm to assure that notifications reach the subscribers, supporting range predicates. Mercury [16] addresses the problem of communication among the entities involved in an online game. The authors propose a routing algorithm based on the incorporation of attribute hubs. Each hub is composed of several nodes, each of them being responsible for a range of the attribute assigned to that hub. Upon receiving a new message, its attributes are analyzed and route through the affected hubs until the notification reaches the interested subscribers.

The evolution of publish/subscribe systems and the incremental complexity of the subscriptions lead to the appearance of type-based systems. While they require the definition of a class or schema prior using the system, several applications already have this categorizations and can benefit from them. Eugster [40] and Pietzuch [87] propose slightly different approaches to build these systems. While the work presented by Eugster is a high-level definition on how to capture the semantics of the subscriptions, Hermes [87] offers a distributed implementation on top a peer-to-peer routing overlay.

Additionally, other works related with publish/subscribe systems have explored other areas of interest. Liu et al. [69] studied the characteristics of 100,000 RSS feeds over a period of 45 days, providing recommendations on how to improve current publish/subscribe implementations of this type of applications. Huang and Garcia-Molina [58] research the problem of implementing publish/subscribe systems in mobile environments. This type of environments introduce new challenges as events can be generated by moving sensors or users, and the notifications require extra fault-tolerance techniques as the subscribers may also appear in different parts of the network.

2.1.2 *Data Grids*

The emergence of the grid computing [49, 48] brought new challenges related with the way we store and access data. Different middlewares were defined and implemented capturing the particularities of the underlying associated projects. In this aspect, we can name the Globus Toolkit [50] [1] and gLite [2] as the most adopted grid middlewares.

With the appearance of the computing grid, the notion of data grid [26] was born. The possibility of using a distributed computing platform requires the definition of basic services that establish how data will be accessed and move on that platform. The modular design of the grid permits to build the required services [5] leveraging the existing functionalities provided by the underlying toolkit. In particular, most of

1 www.globus.org/toolkit/
2 glite.cern.ch

the effort is centered on how replication should be done in order to provide fault-tolerance and parallelization functionalities.

Several studies [34, 60, 94] explore the benefits and tradeoffs of using different replication algorithms when presented with different types of workloads. In this respect, the use of static algorithms to define the required replication levels has been demonstrated to provide inaccurate results due to the dynamicity of the usage patterns over time. As a consequence, a set of dynamic algorithms has been defined in order to adapt the replication level to the incoming demand. As an extension, Abdullah et al. [1] study how replication can be leveraged to permit the parallel processing of data set that can be split due to the absent of dependencies in the processing.

Regarding the required functionalities of a data grid infrastructure, other studies have focused on the consistency and security aspects. Domenici et al. [35] proposed a replica consistency service to address the challenges of maintain data and metadata consistency on this type of distributed infrastructure. Tu el al. [114] explore secure data partitioning and replication on a data grid, presenting two heuristics to solve the graph representation of the problem.

Even though, Cloud computing has emerged in the recent years, several grid computing infrastructures are still in use. As an example, the Teragrid [68] infrastructure that appeared in 2001, has evolved into the XSEDE (Extreme Science and Engineering Discovery Environment) project [3] funded initially for five years ($121 million budget) by the National Science Foundation.

2.1.3 *Content delivery networks*

Content Delivery Networks (CDN) [117] are used in the current Web environment to optimize the distribution of information to final users. These systems can be seen as an evolution of distributed web server infrastructures [20]. The notion of CDN is related with the concepts of edge server, caching and replication. An edge server designates a server that works as a proxy for the original server but which is located near the final user (consumer of information), rather than near the source of information. Content is replicated/distributed among these edge servers and cached for a period of time to improve the access experience of the final users.

Figure 2 illustrates the generic architecture of a CDN. In the center of the architecture, the origin servers represent the source of information. Between the users (consumers) and the origin servers, several edge servers are allocated. When a user request appears in the system, it will be redirected to an edge server. If that server does not have a cached copy of the requested data, it will retrieve first a copy from the original servers, and the forward the information to the final user.

Content delivery networks offer two primary benefits. First, the utilization of a CDN to distribute content reduces network utilization by means of caching content retrieved from the original servers. The caching mechanism reduces the latency perceived by the users, as the content will be delivered from the cache in most cases. Second, the use of a caching mechanism improves system reliability. In the event of a source server failure, the users may not perceive the error as their content is being served from the edge servers.

3 https://www.xsede.org/home

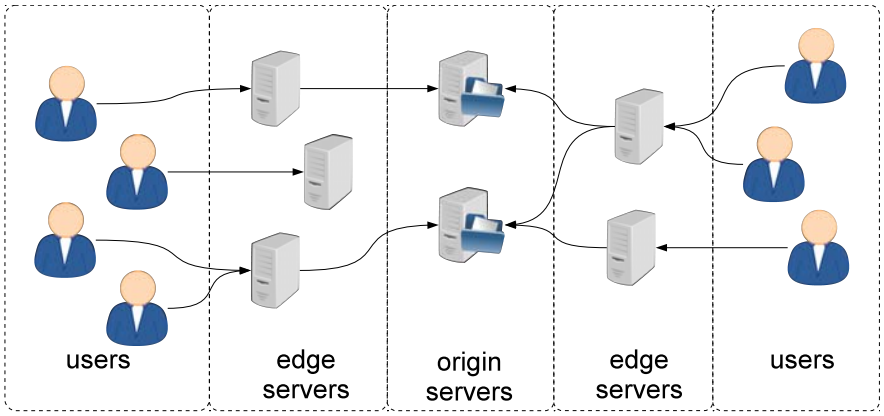


Figure 2: Generic content delivery network

However, using a CDN as a part of our architecture imposes a series of challenges [86] that need to be addressed in order to obtain an optimal performance. In this respect, a CDN can be considered as an external platform that content owners may use in order to speedup the distribution of their contents. In order to adequately leverage of the properties of the CDN several aspects must be considered:

- **CDN composition**: Analyzing the structural attributes of a CDN reveals the components involved. This information helps determine how different from the generic architecture is a particular CDN, and therefore, whether it must be used in a particular way in order to achieve the best performance. The most important aspects in the CDN composition are: (1) whether an overlay or network approach is used as the CDN organization, (2) which are the type of servers involved, (3) what are the relationships between the existing components, and (4) which type of content can we distribute on that CDN: static, dynamic and/or streaming content.

- **Content distribution and management**: The structure of the CDN is also related with the problem of content distribution and management. In this aspect, it is necessary to decide how caches and replicas will be managed, and how this content will be stored in the CDN.

- **Request-routing**: As a distributed system, selecting the appropriate server is an important decision as it will affect the latency perceived by the requesting user. Different policies can be used to determine how user requests will be redirected on the CDN infrastructure.

- **Performance measurement**: Due to the complexity of this type of infrastructure and the fact that their components are usually geographically distributed all over the world it is necessary to determine how to monitor the CDN performance. Monitoring this type of infrastructures provides an updated vision of the behavior of the system and can be used in different situations: from checking that Service Level Agreements are being fulfilled, to detecting emerging hot spots around popular content.

The basic architecture of a CDN (Figure 2) introduces additional problems in user generated content networks such as YouTube or Flickr. These types of networks need the CDN architecture to distribute content, but also need to provide users with the ability to upload new content. In these cases, the typical solution considers the users as both consumers and producers of information. Content is distributed to the community using the CDN infrastructure, while the users upload new content to the origin servers. It is important to highlight that users cannot upload content directly to the CDN, as it is only a cache for the content stored in the origin servers.

The use of CDNs has increased rapidly in the recent years following the increase in volume and size of multimedia content. Major Web 2.0 sites, are using one or several CDNs to distribute their content to a variety of geographically distributed users. Some major sites, such as YouTube [111], opt to develop and maintain their own CDNs. However, due to the large infrastructure costs, and the knowledge required to successfully operate this infrastructures, a large amount of sites opt for third-party owned CDNs such as: Akamai [4], Limelight [5], and Level3 [6].

Akamai [83] started as an academic project and evolved into one of the most important CDN infrastructures. The Akamai infrastructure [7] is composed of more than 105,000 servers deployed on 78 countries. The architecture is based on the idea of deploying part of the content distribution infrastructure inside ISP POPs (Point of presence). With this approach the edge servers are moved towards the end user. The main disadvantages derive from the highly distributed design of the final system. In particular, maintenance and management of the geographically distributed servers become very challenging. From the point of view of the user requests, Akamai is based on DNS redirection. On a user request, the original servers answer the DNS request with a canonical name for an Akamai server. After solving the Akamai server in the CDN DNS infrastructure, the user gets several IPs of edge servers where the content can be retrieved.

Limeligth [57] is another important commercial CDN, based on a different approach than Akamai. The architecture is not based on moving edge servers to the user ISPs as Akamai, but on building data centers at different locations and connecting these data centers to near ISPs. The main advantage of this design approach is that the maintenance and management of the edge server infrastructure is easier as the data centers are less geographically distributed. The possible disadvantage is an increase in the delay depending on the user ISP connectivity with the Limelight servers.

Level 3 [123] originally started as a Tier-1 carrier but moved to the CDN business in 2006. It leverages the IP transport network and global peering relationships already owned. It is characterized by the ability of managing live broadcasting events due to the large existing infrastructure.

Understanding how large web 2.0 sites use CDNs successfully is of interest to the research community. Even though internal details of how they use a CDN infrastructure remain unknown, several studies reveal interesting aspects. Adhikari et al. [2] measure how Hulu [8] uses the aforementioned CDNs. The authors find that Hulu fre-

4 www.akamai.com
5 www.limelight.com
6 www.level3.com
7 Accessed on August, 2012, www.akamai.com/html/about/facts_figures.html
8 www.hulu.com

quently changes the CDN that will serve a particular user, but it maintains the same CDN once it has been selected during the streaming of a single movie. The study also shows that the selection of the CDN is based on a predefined ratio, not by dynamically selecting the best one. In [3] authors show that Netflix [9] also uses the same three major CDNs. However, the selection of the CDN is associated with the user accounts. Torres et al. [111] study how YouTube [10] uses its own CDN. The authors find that while requests are typically redirected to the most appropriate edge servers, this is not the case for 10% of the analyzed requests. The authors identify several reasons for this type of decisions: load balancing, variations across DNS servers, hot spot avoidance on popular content, and limited availability of unpopular content.

2.2 MODELING DATA DISTRIBUTION SYSTEMS

The need of modeling data distribution systems has followed their increase in popularity. Obtaining an accurate model of such a complex distributed system provides a deeper understanding of its inner workings. The knowledge can be leveraged for different objectives: data transfer scheduling, bandwidth optimization, etc. In this thesis, we focus on methods that can be used to produce minimum transfer schedules on distributed systems. Due to the complexity of defining this type of models, approximation algorithms are often used to obtain approximated solutions.

This section provides an overview of the different methods taken to model a distribution system, from formal linear programming approaches to greedy algorithms, considering both formal and approximation approaches. It is important to notice that the data distribution problem shares some common aspects with other problems (e.g., the replica placement problem), making this type of research of interest for the current thesis.

2.2.1 *Approximation algorithms*

Several works in the literature tackle the problem using approximation algorithms. In this sense, different works explore the use of heuristics, iterative algorithms, greedy approximations and many other types of heuristics.

Balman [10] proposes a methodology for provisioning end-to-end transfers. The work uses a graph representation of the problem and proposes a set of heuristics allowing the user to specify the resource and time requirements of the transfers. Carofiglio et al. [21] study the data transfer problem in content-centric networks. The authors characterize the miss rate and propose an analytical expression for the estimated throughput when transferring data between different sites.

In [121] the authors explore the problem of optimizing latency and throughput on workflows executed in cluster environments. A heuristic algorithm is proposed to minimize the latency maintaining a minimum throughput requirement. Yuan et al. [128] analyze data placement strategies in the context of cloud workflows. The authors propose an algorithm to determine how to move large data sets between different cloud instances.

9 www.netflix.com
10 www.youtube.com

Kllapi et al. [63] propose a schedule optimization algorithm for data processing workflows executed on cloud environments. In this work, the authors focus on two different objectives: minimizing the schedule makespan under a constrained budget, and minimize the cost under a given deadline. Henzinger et al. [54] define a declarative language that facilitates the task of expressing the job requirements and propose a static scheduling algorithm with different user objectives for executing tasks on cloud environments. Pandey and Buyya [85] study the scheduling of data intensive workflows on a distributed environment on the presence of multiple data replicas. The authors propose a heuristic to exploit the available parallelism accessing the different replicas of the required data.

2.2.2 *Mathematical models*

The development of mathematical models represents another approach to solving complex scheduling problems. To facilitate the construction of new models, the usual approach tries to find a base well know problem [88], on which the future model is built on top, adding the requirements of the specific scenario. The model specification can be represented as different types of mathematical programming depending on the type of variables involved and the linearity of the problem. In this respect, we find two main approaches: mixed integer programming [99], and constraint programming [96].

Using constraint programming, Zerola et al. [129, 131, 130] research the problem of moving data between different sites in a grid environment with a minimum makespan. The authors based their model on a variation of the Job Shop Scheduling problem with a graph representation. By using constraint programming and different heuristic to speed up the solving process, the authors obtain a viable method to schedule data transfers compared with a peer-to-peer approach.

Barták [13] propose a model for the path placement problem, where the objective is to determine how a particular demand on one node of the network will be served by the available nodes. The authors propose a flexible constraint programming model that captures the specifics of the problem, and is open for future extensions. Holub et al. [56] define a constraint programming model for data transfers using a tree placement representation. The model is complemented with several heuristics, and results on small test cases are provided showing its feasibility.

Nguyen [82] describes a content distribution network provisioning framework that takes into account the available resources and specific application requirements. It permits to define the optimal structure of the CDN to cope with a specific demand. The author proposes a heuristic based on decomposing the model in two complementary models to speed-up the process. Using this approach different models are defined attending to the type of content to be served: static HTML, multimedia content, live streaming, etc.

Sun et al. [104] describe an optimal replica placement strategy for content distribution networks. In the paper, the authors propose the use of a Multiple Minimum Cost Flow model with server storage constraints to capture the complexity of the problem. The model is converted to an equivalent Mixed Integer Programming one, and the simulations confirm the feasibility of the solution.

2.2.3 *Efficient solving of large models*

A formal model requires a solver software in order to produce a solution given a set of parameters depending on the selected scenario. As models become more complex in order to capture as much of the reality as possible, the complexity of the model increases accordingly.

This type of software is usually based on known algorithms (e.g., simplex), but the quality of the implementation with the added heuristics is the factor that differentiates one solver from the other. This difference is usually reflected in the memory and CPU consumption, as well as time required to produce a solution. The most widely used solvers are:

- **GLPK**[11]: The GLPK (GNU Linear Programming Kit) supports mixed integer linear programming models written in MathProg [73]. It uses a revised simplex algorithm and the primal-dual interior point method for non-integer problems, and the branch-and-cut for integer problems.
- **LP_SOLVE**[12]: This solver provides another open source implementation of the revised simplex method and the branch-and-bound method for integer problems.
- **IBM ILOG Suite**[13]: The IBM ILOG Suite provides the commercial solver CPLEX. The solver supports large optimizations problems and provides a threaded implementation to speedup the process in a large variety of platforms (AIX, HP-UX, Linux, Solaris, Mac OS, etc.). The last versions of the optimization suite include CPLEX CP, which can be used to solve constraint programming problems.
- **Gurobi Optimizer**[14]: The Gurobi Optimizer is a commercial solver that offers similar features to CPLEX. Its main capability is the use of extensive parallel solving algorithms in multithreaded environments. Based on this product, the company offers Gurobi Cloud as a ready-to-run solution for launching multiple solvers on Amazon EC2.

Regarding the performance of the different solvers, it is difficult to find a complete up-to-date study. Meindl and Templ [75] test the performance on a single thread configuration of GLPK, LP_SOLVE, CPLEX, and GUROBI. In the test they found a significant difference in terms of execution time and number of completed instances, with times of 22.11, 19.40, 1.45, and 1; and completion success of 3.45%, 5.75%, 84%, and 88% respectively. Koch et al. [64] provide an exhaustive study [15] of the performance of different solvers when presented with a variety of problems. The authors evaluate different solvers using the default configuration varying the number of available threads and memory. Two main conclusions are drawn from the results. First, the open source solvers tested (GLPK and LP_SOLVE) were not able to solve more

11 www.gnu.org/software/glpk/
12 lpsolve.sourceforge.net/5.5/
13 www.ibm.com/software/websphere/ilog/
14 www.gurobi.com/products/gurobi-optimizer/
15 Updated and extended versions of the test available at: plato.asu.edu/ftp/milpc.html

than 20% of the instances within the time limit and there is no multithreaded version available. Second, there is no clear winner, as results vary from test to test. Solvers that are much faster in some particular instances, become the slowest ones in others. These results may be explained by the use of the default configurations, as each solver offers a significant number of parameters that can be tweaked to adapt its performance.

Apart from the commercial solvers, other works have developed specific parallel solvers. Eleyat and Natvig [39] propose an optimized parallel solver leveraging the special capabilities of a Cell processor. Smelyanskiy et al. [100] present an optimization of the interior-point methods for large-scale chip multiprocessor systems. Badia et al. [9] implement a parallel solver specifically designed for large linear systems. The main disadvantage of these approaches is the lack of generalization in terms of problems that can be solved and the utilization of specific hardware.

Some of solvers presented in the previous paragraphs offer parallel solving at thread level. However none of them offers distributed deployments of the solvers. This is explained due to the inner workings of the solving method. Independently of the selected algorithm, solving a linear programming problem usually involves building a centralized structure such a tree, to represent what are the branches (the paths) that have been explored. By pruning branches the solver marks which paths do not lead to a feasible solution. As the centralized structure maintains the status of the solving process, it becomes difficult to efficiently distribute this structure or manage the concurrent access to it. In this aspect, Budiu et al. [19] propose an alternative implementation of a distributed branch-and-bound solver. Even though the previous works propose parallel distributed solvers, in practice, they are tailored to specific problems and not general enough for real uses.

2.3 EFFICIENT DATA TRANSFERS

Different solutions have been proposed during recent years to improve the transfer process of large volumes of data. As data transfers are in many cases executed in a client-server architecture, one of the first optimizations involves the use of mirrors. A mirror can be defined as one server where a subset of files stored in the original server is replicated in order to have an additional site to retrieve these files. The utilization of mirrored architectures has been explored in different studies, which proposed various co-allocation strategies and policies to retrieve different chunks of data from a set of available mirrors.

Figure 3 shows a graphical representation of the main co-allocation strategies that can be used in mirrored environments. To help demonstrate the differences, the figure assumes that the available bandwidth of mirror 1 is larger than of mirror 2; additionally, mirror 2 is supposed to have a significant larger bandwidth than mirror 3. The different policies are:

- **Brute-force** [118]: This scheme divides the file size by the number of available mirrors creating a set of chunks. The transfer of each chunk is assigned to one mirror and only one transfer is done from any mirror.
- **History-based** [118]: This scheme gathers information from past transfers on each mirror. Using the predicted available bandwidth, the file is divided into

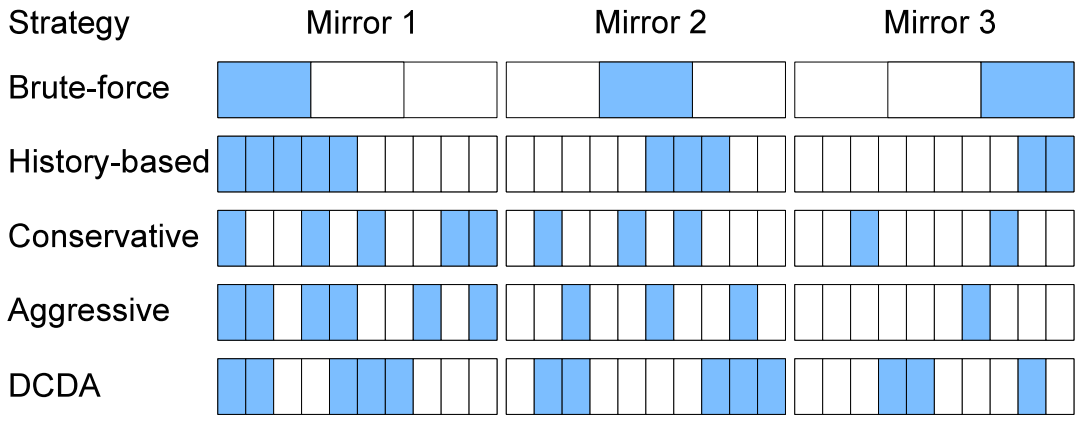


Figure 3: Blocks downloaded from each mirror depending on the co-allocation strategy used during the transfer process.

a number of chunks and they are assigned to mirrors proportionally to their expected bandwidth.

- **Conservative** [118]: With this approach, the amount of chunks transferred from each mirror is decided dynamically. In order to do that, the file is divided into a number of chunks larger than the amount of mirrors. The assignation of which mirror will be used to transfer a chunk is based on mirror availability. In this scenario, faster and well connected servers will transfer a large portion of the file.

- **Aggressive** [118]: The aggressive approach uses information gathered during the download process to assign more chunks to faster mirrors while reducing the number of assigned chunks to slow mirrors.

- **DCDA** [17]: Similar to the aggressive scheme, the DCDA (Dynamic Co-allocation with duplicate assignments) uses information gathered during the transfers to increase the utilization of faster mirrors. The main characteristic of this approach is the capability of assigning the transfer of the same chunk to different mirrors. Using this feature, instead of predicting the available bandwidth, the transfer starts from both mirrors and the block is completed from the faster one. The main disadvantage of this approach is the over utilization of resources as some mirrors will be using a fraction of the available bandwidth, without completing a chunk.

The main challenge of co-allocation strategies is deciding which strategy is the most suitable for a particular scenario. Using or extending the previous co-allocation strategies, different studies offer new solutions for the replica management and selection problem. Based on the DCDA strategy, an anticipative recursively adjusting mechanism for selecting replica servers is defined in [125]. The system stores finish rates of previous transfers and uses that information along with the expected mirror load to select the optimal replica server. While the previous approach is centered on selecting different replicas, other solutions are centered on providing mechanisms for partial replication. A file can be completely replicated among a set of mirrors or

it could be split and partially replicated. In this respect, Chang and Chen [25] implement a block mapping procedure that allows servers to provide partially replicated content to requesting clients.

Other solutions are oriented to grid environments, such as rFTP or Reptor that include co-allocation strategies. rFTP [43] is an improvement of GridFTP that uses multiple sources offering fault tolerance at download level. Reptor [65] on the other hand is a replica management grid middleware. Both applications coexist with the Globus Toolkit offering different levels of integration.

Independently of the co-allocation strategy selected for a particular environment, the selection of the underlying transfer protocol is a key decision as it will affect the resulting performance. Different transfer protocols can be used to perform a download. However, depending on the application scenario, some protocols may outperform others. The most used protocols in practice are:

- **FTP, FTPS, SFTP**: File Transfer Protocol (FTP) was defined in the RFC 959 [90] and it is one of the oldest protocols still being used without any type of optimization from the original RFC. The protocol uses two different connections: control and data. The control connection is the channel used for the exchange of commands and replies between the client and the server. The data connection is opened for the transfer of information between both ends. The main problem found in FTP is the lack of security, with user data being transferred without any type of protection. To address the security concerns, two protocols were proposed: FTPS [46] and SFTP [51]. FTPS also known as FTP Secure adds support for the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols. The protocol adds new commands that allows the client and the server to negotiate and TLS or SSL channel. SFTP or SSH File Transfer Procotol is another approach to securing a file transference over FTP. Instead of negotiating a TLS/SSL session inside an FTP session, the protocol creates an SSH session and then opens an FTP session inside the current secure session.

- **HTTP, HTTPS**: In the last years numerous internet sites are using the HTTP protocol to distribute multimedia content (e.g. YouTube or Flickr). The HTTP protocol, now in its version 1.1 [44] was originally designed to transfer web pages from web servers to the user browsers. Nowadays, the HTTP protocol is often used to transfer also binary information, displacing FTP. As in FTP, HTTP does not provide any type of security. In order to provide security services, an extension of the protocol named HTTPS [95] (HTTP Secure) provides mechanisms to encrypt the communication between the user and the servers with optional authentication services. Regarding the performance of FTP against HTTP [112], the HTTP protocol presents less communication overhead. It uses only one connection for both control and data transferences and the interaction to establish an HTTP session is simpler than in FTP.

- **GridFTP**: With the development of grid computing, a new protocol was created to perform the transfer of information in these distributed environments. Based on FTP, the GridFTP protocol [5, 6] adds features related with the security and performance of the transfers in distributed environments. Regarding the security, GridFTP authenticates users prior transferring the information using the Grid Security Infrastructure (GSI) mechanisms (e.g., X.509 certificates).

A new extension (sshftp://) allows the control channel to be secured by creating a SSH wrapper. In terms of performance, GridFTP offers some important features: parallel striped transfers [7], third-party transfers, and partial file transfers. Parallel striped transfers allow a transfer to create parallel GridFTP data channels among a set of servers. Third-party transfers allow a user to transfer data between two other entities. Finally, in case of failures during the transfer process, the GridFTP protocol permits partial file transfers to complete the failed transfer. Last versions of the Globus Toolkit introduce a Reliable File Transfer (RFT) service on top of GridFTP that is able to relaunch a previous download in case of failure. Compared to other protocols, GridFTP has two main disadvantages. First, it requires a working grid infrastructure both on the client and the server, making it difficult to use it in other situations. Second, even though it offers the possibility of encrypting the payload, it is disabled by default as it incurs into a noticeable overhead.

- **Peer-to-Peer protocols**: The growth in the number of Internet users and the bandwidth capacity have contributed to the popularity of peer-to-peer networks. Instead of a client-server approach, peer-to-peer transfers are based on the idea that each user behaves both as client requesting data and as server of the data it has already downloaded. Several peer-to-peer applications such as Napster [16], Gnutella [17], eMule [18], etc. contributed to the popularization of peer-to-peer technologies. In this respect, BitTorrent [28] is one of the most popular peer-to-peer protocols currently in use. The protocol defines two participating entities: the tracker and the peers. The BitTorrent tracker is the entity in charge of managing the availability status. It stores a list with the active peers and the available parts of the file they have already downloaded. In order to start a transfer, the peers contact the tracker (contact information is obtained from the *.torrent* files) in charge to retrieve the list of seeds. Once the peer knows other peers involved in the transfer it automatically negotiates which parts of the files will be downloaded from each client based on the popularity, network status and other peer statistics. In order to reduce the load on the tracker and improve the fault tolerance, new extensions of the protocol try to create a peer-to-peer network of peers. The Distribute Hash Table (DHT) extension [72] allows a tracker less utilization of BitTorrent.

In practice, the selection of the transfer protocols has a huge impact in the perceived performance. However, it is necessary to assess external factors that can affect the final performance. While some protocols may be the best candidates for a particular scenario, how the Internet provider manages them may be critical from the performance point of view. As an example, both HTTP and BitTorrent typically suffer from ISP throttling, and therefore their effective performance is reduced compared with the theoretical one.

16 music.napster.com
17 gtk-gnutella.sourceforge.net
18 www.emule-project.net

2.4 SOCIAL NETWORK ANALYSIS

The study of social networks is generating a lot of interest in the recent years due to two emerging factors. First, online social networks are becoming a popular form of communicating with friends, family, etc. Their increase in popularity makes possible to have access to large networks that in the past where not practical to obtain. Second, the improvement in technology to store, access, and analyze this type of large data based on new paradigms (e.g., map-reduce, cloud computing, etc.) permits to obtain the required computing infrastructures at acceptable costs. The information contained in a social network is highly valuable as it provides insights of the user behavior and can be used for a variety of purposes: optimization of server infrastructures, optimization of storage systems, user-targeted marketing, etc. This section introduces the concept of static analysis and community detection in online social networks.

2.4.1 *Statistical study of social networks*

The evolution of social networks has created new challenges regarding their study and analysis. Large social networks can easily overpass 100 million users and billions of links, making them one of the most complex structures to be analyzed. In this context, statistical studies provide a general view of the inner workings and structure of the network. For example, understanding which is the diameter of a network (i.e., longest shortest path) provides information about how many flooding steps may be necessary to cover the network (e.g., for improving search or discovery processes). Similarly, the clustering coefficient provides information about how connected are the users among themselves (e.g., to determine the propagation speed of a message broadcasted by a user). In this section, we overview several works that analyze different characteristics of a social network.

Cha et al. [24, 23] study the popularity of video in YouTube [19] and Daum Videos [20] analyzing the dynamics of the popularity distribution and evolution and the level of content duplication. Mislove et al. [76] analyze the structure of four popular social networks: Flickr [21], YouTube, LiveJournal [22], and Orkut [23]. The authors obtain a data set with 11.3 million users and 328 million links by crawling the public user web pages. The results confirm that these type of networks show power-law, small-world and scale-free properties.

Ding et al. [33] center their study on the behavior of YouTube users that upload content. The authors discover that the number of uploaded videos per user follows a Zipf distribution and that only a minority of users uploads content frequently. Additionally, the authors discover that a large fraction of the uploaded videos correspond with multimedia material not generated by the users themselves (e.g., fragment of a movie, video-clip, etc.). Ahn et al. [4] center their study on Cyworld [24], MySpace [25]

19 www.youtube.com
20 tvpot.daum.net
21 www.flickr.com
22 livejournal.com
23 www.orkut.com
24 Popular online social network in South Korea cyworld.com
25 www.myspace.com

and Orkut. The authors are able to obtain access to the complete Cyworld data set, and evaluate whether the sampling method is effective when compared with other social networks. The paper also compares how the scale of the networks evolves over time showing significant differences between sites.

2.4.2 *Community detection in social networks*

The use of static analysis of social networks provides a general overview of the inner workings of the whole network. However, the knowledge extracted from this analysis is based in the assumption that all the nodes in the network share the same characteristics. To overcome this limitation a significant research effort has been dedicated to detect communities in social networks. In this way, a network is partitioned into a set of communities, each of them containing nodes that share common characteristics. This finer definition of the network as a composition of communities permits to increase the level of detail in future static analysis refining the knowledge that can be extracted from them. As an example, the social network formed by the users of YouTube could be divided into communities based on the type of videos they usually consume (music, games, entertainment, etc.).

Community detection algorithms share common characteristics with clustering algorithms. In general terms, a community detection algorithm can be seen as the application of a clustering algorithm using a similarity metric related with the social characteristics of the network. In this respect, community detection algorithms can be also classified [15, 124] using the existing categories for clustering algorithms: how the partitioning is performed and what type of supervision has the algorithm.

Regarding the method used to define the partitions, we find hierarchical and partitioning methods. Hierarchical methods employ a dendrogram structure to build a hierarchy of clusters either using an agglomerative or a divisive approach. The agglomerative approach considers an initial state where each node corresponds with one cluster and builds the upper clusters using a bottom-up approach. In the divisive approach, the network is initially composed of one cluster and it is successively split. Partitioning methods on the other hand target to determine which is the optimal composition of the clusters usually by iteratively deciding which community a node should belong to. Clustering algorithms are also characterized by the level of supervision involved. From the practical point of view it affects what are the input parameters of the algorithm. For example, whether the number of clusters in the networks is an input parameter or is a result of applying a clustering algorithm.

From the point of view of the algorithms, many community detection methods [47] have been proposed. However, the computational complexity involved in the process usually leads to modified algorithms or heuristics that can provide similar results in a fraction of time. Duch and Arenas [37] propose a community detection method based on the upper bound value of the modularity. This metric [81] measures the quality of partitions of a network. By measuring the fraction of edges in the social network that are interconnected within the same community, it is possible to evaluate the quality of the partitioning.

In [36] the authors propose an algorithm based on measuring the form of the overlapping cliques after generating a set of communities. Contrary to other supervised approaches, the algorithm does not require to introduce the number of expected com-

munities. Similarly, Zhao and Zhang. [133] propose a clustering method that considers overlapping communities with the addition of a hierarchical tree representation to speedup the detection process. Liu et al. [71] define a method to transform the community detection problem into a clustering problem by introducing an affinity propagation metric. As before, the algorithm does not require previous knowledge of the number of communities. In [126] the authors propose an algorithm that combines link and content information in the community detection process. Authors take into account the popularity of nodes, and introduce a discriminative selection of content attributes.

The applicability of the existing community detection algorithms to large social network traces poses new challenges due to their size and complexity. In [52] the authors propose a multi-stage scalable community detection algorithm applied to YouTube traces with the objective of improving content recommendation and discovery. In [66] the authors describe how community detection can be speedup for large networks. In the same research direction, Mislove et al. [77] study the feasibility of discovering missing user attributes by inference from the users in the same community.

2.5 SUMMARY

This chapter presented an overview of different topics related with data distribution systems. First, we provide a classification of data distribution systems according to how data is discovered, stored and accessed by overviewing publish/subscribe architectures, data grids and content delivery networks. Publish/subscribe architectures create an intermediate point in a message distribution architecture. Publishers create events with new content and the infrastructure delivers the events to interested users. While this infrastructure permits a highly decoupled deployment and does not require the entities to be online when new events are notified, they require a significant knowledge to determine which users must receive each message. Data grids represent a more structured architecture for transferring data. As this type of networks require an underlying grid infrastructure (users must obtain a grid certificate for accessing data) it is convenient for existing grids, but are difficult to introduce in other more open deployments.

Content delivery networks represent one of the most employed technologies nowadays to facilitate the distribution of content. They simplify the infrastructure required to distribute data, as the data owner only leases the necessary infrastructure. Content delivery networks are geographically distributed and mirror the required content near the user location to improve access latency. However, managing this type of infrastructure is challenging, as it is necessary for the data owner to determine how data will be distributed in the network and how many edge servers are required for a particular case. This is a challenging problem that appears in many situations, and solving it is an ongoing research topic.

Second, we explore different methods to model data distribution systems. The modelization of these types of systems permits to define optimal policies for data management and movement. Approximation algorithms represent a common approach for this type of problems. This type of algorithms is characterized by providing solutions in a reasonable time but usually do not guarantee what is the quality

of the solution provided compared with the optimum value. On contrast, mathematical models provide optimum solutions but they require the construction of complex models, and this complexity usually results in significant computing times. While some approaches propose the use of distributed solvers, they lack generalization, and therefore are tailored to specific problems.

Third, we overview different methods to provide efficient data transfers. This problem is fundamental, as it does not matter the quality of a schedule produced by the most reliable model, if it is not possible to deploy it in a real system. In this respect, different policies can be used to parallelize data transfers, and different protocols offer a variety of functionalities such as secure or reliable transfers. While there is a rich set of protocols and policies, it remains unclear how to combine different protocols to access the same data in parallel.

Finally, we describe how social networks can be analyzed to obtain information regarding the user behavior. In this respect we overview statistical studies and community detection methods. By applying a statistical study it is possible to obtain a general view of the structure of the networks. In this sense, metrics such as clustering coefficient or network diameter can be used to determine how connected are the users of a network, what is the friendship distribution, or the theoretical speed in message propagation. Then, we overview different studies centered on finding communities in a social network. By applying community detection methods it is possible to split the network into a set of partitions. Each partition corresponds to a set of users sharing common characteristics such as their preferred music or movie genre. By combining community detection methods and statistical analysis of each partition it is possible to obtain a more detailed view of the inner workings of a social network. This knowledge can be used in different aspects such as optimization of server infrastructures or optimization of storage systems. However, the studies show that analyzing large online social networks imposes different challenges derived from their size and the complexity of the available algorithms.

The analysis of the existing related work shows the different existing techniques to distribute data, model data distribution systems, and analyze social networks. However, the challenge still remains to define a single architecture capable of combining the different techniques. In this thesis, we combine the aforementioned techniques to define a publish-subscribe architecture that: employs a mathematical model to schedule data transfers, uses community detection algorithms to reduce the number of transfers from the server infrastructure, and provides reliable fast data transfer mechanisms.

3

A GENERIC DATA DISTRIBUTION SCENARIO

The main objective of the presented work is to define a new efficient and reliable data distribution architecture that leverages the knowledge extracted from the user community to improve the data transfer performance. Before presenting in detail the different components of the architecture, this chapter presents a high level view of the proposed architecture and its main objectives and functionalities. This chapter overviews the different challenges found in current data distribution systems and describes how our architecture addresses each of them.

Designing an efficient data distribution architecture is a challenging problem, mainly because it involves many components at different abstraction layers. In order to abstract the data transfer problem, we opt for a graph representation. Therefore, the different actors involved become the nodes of a graph G. A set of server nodes S stores files from a global set F. The set of users U requests a subset of files from the server infrastructure. Considering these sets as possible nodes of a graph, we define E as the set of edges connecting different elements.

The resulting graph $G = (S \cup F \cup U, E)$ represents a data distribution scenario in which subsets of files are stored in different servers, and the users request files from servers. Figure 4 depicts the graphical representation of a example graph. In this figure, we observe two important characteristics. First, the graph may be composed of several disconnected components. In this case, each component of the graph represents a set of users, servers, and files that are not linked with any other node in the graph. Second, a file may be stored in more than one server in the infrastructure (mirroring and/or proxying), which requires the introduction of a mechanism to manage the lifecycle of the file in the system and how the users access it. It should be noticed that even though we have considered files as the data entities to be transferred, our approach can be easily extended to support other data sources.

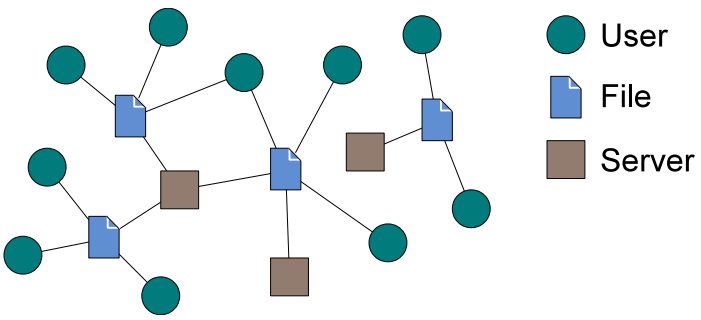
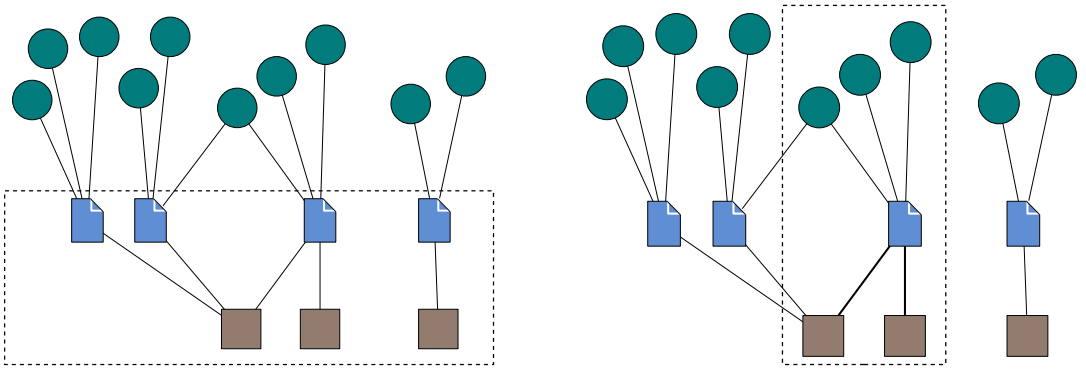


Figure 4: Data distribution graph



(a) File mapping on the server infrastructure (b) User redirection on the server infrastructure

Figure 5: Determining file-server mapping (left) and user redirection (right) on the server infrastructure.

The graph representation has been employed in previous studies such as [59]. However, our graph is not limited to the user interests, including the relationships of the content with the underlying server infrastructure. This additional knowledge exposes two main problems. First, we need to determine how files should be mapped onto the set of available servers (Figure 5a). A file may be replicated across the infrastructure, thus requiring to determine the number of replicas and their placement. Second, as files may be replicated, it is necessary to decide how users requesting a particular file will be redirected to a server storing that file (Figure 5b). An efficient user request distribution reduces the hot spots and permits to achieve a better load balancing among servers.

The main objective of this thesis is to propose an optimized data distribution infrastructure. As this type of infrastructures are composed by different types of elements that need to work cohesively, it is important to determine what would be the effect of modifying existing elements or introducing new ones. Designing such a system can take two completely different paths. A first option would be to completely redesign the system in order to improve the performance. However, significant changes in the involved entities or in the underlying protocols can be more disruptive than functional. As an example, consider the impact and effort required to modify all servers in a large server infrastructure to install a new protocol, or modify their current configuration. In this work, we approach the design phase as a problem of how to efficiently use the existing elements to improve the overall performance.

In this sense, we provide a general overview of the main challenges faced by this kind of distributed systems and the limitations imposed. In particular, we consider three main challenges: system sizing, system availability, and compatibility with existing infrastructures, which we explore in detail:

- **System sizing**: Different type of applications such as web pages serving user generated content (e.g., Youtube, LastFM or Facebook), scientific applications running in cloud environments or dedicated clusters, etc., have been shown to present highly variable workload patterns [116, 53, 127]. As an example, Figure 6 shows the average number of requests per minute received by of 10% of the Wikipedia [110] servers between September 19^{th} and October 2^{nd}, 2007.

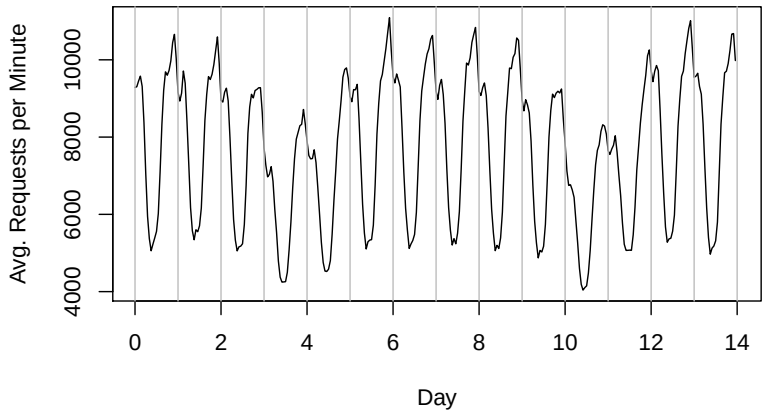


Figure 6: Number of requests received by 10% of Wikipedia servers between September 19th and October 2nd, 2007.

This variability makes necessary to dynamically adapt the system size to the current demand in order to optimize the infrastructure cost.

- **System availability**: Considering an application with a variable workload, assuring quality of service becomes a challenge. Changes in the system size have a direct impact on the availability of the infrastructure. Adding new servers reduces the problem, as the requests are distributed among a bigger set of servers. In this situation, the likelihood of redirecting a request to an overloaded server decreases. By contrast, in the event of decreasing the system size to reduce the infrastructure cost, maintaining the previous quality of service level becomes a problem.

- **Compatibility with existing infrastructures**: Different proposals [102, 7, 14] to optimize the behavior of data distribution infrastructures require changes in low level components of the system. In this sense, the approach of this thesis differs from other works in that our proposal maintains the *de facto* infrastructure standards. Changing the underlying protocols may increase the throughput of a system, but it has a major drawback: it requires updating or modifying the software stacks of all components involved (users, servers, and intermediate appliances such as routers). While users of the system may be more inclined to update their systems, this type of update is a challenge in itself when applied to servers or intermediate appliances as backward compatibility must be assured in most cases and it requires of a significant investment. In this work, we focus on optimizing the system behavior using the existing protocols and server infrastructure practices.

Taking into account the presented challenges, this thesis introduces a set of components that optimize specific problems in the data distribution process, without requiring any low-level change in the underlying infrastructure.

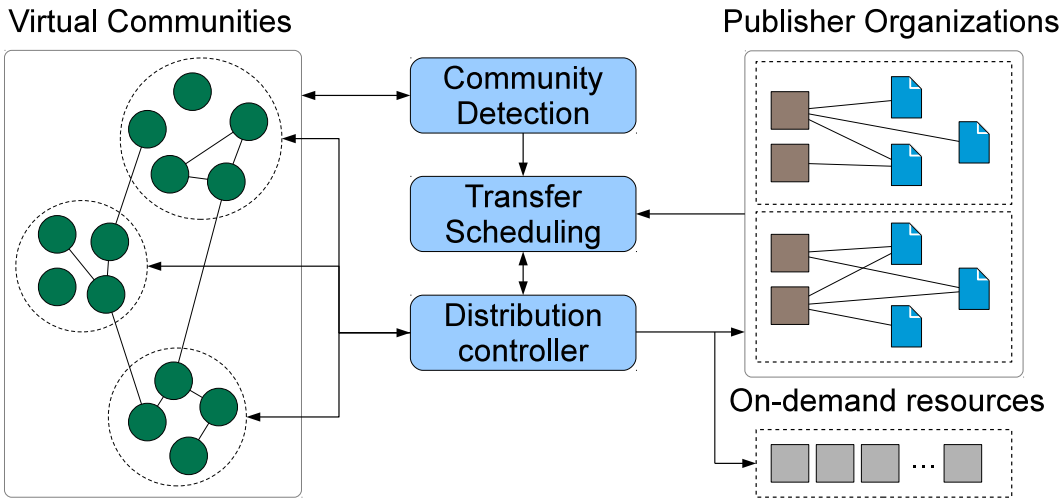


Figure 7: Generic data distribution architecture.

3.1 DECOMPOSITION OF A DATA DISTRIBUTION PROCESS

The complexity of the data distribution process makes necessary to divide the problem into smaller parts, such that each proposed technique produces a clear improvement on specific parts of the process. Going back to the data distribution graph presented before (Figure 4), a data distribution process involves a set of users requesting subsets of files from a server infrastructure.

In this thesis, we introduce a set of modules, each of them addressing a particular problem: analyzing the user community requesting the files, monitoring the status of the system, scheduling the file transfers, and notifying and transferring data from servers to users. Figure 7 provides a high level overview of the resulting architecture showing the main components involved:

1. **Virtual communities**: The users requesting files from the servers are grouped into virtual communities based on a similarity measure. Each virtual community will contain a local proxy permitting to reduce the number of transfers from the servers to the users. A file accessed by all users in a virtual community will be transferred from the servers to the local proxy, and then locally distributed to the users. In this way, we reduce the workload on the server infrastructure.

2. **Publisher organizations**: The infrastructure serving content to users is divided into publisher organizations depending on the data owner. The data owner is the entity responsible for creating and maintaining the data, and granting access to the users. In this respect, our architecture considers the possibility that in order to access some files the users must present the necessary credentials. This characteristic therefore limits the movement of data between different components of the infrastructure and requires the definition of the appropriate mechanisms to enforce it.

3. **On-demand resources**: The publisher organizations control a series of dedicated resources to serve content to users. However, the distribution infrastructure must be able to cope with unexpected workload spikes. In order to achieve this objective a pool of on-demand resources is available to activate new servers when necessary. Notice that in practice, these resources are typically expected to be implemented as cloud computing instances, but any other solution can be also considered for our purposes.

4. **Community detection**: The community detection module analyzes the relationships between the users of the system and proposes different aggregations in the form of virtual communities (*VC*). By grouping together users with the same interest (e.g., requesting the same file or being in the same location), the size of the data distribution graph is significantly reduced. Additionally, the information obtained from the formation of virtual communities can be used to determine request patterns and to identify possible proxy locations.

5. **Transfer scheduling**: Using the information from the community detection module and the current state of the system, this module schedules the required transfers. The scheduling module uses an optimization process to obtain a schedule with a reduced makespan, avoiding hot spots in the server infrastructures, and reducing the flash crowd effects. As a result of this process, the distribution controller infrastructure manager may need to trigger a system resizing action in order to satisfy the existing demand.

6. **Distribution controller**: The distribution controller has two main responsibilities. First, it checks the status of the system in order to determine the number of available servers, their characteristics, and their cache contents. Seconds, it monitors the arrival of new data from any publisher organization and commands the remainder of components to perform the distribution to interested users. In order to do that, the controller evaluates the current state of the distribution infrastructure and schedules the transfers. Once a schedule has been determined, this module notifies the interested users in the order determined by the schedule. This module is connected with a proxy location inside the virtual organizations that will download the data from the specified sources.

The modularity of the architecture permits to introduce new modules or modify the existing ones to adapt the behavior of the system to the specific requirements of the application scenario. As an example, we could replace the community detection module with a user-defined list of communities; we could add data availability prediction module, etc.

3.2 APPLICABILITY

Presenting a generic data distribution process at a high level of abstraction introduces the risk of missing details that may not make possible to use the proposed modules in a real scenario. This section overviews several real application scenarios that could benefit from this work. The following paragraphs describe their characteristics and the possible benefits of adopting the proposed architecture.

- **Massive data distribution processes**: To improve the user experience, a large number of applications and systems implement automatic updates. These types of updates usually involve the client checking for updates and transferring the new data when necessary. While this scenario reduces the user intervention to download new versions of software, it introduces the major problem of flash crowds. Large systems with this type of updates such as Apple, Ubuntu, etc., suffer from massive number of accesses on update days. When a large number of users discover the update at the same time, the servers containing the update are likely to be overloaded by the demand. One possible solution is to enlarge the infrastructure, but it incurs in a higher cost. By adopting our proposed architecture it is possible to reduce the overload in the servers, by means of grouping user requests into clusters and planning the transfer processes.

- **Pre-caching on dynamic infrastructures**: A large number of user generated content networks such as Wikipedia, Youtube, LastFM, etc., employ the benefits of cloud infrastructures to scale the number of servers according to the demand. New machines added to the infrastructure start from a fresh state, resulting in empty caches. Depending on the content, pre-caching at least the most popular contents will improve the machine responsiveness when it starts to process user requests. In this case, the new machines added to the system behave like user destinations, and the existing server infrastructure is used to provide the copies of data avoiding backend accesses. In this scenario we can optimize its performance by introducing scheduling in the transfer process in order to reduce the makespan of booting and preparing new instances to serve content to users.

- **Scientific applications in cloud computing environments**: Running scientific applications in cloud computing environments is a growing practice. The cloud computing permits to reserve computing clusters on demand with similar characteristics to the owned computing clusters. This type of application is characterized by being mostly computing intensive. In this sense, the usual workflow is to read the input data, perform the required computation and generate the results. In this scenario, if we consider cloud computing instances as user requesting data and the server infrastructure as the servers storing the input data, we obtain a version of our base data distribution problem. This scenario also presents other interesting characteristic: once the computation has finished, the roles are interchanged, as the cloud instances are the one sending data to the server infrastructure.

The mentioned scenarios constitute a basic sample of the target applications where our proposal can optimize the system performance. In this respect, many other scenarios such as big data analysis on cloud infrastructures, optimization of communication in parallel applications could also benefit from parts of our proposal.

3.3 SUMMARY

In this chapter, we have presented the data distribution problem addressed by this thesis. The objective of this work is to define a new efficient and reliable data distri-

bution architecture that leverages the knowledge extracted from the user community to improve the data transfer performance.

In order to fulfill this objective, we define a modular architecture and provide a high level view of the different components involved. Our architecture considers the existence of a set of users that request files from a set of servers. The servers may be grouped in different publisher organizations, and the files may be replicated across the server infrastructure. The architecture is composed of four main modules. The system controller is responsible of monitoring the status of the servers and triggers the distribution operation whenever new files are requested or become available. In this respect, we support a publish/subscribe scenario where the users have previously subscribed to different types of contents.

A community detection module analyzes the relationships between the users and the requested files and provides a set of virtual communities. This approach simplifies the representation of the problem by grouping the users into virtual communities attending to their common interest. In this way, we are also able to introduce local proxies in each community in order to reduce the server infrastructure workload. The information generated by this module is used by the transfer scheduling module in order to decide which is the best strategy to distribute the requested files avoiding hot spots and flash crowd effects. Once a schedule with a reduced makespan has been produced, the notification module triggers the download operation in the user locations. Using this approach it is possible to efficiently distribute large set of files to a large set of users in the minimum time.

To conclude this chapter, we discuss the applicability of our proposal and describe how it can be applied to different scenarios such as: massive data distribution processes, pre-caching on dynamic infrastructures or the execution of scientific applications in cloud environments.

4

ANALYSIS OF ONLINE COMMUNITIES

The advent of the Web 2.0 has popularized the concept of social network up to the point where using some type of social network is now part of the everyday life of a significant part of the world population. Social networks allow users to declare their interests, their friendships and to share content among friends. As an example of the magnitude of this type of applications, Facebook [42] reported 955 million monthly active users at the end of June 2012, storing more than 100 petabytes of photos and videos. Understanding the information contained in a social network permits to obtain a deeper knowledge of the user behavior and optimize different processes accordingly. In a previous work [108], we studied how a peer-to-peer network could benefit from the existence of small-world communities. In that work, we proposed Affinity P2P (AP2P), a self-organizing peer-to-peer network where each node joins a set of clusters of interest according to its content affinity. Our evaluation demonstrated that grouping users into communities improves the search recall and reduces search latency. However, our experiments were carried out using a YouTube dataset in which the existing categories where known *a priori*. This restriction imposes a significant limitation in order to exploit the existence of user communities in other scenarios. As a result, we started exploring different methods to automatically discover user communities in social networks, allowing us to employ social knowledge in other application scenarios.

In this thesis, we focus on the identification of the virtual communities found in a data distribution graph. Detecting the existing virtual communities brings two significant benefits. First, the existence of virtual communities permits to reduce the graph representation of a data distribution problem as the user nodes may be substituted by a significant smaller set of virtual communities. Second, each virtual

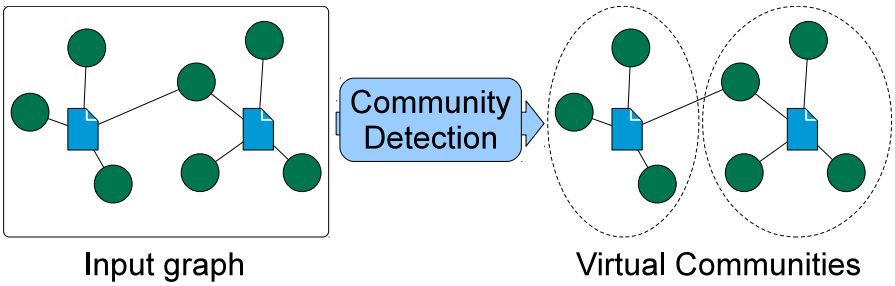


Figure 8: Definition of the virtual communities using the community detection module. The input data distribution graph (left) is processed by the module to produce a set of virtual communities over the input graph.

community can be seen as a candidate for proxy locations as each community will group users with similar preferences.

Based on the data distribution architecture defined in Chapter 3, this chapter focuses on the Community Detection (Figure 8) module of the architecture. This module analyzes the relationships between the users of the system and the files they access, and defines a set of virtual communities. The extracted virtual communities will then be used as the input for the transfer scheduling module of the architecture. The existence of virtual communities in the system permits to employ compact representations of the data distribution graph, thus simplifying its future processing.

In this work, we propose a method to apply community detection algorithms to data distribution networks. We study the performance of the existing algorithms and propose a new iterative community detection algorithm to overcome the existing limitations.

4.1 EXISTING METHODS TO DETECT USER COMMUNITIES

The concept of social network refers to a set of entities connected by different types of links between them. We can categorize the social networks into two main types depending on whether the links are explicitly declared by the entities, or they are inferred through a data mining process. Using this characteristic, we distinguish between two types of networks: *explicit* and *inferred* social networks.

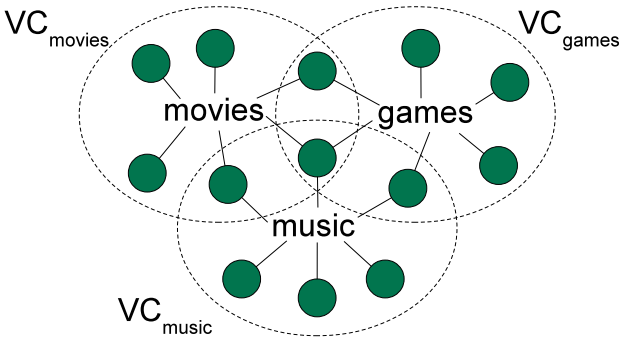


Figure 9: Example of an explicit social network. A virtual community is formed around each of the declared characteristics. Notice that a user may belong to several communities at the same time.

- *Explicit social networks*: This type of network is the most common form associated with the broader term social network. A declarative social network requires the entities to actively declare the existing links with other entities. Typical examples of this type of networks are sites such as Facebook [1] or Google+ [2]. Nonetheless, social networks are not limited to friendship relationships. Other networks focus on other aspects. For example, Youtube [3] forms a social network based on the different types of relationships (share, comment, like, etc.)

1 facebook.com
2 plus.google.com
3 youtube.com

of people with videos; Last.FM [4] analogously focuses on music content. As an example, Figure 9 shows an explicit social network with the entertainment preferences of users. Each entity of the system declares its specific interests for a particular item. As a consequence, a virtual community is formed surrounding each preference or taste (e.g., movies, games, music, etc.). This results in the existence of one virtual community per item in the system, where a user may be present in different communities at the same time.

- *Inferred social networks*: When the relationship is not actively defined in the network, it is also possible to define virtual communities that aggregate users based on the results of a data mining process. The information contained in the system is analyzed to reveal the hidden characteristics or facts that are not known a priori. The typical approach to obtain this information is to apply a community detection algorithm grouping users that have similar characteristics. As an example, Figure 10 shows an inferred social network considering the files accessed by users in the last 24 hours. The access information is used to partition the network into a set of virtual communities. While this approach is usually employed in this type of networks, it is also possible to apply these procedures to explicit social networks in order to reduce the number of virtual communities.

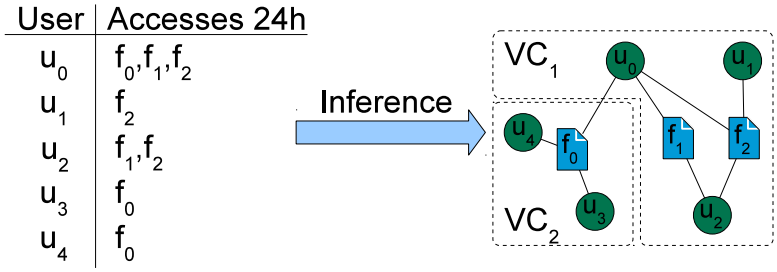


Figure 10: Example of an inferred social network based on the file accesses of a set of users during the last 24 hours. Notice that each user only belongs to a single virtual community and there is no overlapping.

In this work, we consider the possibility of using both types of networks as input for the community detection module. The objective of the module is to define a set of virtual communities (VC) using the available information, either the explicit social network, or the information needed to infer it. This permits to determine points in the network that can behave as proxies in such a way that when the same file is requested by a group of users, it is possible to reduce the number of transfers from the origin servers to the proxies to one, and make all the users access the same proxy copy of the file. The addition of proxies to a data distribution infrastructure permits to control the tradeoff between the backend cost for the infrastructure owner, and the quality of service perceived by the user.

Given a data distribution graph, our objective is to find a method that provides partitions that satisfy three characteristics: 1) contain a minimum number of users, 2) have a maximum number of files assigned to the partition, and 3) the set of files

4 last.fm

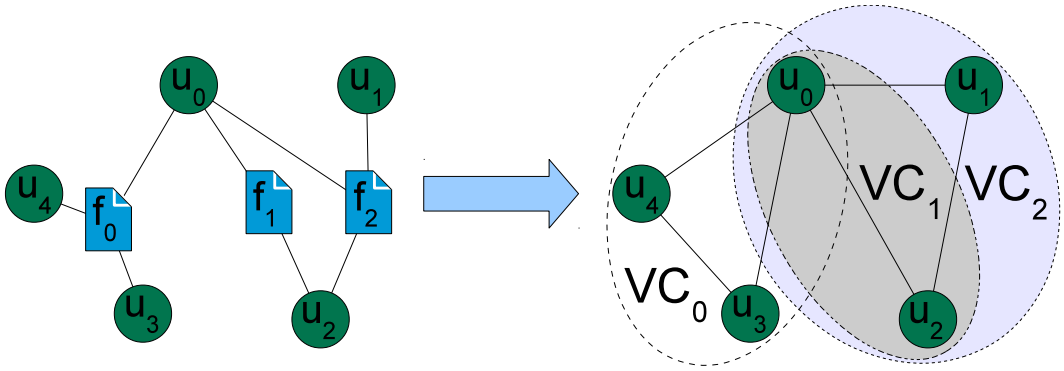


Figure 11: Detecting communities on declarative social networks using item selection.

shared on all partitions should be minimal. These characteristics are desired in a data distribution system in order to avoid communities with a large number of users and communities with a large, and possibly the same, set of assigned files. The following sections describe two techniques to partition a social network into a see of virtual communities.

4.1.1 *Detecting online communities using item selection*

A common approach to detect online communities in explicit social networks is to identify which subset of users are interested in the same items. Notice that the definition of item is generic and may vary from network to network. For example, an item may be a category of files, a music genre, a preference, a set of files, etc. Considering the data distribution graph $G(F \cup U, E)$ as input, we can define the set of virtual communities VC as:

$$VC_k = \bigcup_{j=0}^{|U|} u_j : (\exists\ e_{jk} : f_k \in F_j, e_{jk} \in E) \tag{1}$$

where VC_k is the resulting virtual community, e_{jk} is the edge connecting user u_j with item f_k, and F_j is the set of items the user u_j is linked with. In the case of a data distribution scenario, the items correspond to the files. As an example, Figure 11 shows the result of applying the aforementioned method to a graph composed of 5 users and 3 files. The partitioning method defines a set of virtual communities (VC_0 for f_0, VC_1 for f_1, and VC_2 for f_2.) in such a way that a community is defined around a related file.

This type of approach presents several problems that discourage its use in real applications. First, it produces a set of overlapping communities, which in our case translates in a user belonging to a set of different virtual communities. From the point of view of a data distribution infrastructure, it is preferred to assign each user a single entry point to the infrastructure. This is the usual practice in many content delivery networks. Second, the partitioning method requires the global data distribution graph to define the communities (i.e., a graph with users and files).

In practice, this graph may be significantly large, and therefore the computational complexity of the method may make it unfeasible for medium to large networks.

4.1.2 *Detecting online communities using community detection algorithms*

The detection of virtual communities using the aforementioned approach produces a virtual community per item. As this number can grow rapidly and it can produce low populated virtual communities, we propose a method to apply existing community detection algorithms to data distribution networks. This type of procedure can be applied to both explicit and inferred social networks and produces a number of virtual communities independent of the number of items in the system.

Existing community detection algorithms require an input graph with only one type of vertex (either user or file). Therefore it is necessary to define a mechanism to adapt a data distribution graph in such a way that can be used with existing algorithms. Our approach consists of two phases: first, we abstract the information contained in the initial graph and define a new condensed weighted graph; second, we analyze the resulting graph using a community detection algorithms to obtain the existing partitions. The following paragraphs provide a detailed description of both processes.

1. **Condensing the graph representation**

   The first step in the process is to transform the initial graph in such a way that it only contains one type of vertex. The original graph $G = (S \cup F \cup U, E)$ contains three types of vertices: servers, files, and users. The exiting algorithms are not able to detect communities in this type of graph, and therefore it is necessary to condense the representation limiting the type of vertex to a single one. For the purpose of detecting online communities, the set of servers that store the files are not relevant and therefore can be directly removed. In contrast, the relationship between users and files needs to be maintained.

   In order to maintain this information, it is necessary to produce a condensed graph that maintains only one type of vertex: users or files. To establish the relationship between the selected type of vertex, we link together two vertices if they share at least one common item of the other type. The links are weighted according to the number of common items shared. In practice, we could obtain two possible graphs. Either a graph of users, where two users are linked together depending on the common files accessed by them; or a file graph, were two files are linked when they have a common user accessing both files.

   The existing challenge in this aspect is to define the metric used to establish the weight of the relationship between two entities. In this thesis, we focus on two metrics that weight the commonality between two entities: *intersection* and *Jaccard*. However, the method described for condensing a graph, can be applied using any other similarity metric. For brevity purposes, the remaining of this section considers the creation of a user graph. The file graph can be created analogously using the same metrics.

   The intersection metric (Equation 2) corresponds to the number of files that are accessed by user i and j. It is an absolute measure that captures the number of

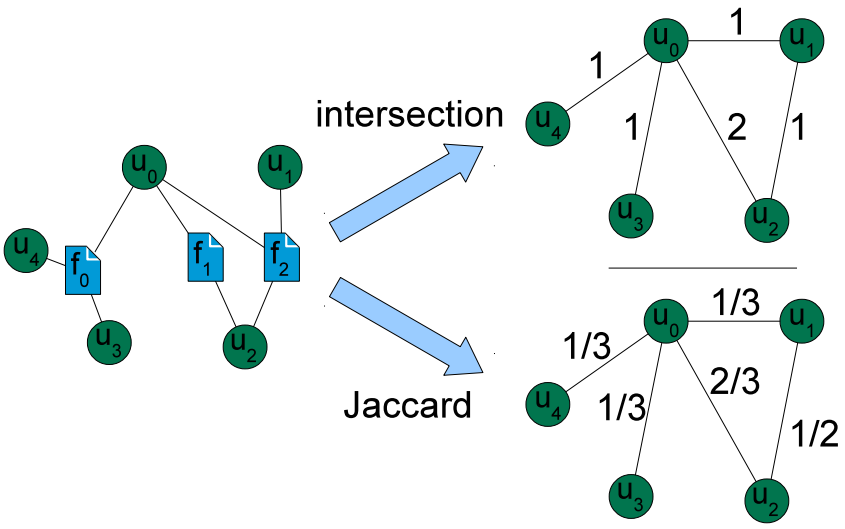


Figure 12: Application of intersection and Jaccard metrics to produce a condensed graph, with $\mu_I = \mu_J = 0$.

common items associated to both entities. Iamnitchi et al. [59] propose this metric to determine the weights in an *interest-sharing graph*. The authors introduce a threshold μ_I that determines the minimum intersection value to consider a relationship meaningful.

$$w_{i,j}^{intersection} = \begin{cases} |F_i \bigcap F_j| & \text{if } |F_i \bigcap F_j| \geqslant \mu_I \\ \emptyset & otherwise \end{cases} \tag{2}$$

In addition, we also consider the Jaccard metric (Equation 3), which is a relative measure of the common interest. It is defined as the number of common files requested by two users i and j divided by the total number of distinct files requested by both users. As before, we also introduce a threshold μ_J that establishes when a weight is considered meaningful and must be defined.

$$w_{i,j}^{jaccard} = \begin{cases} \frac{|F_i \bigcap F_j|}{|F_i \bigcup F_j|} & \text{if } \frac{|F_i \bigcap F_j|}{|F_i \bigcup F_j|} \geqslant \mu_J \\ \emptyset & otherwise \end{cases} \tag{3}$$

To illustrate both metrics, Figure 12 depicts a graph of users and files taken as input (left), and the resulting user graphs using the intersection metric (top right) and the Jaccard metric (bottom right). As shown in the figure, the shape of both resulting graphs is equivalent, and the differences are found in the weights. The selection of which measure will produce the best condensed graph depends on the application scenario, and the impact of using relative or absolute measures in the edge weight distributions.

2. **Applying a community detection algorithm**

   After the condensed graph has been produced, the next step is to analyze the graph and proceed to the partitioning into a set of virtual communities. It is important to notice that we analyze the graphs without any prior knowledge of the number of existing communities (unsupervised learning). Therefore, it is the responsibility of the algorithms to determine this number and the membership of each community. In this work, we focus on several well-known unsupervised algorithms for automatic community detection: Fast greedy [27], Walktrap [89], Label propagation [91], Leading eigenvector [80], Multilevel [18], and Infomap [97].

   The selection of the algorithms has been made according to their popularity and the computational complexity when presented with large graphs. Thus, some well studied algorithms have been discarded due to their excessive computational complexity. The next paragraphs provide a general description of each algorithm showing their main characteristics.

   - *Fast greedy*: The method proposed by Clauset et al. [27] is an improved greedy approximation to the algorithm developed by Neuman [79]. The algorithm is based on optimizing the *modularity* of the different communities found in a network. The metric weights the relationships between the nodes of a community and the links of those nodes with other communities, and compares them with the results of an equivalent random graph. The algorithm assigns entities to a community in such a way that the modularity score of the graph is maximized. The algorithm offers a $O(|V| \cdot \log^2|V|)$ complexity for sparse networks with V vertex, compared with the original complexity of $O(|V|^3)$.
   - *Walktrap*: It is a method proposed by Pons and Latapy [89] based on the idea that short random walks tend to visit nodes belonging to the same community. The authors use the information gathered by the random walks to determine the hierarchical structure of the network, and then select the appropriate partitions to obtain the communities. The algorithm runs in a $O(|V|^2 \cdot \log|V|)$ time in the average case.
   - *Label Propagation*: Raghavan et al. [91] propose an iterative algorithm based on labeling vertices of the social network graph. The algorithm starts with an initial random labeling of the nodes, and in each step, each node adopts the most common label among its neighbors. In this way, after the iterative process ends, nodes in a community are likely to be using the same label. The algorithm has a complexity of $O(|E|)$ for each iteration, being E the set of edges. However, due to the random selection of labels in the first iteration, it may be necessary to execute the algorithm several times to consolidate the solutions.
   - *Leading eigenvector*: In [80] the author propose an algorithm to detect communities based on information contained in the modularity matrix of a network. This iterative algorithm follows a top-down approach. At each step based on the information contained in the matrix, it is decided whether to create two communities or not. This decision is taken based on

whether the modularity will be maximized by the split. The algorithm has a complexity of $O(|V|^2)$ for sparse networks.

- *Multilevel*: Blondel et al. [18] propose an iterative algorithm based on the modularity gain of assigning a node to a community. Initially, the algorithm assigns a community per node. On each iteration, the community formed by the neighbors of each node is evaluated to determine the gain of removing that node from the community. If the gain is positive, the node is placed in the community with the highest gain.
- *Infomap*: Rosvall et al. [97] propose an algorithm based on information theory principles and not around the concept of modularity. In particular, the algorithm centers on a map equation that measures the accuracy of the trajectory of a random walk inside a network. The trajectories are described by labeling each node with a unique identifier. The main idea behind the algorithm is to find an optimal code that describes the trajectories. This optimization leads to iterative refinements of the node labels, where the nodes belonging to the same communities share the same prefix.

Notice that even though we have concentrated on this selection of algorithms to evaluate our method, any other type of algorithm can be applied whether unsupervised or not.

In this work, we employ the two-step method described in this section to determine the existing virtual communities in a data distribution scenario with the objective of assigning one proxy per community. However, the method can be adapted to other scenarios such as content delivery networks. In that scenario, requests would be transformed into probabilities of accessing a particular content and the detected communities would indicate points in the CDN where it is optimal to situate a proxy.

4.2 ITERATIVE WEIGHTED COMMUNITY DETECTION ALGORITHM

The community detection algorithms presented in the previous section produce a set of partitions based on the utilization of unsupervised methods. As a consequence, it is not possible to tune the algorithms to produce communities under certain constraints. While some supervised algorithms [61, 11] permit to specify the maximum and minimum size of a partition, it is not possible to incorporate into the algorithm other metrics related with the quality of the partition from the point of view of a data distribution problem.

To address this problem, we propose a new algorithm that permits to guide the partitioning process. From the point of view of a data distribution infrastructure, we would like to control two parameters: the number of users in a community and the number of files assigned to a community. These parameters impact the requirements of the proxy locations in terms of how many connections they will receive (number of users), and how much storage is required (number of assigned files).

Our iterative community detection algorithm (Algorithm 1) takes as input five parameters: 1) the condensed graph, 2) the entity set used in the condensation phase,

3) the minimum number of elements in a community, 4) the maximum ratio of entities assigned to the community, and 5) an auxiliary community detection algorithm selected from the ones presented in the previous section. The algorithm uses that information to generate a set of partitions given an initial graph.

The algorithm defines a list of communities to be processed, and finishes when the list becomes empty. Initially, the algorithm is executed over the condensed graph, and the resulting communities are added to the list for further processing. On each iteration, we extract a single community from the list and evaluate whether it is necessary to generate new sub-partitions.

Algorithm 1 Iterative community detection algorithm.

```
Input: G, entities /* Entities used during the condensation phase */
Input: communityDetectionMethod()
Input: ratio_VC^entities , min_VC^elements
    solution = ∅
    /* Determine the number of entities per community */
    max_VC^entities = |V| · ratio_VC^entities
    toProcess = communityDetectionMethod(G)
 5: /* Start the iterative processing */
    while toProcess ≠ ∅ do
      c = toProcess.pop()
      entitites_VC = ∅
      for vIndex ∈ c.vertexList() do
10:     entities_VC = entities_VC ⋃ entities[vIndex]
      end for
      if |entities_VC| > max_VC^entities ∧ c.numberVertex() > min_VC^elements then
        aux = communityDetectionMethod(c)
        if |aux| > 1 then
15:       toProcess.push(aux) /* Several partitions detected */
        else
          solution.push(c) /* Only one partition produced. */
        end if
      else
20:     solution.push(c) /* Stop iterative partitioning */
      end if
    end while
    return solution
```

In this respect, the algorithm evaluates whether the virtual community meets two requirements: 1) the partition contains more entities than expected; or 2) the number of elements assigned to the virtual community is larger than expected. These requirements are evaluated in order, and if not satisfied, the algorithm decides that the virtual community must not be further split. Otherwise, the algorithm tries to generate a new set of sub-partitions. These sub-partitions are added to the list of communities to be evaluated unless only one partition is generated. In this respect, we have observed that some of the algorithms are not able to split some input graphs. By iteratively taking one community from the processing list and moving the results

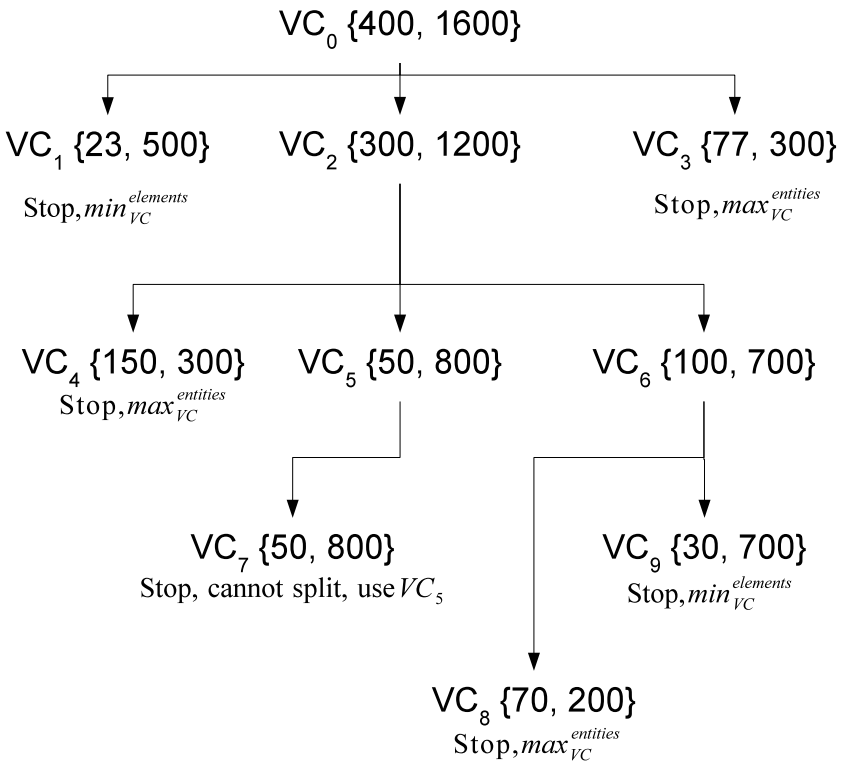


Figure 13: Iterative community detection algorithm applied to a sample graph with 400 users and 1,600 files using $\min_{VC}^{elements} = 30$ and $\max_{VC}^{entities} = 400$. The values of each virtual community represent the number of elements and the number of assigned entities respectively.

to the solution list, the algorithm may be able to produce a larger number of communities when compared with previous approaches.

As an example, Figure 13 describes the different branches of an execution of the proposed iterative algorithm. Initially a graph with 400 users and 1,600 files is taken as input with the algorithm configured with $\min_{VC}^{elements} = 30$ and $\max_{VC}^{entities} = 400$. In the first iteration, the initial community VC_0 is partitioned into three: VC_1, VC_2, and VC_3. The virtual community VC_1 is moved to the solution list as it does not meet the requirement of minimum number of elements in the community ($23 > 30$ is not satisfied). In the next iteration, VC_2 is split into three new virtual communities: VC_4, VC_5, and VC_6. Next, VC_3 is evaluated and considered part of the solution as the number of assigned elements is within the desired range. The same situation is found in the next iteration for VC_4. The evaluation of VC_5 tries to partition the community but fails to produce more than one partition. This may happen due to the internal structure of the subgraph or due to the inability of the selected algorithm to produce new partitions. Finally, VC_6 is evaluated and partitioned into VC_8 and VC_9, which both fail one of the requirements. As a result of the algorithm, a graph that otherwise will generate 3 partitions (VC_1, VC_2, and VC_3), produces 6 refined partitions: VC_1, VC_3, VC_4, VC_5, VC_8, and VC_9.

As shown in the previous example, the algorithm iteratively uses an underlying community detection algorithm to split a community and based on the results of

that partitioning it decides whether additional partitions should be created or not. The complexity of using our iterative solution is therefore the number of iterations of the algorithm multiply by the complexity of the underlying community detection algorithm. To provide an average complexity, we assume a general case where the underlying algorithm at least creates two partitions each time is executed. In this situation the complexity of the algorithm is:

$$iterative_{depth} = max(\frac{|entitites|}{max_{VC}^{entities}}, \frac{|elements|}{min_{VC}^{elements}}) \tag{4}$$

$$O(iterative) = iterative_{depth} \cdot O_{algorithm} \tag{5}$$

where $iterative_{depth}$ corresponds to the number of expected iterations of the algorithm and $O_{algorithm}$ corresponds to the complexity of the underlying community detection algorithm.

4.3 EVALUATION SETUP

In this section, we describe the experimental setup used in the evaluation. First, we present the input data set and summarize its main characteristics. Second, we describe the selected metrics to evaluate the quality of the produced partitions.

4.3.1 *Data set*

In other to evaluate the applicability of community detection algorithms to data distribution scenarios, we create a set of synthetic data distribution networks. We define various graphs, with a fixed number of users and files. In order to build a realistic network, we select a popularity distribution for the files, and connect each file to a selected number of users based on that information. Several studies [24, 108, 109] have investigated the popularity distribution found in many data-sharing systems. The results show that popularity distributions usually follow a combination of statistical distributions depending on the range of the ranking. That is, the top 10 elements may adjust to an exponential distribution, the next 100,000 elements to a power-law, and the next 10,000 to other type of power-law. For our study, we select the *zipf* distribution, as it has been found to be the most common, and it is able to represent the popularity ranking for the larger number of elements in the system.

In this study, we generate two types of initial networks, one with a fixed number of files and varying number of users, and other with a fixed number of users and a varying number of files. The motivation behind this decision is to understand how community results vary when presented with different proportions of users and files. Table 1 details the characteristics of the different graphs generated based on a *zipf* distribution with an α value of 1. From the generated graphs, we can observe two main characteristics that may play a fundamental role when different community detection algorithms are applied.

First, we see that the number of edges quadruples when varying the number of users for a fixed number of files. As expected, this is reflected in the number of edges contained in G_u. However, in both tables, we observe that the amount of

USERS	FILES	EDGES IN G_u	EDGES IN G_f
100	1,600	4,950	615,887
200	1,600	19,900	614,375
400	1,600	79,800	613,720
800	1,600	319,600	614,207
1,600	1,600	1,279,200	614,917
3,200	1,600	5,118,400	615,108
6,400	1,600	20,476,800	614,715
12,800	1,600	81,913,600	614,563
1,600	100	1,264,393	2,311
1,600	200	1,278,938	8,954
1,600	400	1,279,200	38,250
1,600	800	1,279,200	154,852
1,600	3,200	1,279,200	2,500,696
1,600	6,400	1,279,200	10,014,989
1,600	12,800	1,279,200	40,080,631

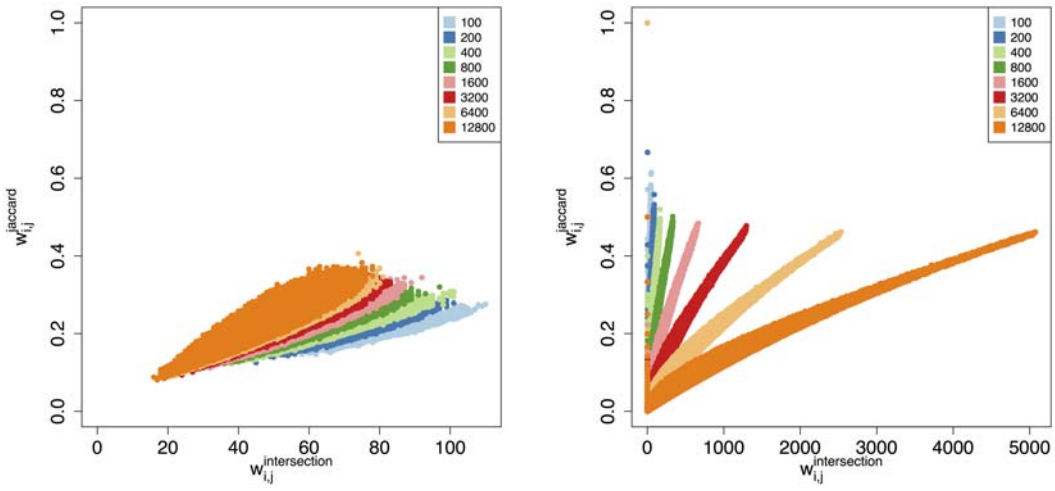
Table 1: Number of edges found in the file and user graphs varying the number of users with a fixed number of files and vice versa.

edges is particularly large. As an example, for 6,400 users and 1,600 files, we obtain a G_u graph with 20,476,800 edges. As a comparison a complete graph with the same number of vertices would have $|U| \cdot (|U| - 1) = 40,953,600$ edges. While our graph has half of edges of the complete graph, it presents a high connectivity degree that may difficult the partitioning process.

Another important aspect is to evaluate how the different weighting metrics, intersection and Jaccard, perform on this type of graphs. Figure 14 plots the correlation between the similarity metrics for a user and a file graph. The results correspond to a configuration of 1,600 files and a varying number of users (100, 200, 400, 800, 1,600, 3,200, 6,400 and 12,800). Notice that in both the user (Figure 14a) and file (Figure 14b) results, the observations obtained for a particular number of users are significantly clustered.

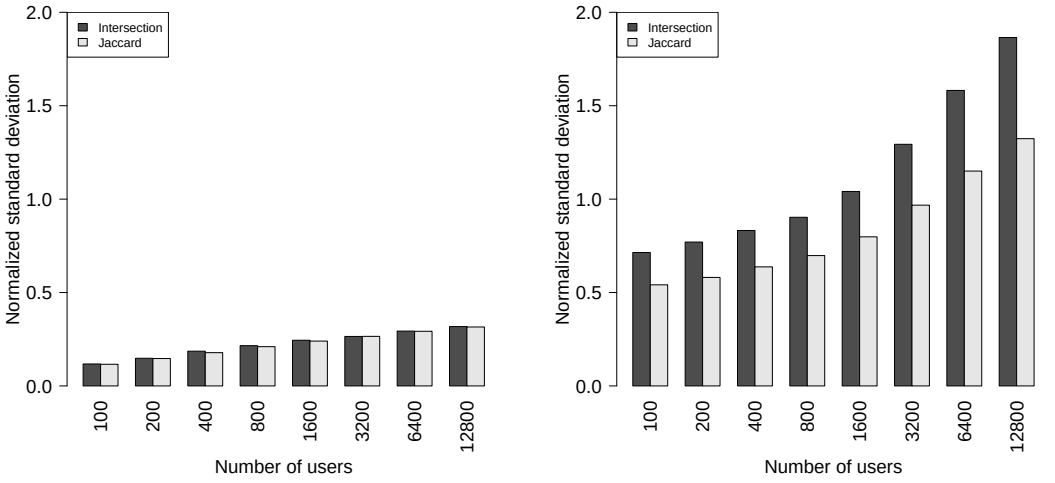
In the case of the user graph, we observe that the values of the intersection metric are reduced as the number of users increases, and the values of the Jaccard metric are marginally increased. However, in the case of the file graph, we observe that the values of the intersection increase with the number of users, and the Jaccard values have a marginal decrease. These results are explained by the fact that on a user graph, as the number of users increases, the probability of finding two users accessing the same content decreases. On the file graph, this situation is reversed as the number of vertices in the graph is fixed for all configurations.

To put into perspective the differences between both metrics, we measure the spread of the produced values. To obtain this measure, we use the normalized standard deviation (i.e., the standard deviation of the values divided by the mean value).



(a) User graph. (b) File graph.

Figure 14: Correlation between the intersection and Jaccard metrics using a configuration of 1,600 files and a varying number of users.



(a) User graph. (b) File graph.

Figure 15: Normalized standard deviation of the intersection and Jaccard metrics using a configuration of 1,600 files and a varying number of users.

Figures 15a and 15b show the values of this metric for the user and file graphs respectively. In the case of the user graph, the normalized standard deviation is slightly higher when using the intersection metric. In the case of the file graph, this difference becomes noticeable, with values up to 40% higher when using the intersection metric.

As a conclusion, we focus our study on condensed graphs generated with the Jaccard metric. This metric has demonstrated to produce weights with a limited spread, and a similar behavior in terms of clustering than the intersection metric.

4.3.2 *Metrics*

This section presents the metrics used in our evaluation. In this thesis, we explore different metrics and try to understand which metric or combination of metrics reflects better the quality of a certain partitioning.

To measure the quality of a partition, we employ a combination of metrics that come from different fields: *number of communities* and *modularity* from the community detection field; *clustering coefficient* from social network analysis; and *number of vertices per community*, *number of assigned resources per community*, and *commonality level* that are relevant for data distribution infrastructures. A detailed description of each metric is described in the following paragraphs.

- *Number of communities* ($|VC|$): One important aspect of a community detection algorithm is the number of partitions of the network. This number is determined by each of the algorithms based on the partitions they are able to discover. Therefore, the implementation of the algorithms and in particular the stopping conditions employed play a fundamental role. The number of possible partitions in a graph is bounded in the range from 1 to the number of entities in the graph. In our case, we target to find a number of communities in the first quartile of this range, as there is a tradeoff between the number of proxies in the system, and the system efficiency.

- *Modularity* ($\mathtt{mod}_{VC}$): The modularity metric is commonly used for evaluating community detection algorithms. It is defined as the number of edges inside each community minus the expected number of edges found in an equivalent random network. The utilization of this metric is subject to controversy due to the underlying comparison with a random graph. The main issue is that is it not clear what is the modularity value of a good partitioning. While some authors [79] conclude that significant communities are detected with modularity values larger than 0.3; other authors [8] also notice the existence of *anti-modularity* networks (modularity < 0), that also provide a good partitioning. In this aspect, we want to investigate the significance of the metric when applied to a data distribution scenario.

- *Clustering coefficient* ($\mathtt{clustering}_{VC}$): The clustering coefficient [122] measures how well the neighbors of a node are connected among themselves when that node is removed from the graph. In this work, we employ the clustering coefficient to understand the shape and connectivity inside each partition.

- *Number of vertices per community* ($\mathtt{vertices}_{VC}$): Depending on the number of communities detected in a particular network, this metric provides a measure of the number of entities associated per community. It is an indicator of how different in size are the partitions produced by the community detection algorithms.
- *Number of assigned items* ($\mathtt{assigned}_{VC}$): Similar to the number of vertices per community, we are interested in measuring which is the number of items assigned to a partition. In the case of analyzing a user graph, this metric determines how many files are associated with each community.
- *Commonality* ($\mathtt{com}_{VC}$): As introducing proxies in a network requires replicating common content over different servers, this metric measures the number of items that are present in all communities. Considering a user graph, the metric will provide the number of files that are present in all communities. This metric also reflects how many popular items exist in the network, and whether the partitioning process replicates them in all communities, or isolates them. Additionally, to provide a finer grain measure of the common files in different communities, we also measure the pair-wise commonality ($\mathtt{com}_{VC}^{pw}$). Using this measure, we obtain the average number of files that are shared between pairs of communities.

The use of different metrics is motivated by the fact that it is not clear a priori how to measure the quality of a partition when the algorithms are applied to data distribution scenarios. In this aspect, we want to determine which is the metric or combination of metrics that are able to capture the relevant characteristics and offers the best results.

4.4 EXPERIMENTAL EVALUATION

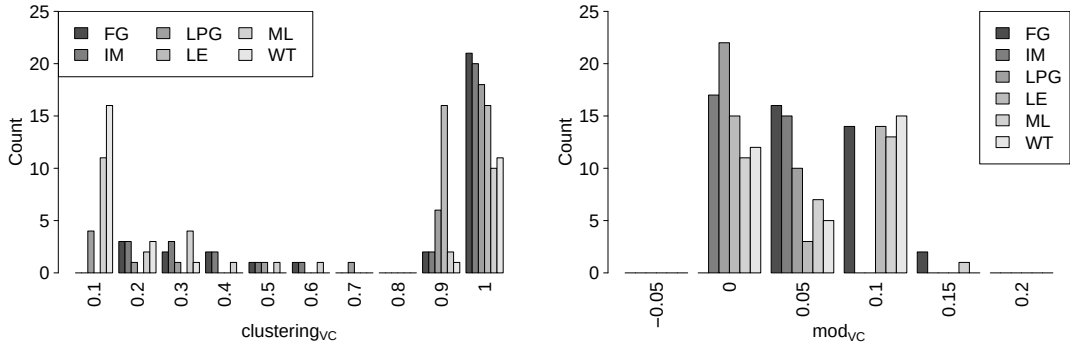
This section presents the evaluation of the different community detection algorithms when applied to the dataset defined in Section 4.3.1. The experimental results are obtained using the IGraph [5] library that provides state-of-the-art implementations of the selected community detection algorithms. To simplify the notation, we abbreviate Fast Greedy by *FG*, Walktrap by *WT*, Label Propagation by *LPG*, Leading Eigenvector by *LE*, Multilevel by *ML*, Infomap by *IM*, and Iterative Communities by *IC*.

The section is organized as follows. First, we evaluate the clustering coefficient and the modularity of the detected communities. Second, we analyze the quality of the partitions by evaluating the number of assigned vertices and items. After that, we evaluate the commonality of the partitions. Then, we evaluate the advantages of using the proposed iterative community detection algorithm. Finally, we explore the effect of varying the threshold values during the definition of the condensed graphs.

4.4.1 *Modularity and clustering coefficient*

In this section, we analyze the results obtained using the clustering coefficient ($\mathtt{clustering}_{VC}$) and modularity ($\mathtt{mod}_{VC}$) metrics. Both metrics provide information

5 Python IGraph library 0.6 igraph.sourceforge.net/



(a) Clustering coefficient

(b) Modularity

Figure 16: Modularity ($\mathtt{mod}_{VC}$) and clustering coefficient ($\mathtt{clustering}_{VC}$) for different input graphs and community detection algorithms.

regarding the internal structure of a community. The clustering coefficient measures how well connected are the neighbors of a node, when that node is removed from the subgraph. Figure 16a shows the histogram with the average clustering coefficient values.

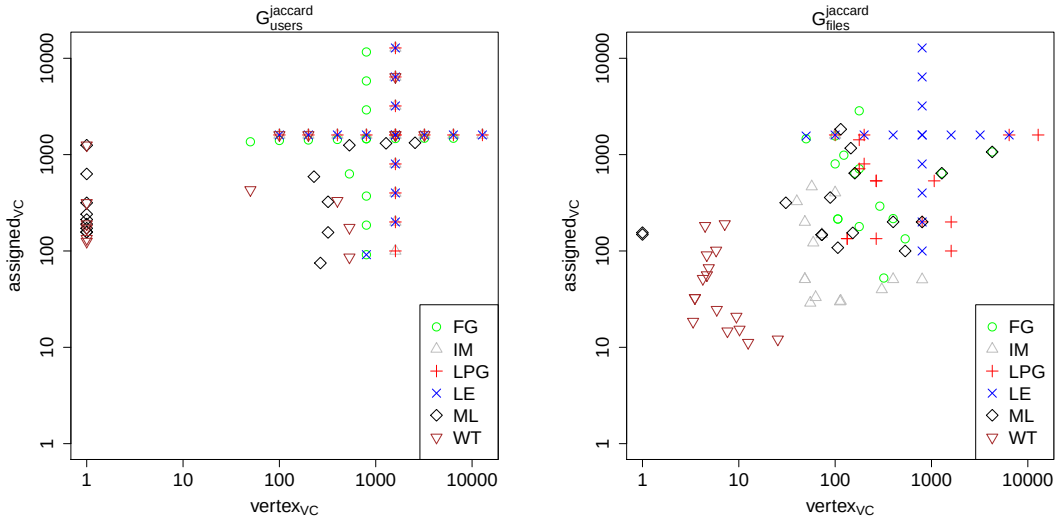
The results show that the distribution is highly skewed to high clustering coefficient values. The clustering coefficient values are concentrated on the interval [0.9, 1] for *FG*, *IM*, *LPG* and *LE* with median values between 0.93 and 0.99, which translates in the existence of highly connected communities. These results are related with the shape of the input graphs, and the ratio of edges per vertex observed when analyzing the characteristics of the dataset.

The modularity metric provides another point of view to analyze the shape of the detected communities. Figure 16b shows the histogram with the modularity values. In this case, the distribution tends to the values near zero with a median modularity of 0.00145. These results indicate that after partitioning the graph into a set of communities, each community shares the same characteristics of a random graph independently of the underlying community detection algorithm.

Taking into account the values of $\mathtt{clustering}_{VC}$ and $\mathtt{mod}_{VC}$ we conclude that these types of graphs exhibit a high degree of connectivity and that the connections are established using a random distribution. These observations have a significant impact on the performance of the community detection algorithms. The high values of clustering coefficient imply that, when building a partition, there will be a large number of edges to be analyzed. Moreover, the algorithms based on modularity-like metrics may require more steps to determine which partitions must be defined at each step as the gain from adding a vertex is marginal. In this sense, algorithms such as *Infomap* that use other type of metric may benefit and require less computational effort to produce the partitions.

4.4.2 *Number of assigned elements to a partition*

In this section, we analyze the results in terms of the number of items and users found in each partition. Figure 17 shows the relationship between the number of



(a) User graph condensed with the Jaccard measure.

(b) File graph condensed with the Jaccard measure.

Figure 17: Relationship between number of assigned elements and number of vertices per partition for different graph configurations and algorithms.

vertices ($vertices_{VC}$) contained in a partition and the number of elements assigned ($assigned_{VC}$) to that partition for all the input graphs of the dataset (Section 4.3.1). From the point of view of a data distribution infrastructure, we are interested in finding algorithms that produce a small to medium number of users with a small to medium number of elements assigned to a partition. If the partitions contain a large number of users, we face the risk of overloading the server. Similarly, if the partitions contain a large number of assigned elements, we face the risk of replicating a significant number of files in the system and impacting the required storage space.

Figure 17a shows the results of a user graph condensed using the Jaccard metric. We can distinguish two clusters of points in the figure: 1) medium to large number of assigned elements with only one vertex, and 2) a medium-to-large number of vertices and assigned elements.

The first cluster corresponds to degenerated cases produced by *ML* and *WT*. Theses cases illustrate one significant limitation of some clustering algorithms. Given an input graph with $|U|$ vertices, the algorithms try to find the partition that maximizes the value of the resulting modularity. If the graphs are highly connected, it may occur that the decision taken by the algorithm to improve the modularity is to define two partitions: one with a single entity, and other with $|U| - 1$ entities. As it is not possible to tune this behavior, the only practical solution is to employ other community detection algorithms in these situations.

The second cluster corresponds to the majority of the cases. A set of communities with 40 to 12,800 vertices and 80 to 12,800 assigned elements is produced. From this set of observations, we observe that the best tradeoff between $vertices_{VC}$ and $assigned_{VC}$ is produced by *WT* and *ML*. While these results may be considered contradictory with the previous observations, they indicate another important aspect when selecting a community detection algorithm. From our results, we can conclude

that the quality of the partitions in terms of $vertices_{VC}$ and $assigned_{VC}$ produced by some algorithms is highly influenced by the input graph. This suggests that some parts of the graph may show a structure that is difficult to process by the algorithm, and subsequently the partitioning process is decided by a marginal gain on modularity.

By contrast, Figure 17b shows the results for a file graph condensed with the Jaccard metric. In this case, we observe two significant differences. First, the number of degenerated cases has been reduced. Only two partitions produced by *ML* are observed, and *WT* no longer generates any single-entity community. Second, the shape of the main cluster has significantly changed. On the small to medium range, *WT* produces communities between 3 and 30 vertices with 10 to 190 assigned elements. The other algorithms produce communities starting with 30 vertices and 25 assigned elements to 12,800 vertices and elements. Based on this observations, the best performing algorithm is *WT* followed by *IM* and *ML*.

Considering the results of both types of graph, we conclude that even though some algorithms perform better than others, their results vary based on the underlying structure of the input graph (connectivity degree). Additionally, all studied algorithms lack an important characteristic: how to tune the algorithms to produce communities under certain constraints in terms of number of elements and entities.

4.4.3 *Commonality*

In this section, we explore the similarity between the items assigned to different virtual communities. To study this characteristic, we employ the commonality (com_{VC}) and pair-wise commonality (com^{pw}_{VC}) of the resulting communities.

Measuring these characteristics provides an important insight regarding how the items on the network should be distributed among the detected virtual communities. Large values of com_{VC} imply that all communities behave like mirrors and therefore a large number of items must be replicated in all the system. Similarly, large values of com^{pw}_{VC} imply that part of the content must be replicated in at least some part of the existing communities. With this metric we are able to detect situations when a portion of the items must be replicated in a subset of communities. Small values of com_{VC} or com^{pw}_{VC} translate in the existence of content that is only consumed by a fraction of the entities.

Figure 18 shows the relationship between com_{VC} and com^{pw}_{VC} for the different community detection algorithms. Two conclusions can be drawn from this figure. First, we observe that zero or close-to-zero values of com_{VC} may produce values of com^{pw}_{VC} up to 0.4. This behavior matches the expectations and it is more likely to find two communities that share at least a common file, than requiring that file to be present in all communities. Second, we observe a high correlation between both metrics for values larger than 0.5. In particular, *ML*, *WT*, and *FG* produce the set of communities with larger values of commonality. This translates in the existence of communities that mirror each other in terms of the content server to their users.

To understand the commonality value distribution, Figures 19a and 19b show the histogram for com_{VC} and com^{pw}_{VC} respectively. We observe that the value distribution of both metrics concentrates on the minimum (0) and maximum (1) possible values independently of the selected algorithm. In this respect, the only significant

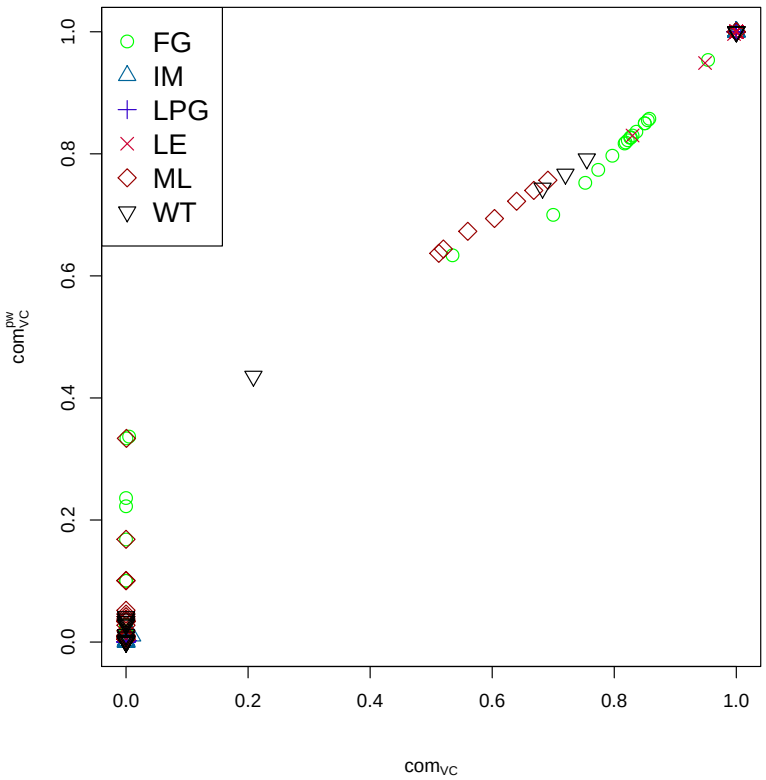
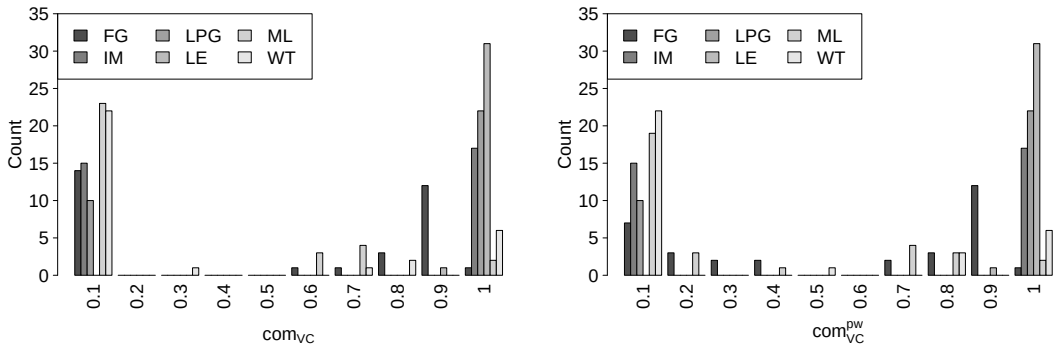


Figure 18: Influence of the community algorithm in the commonality metrics.



(a) Commonality histogram.

(b) Pair-wise commonality histogram.

Figure 19: Study on the commonality metrics for different community detection algorithms.

ALG.	$\|VC\|$	mod_{VC}	$clustering_{VC}$	$vertices_{VC}$	$assigned_{VC}$	com_{VC}	com^{pw}_{VC}
FG	2	0.06	**0.98**	400	1026.88	**0.73**	**0.73**
IM	1	-10^{-7}	**0.99**	352.38	401.5	1	1
LPG	1	-10^{-6}	**0.97**	1333.33	1512.39	1	1
LE	2	0.02	**0.93**	800	1600	1	1
ML	**14.5**	0.01	0.21	**110.48**	**226.44**	**0**	**0.040**
WT	199.5	0.03	0.10	6.52	**129.81**	**0**	**0.020**

Table 2: Median values of each of the studied metrics per algorithm.

difference is the fact that there are more communities with zero com_{VC}, and more communities with a com^{pw}_{VC} of one. This matches the expected behavior observed in the previous figures.

In summary, we find that the resulting communities are a mix between a set of communities that have the same items assigned (mirrors), and a set of communities that are the only ones serving a particular content (proxies). Regarding the community detection algorithms, *ML* and *FG* obtain the larger commonality values. In this respect, it should be notice that the ideal value of the commonality depends on the application scenario. Some applications may prefer generating several exact mirrors while other may prefer proxies without any type of overlapping in terms of the stored files.

4.4.4 *Effect of the algorithm selection*

In the previous sections we have studied the results obtained by different community detection algorithms on several aspects such as the relationship between the number of vertices in a community ($vertices_{VC}$) compared with the number of assigned items ($assigned_{VC}$). In this section, we concentrate on the results obtained by each algorithm on each of the metrics used in the evaluation and try to define which algorithms provide the best overall performance across all metrics.

To illustrate the discussion in a manageable way Table 2 shows the median value of each metric from the results obtained when applying each algorithm to our evaluation dataset. Due to the significant variability across different scenarios, we use the median value instead of the mean value as a measure the expected value. The values of the best performing algorithms on each metric have been highlighted.

In the ideal case considering a data distribution scenario, a suitable algorithm should produce a set of virtual communities that have: 1) a well-balanced number of vertices and assigned items per community, 2) a high clustering coefficient.

To study these characteristics, we first address the number of virtual communities ($|VC|$), the number of vertices per community ($vertices_{VC}$), and the number of assigned items ($assigned_{VC}$). All of these metrics are related as the number of de-

tected virtual communities will affect the number of vertices, and depending on the algorithm we expect the number of assigned items to vary. This is due to the fact that different algorithms choose differently which nodes are assigned to the same virtual community. The results of |VC| illustrate one of the main problems experienced by some algorithms: the inability to produce partitions on some of the input networks. As shown in the table, *FG*, *IM*, *LPG*, and *LE* are only able to produce at most 2 partitions of the network. Considering the values of |VC| and $vertices_{VC}$, *ML* produce the best results. In the case of $assigned_{VC}$ both *ML* and *WT* produce the best results.

Regarding the clustering coefficient ($clustering_{VC}$), *FG*, *IM*, *LPG* and *LE* produce the best results maintaining a high clustering coefficient on the resulting partitions. Notice that in our case, we do not take into account the value of the modularity (mod_{VC}) for the selection of the best algorithm. Our results indicate that all algorithms produce communities that show random-graph characteristics, and therefore this metric is not relevant for our purpose.

Finally, we study the values of the commonality (com_{VC} and com_{VC}^{pw}) and found that *FG*, *ML* and *WT* produce the best values for scenarios that aim to avoid mirrored servers. On the other hand *IM*, *LPG*, and *LE* produce mirrored communities independently of the number of assigned items.

To summarize our analysis, *ML* (Multilevel [18]) produce the best overall results with a small number of communities, a balanced number of assigned items, producing non-mirrored partitions.

4.4.5 *Iterative community detection algorithm*

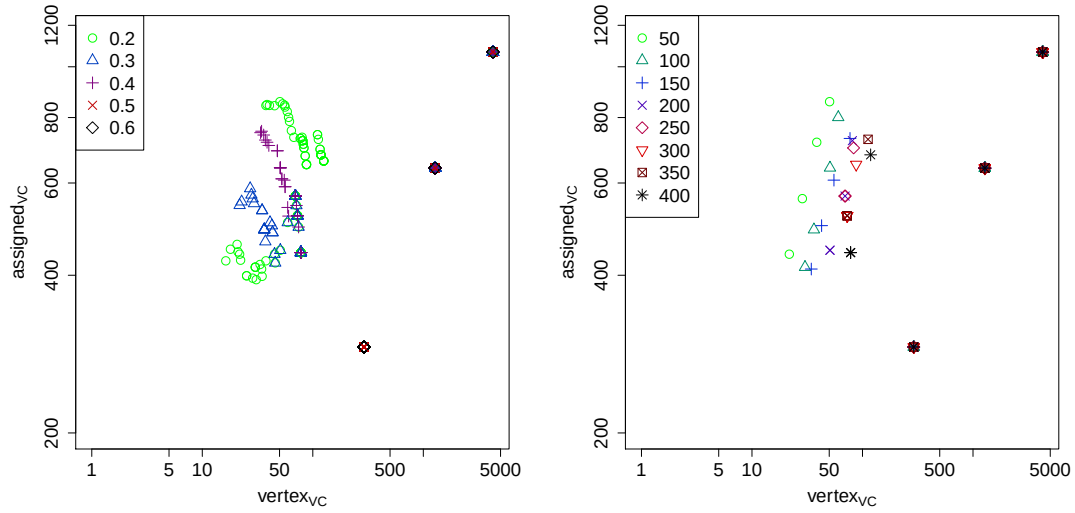
The results obtained in the previous analyses demonstrate the influence of the underlying community detection algorithm. In this section we explore the results obtained using the iterative community detection algorithm proposed in this thesis. This algorithm allows the user to specify two parameters that permit to control the characteristics of the resulting communities.

In the evaluation, we focus our study on graphs with 1,600 users and 3,200, 6,400 and 12,800 files. Using this configuration we evaluate the impact of varying the number of minimum elements per community ($min_{VC}^{elements}$), and the ratio of entities per community ($ratio_{VC}^{entities}$). We employ FG as the underlying community detection algorithm as previous tests demonstrate that is one of the best performing algorithms overall.

Figure 20a shows the results of varying $ratio_{VC}^{entities}$ to 0.2, 0.3, 0.4, 0.5, and 0.6. As expected, the number of assigned elements per community decreases as the ratio of entities per community decreases. However, as the ratio is used to decide whether a community should be split, the number of entities remaining in the resulting communities is less or equal than the established ratio.

From this figure, several conclusions can be drawn. First, we notice the existence of different bands especially for $ratio_{VC}^{entities}$ values of 0.2 and 0.3. This is explained by the fact that the figure contains information from different sized graphs and therefore the minimum number of entities changes accordingly.

Second, we observe that in some cases, different ratios produce the same results. If we focus on the results setting the ratio of entities to 0.5 and 0.6, we observe that both produce the same results. In detail, both produce 11, 5 and 3 communities with



(a) Influence of the $ratio_{VC}^{entities}$ parameter.

(b) Influence of the $min_{VC}^{elements}$ parameter.

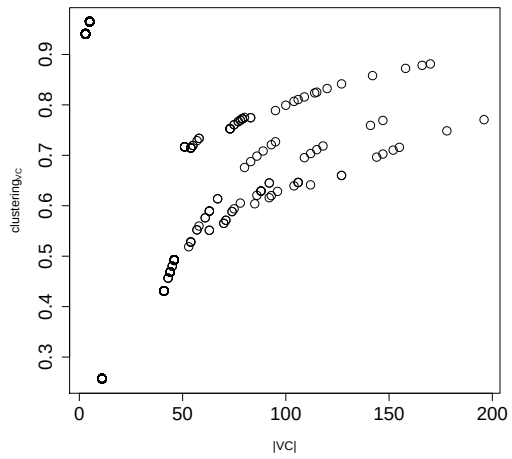
Figure 20: Evaluation of the iterative algorithm varying the $ratio_{VC}^{entities}$ and the $min_{VC}^{elements}$ parameters.

and $vertex_{VC}$ values of 290, 1280 and 1267; and $assigned_{VC}$ values of 291, 640 and 1067 depending on the size of input graph (3200, 6400 or 12800). This is explained by the fact that the number of iterations of the algorithm is influenced by this parameter, and both algorithms stopped at the same time as the number of assigned elements matched the expected limits.

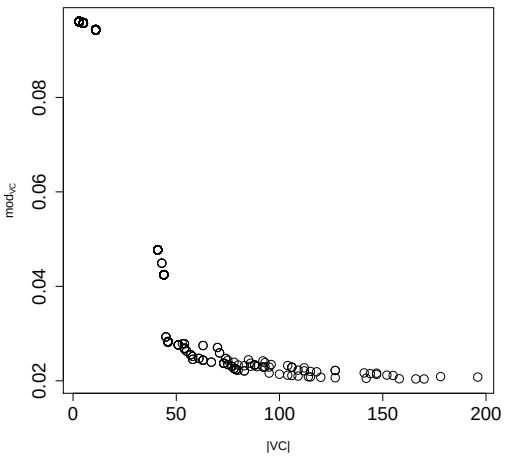
Figure 20b shows the results of varying $min_{VC}^{elements}$ to 50, 100, 150, 200, 250, 300, 350, and 400. The results show that the number of elements increases as expected with the number of $min_{VC}^{elements}$. As shown before, some values of the parameter may produce the same results due to the fact that they share the same branch structure when the iterative algorithm is executed.

Additionally, there are other metrics that provide a detailed view of the impact of using the proposed algorithm. First, in Figure 21a we analyze the relationship between clustering coefficient and the number of communities. The results show that as the clustering coefficient increases with number of communities. This result translates into the fact that as the communities decrease in size, they become more connected.

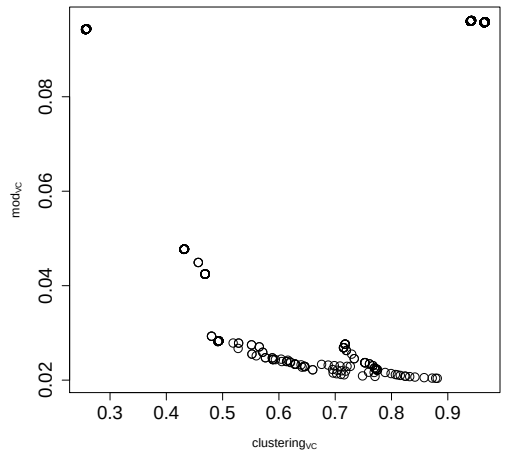
In Figure 21b we analyze the relationship between the number of communities and the modularity. As expected, the modularity decreases with the number of communities as it becomes more difficult to find a single file assigned to all communities. In Figure 21c we study the relationship between the clustering coefficient and the modularity. As shown before with the number of virtual communities, the modularity decreases with the clustering coefficient. Finally, Figure 21d shows the relationship between the number of assigned elements and the pair-wise commonality. The results show that the pair-wise commonality increases with the number of assigned elements per community. This behavior is expected, as when the number of assigned items decreases, it becomes more difficult to find other communities with the same assigned items.



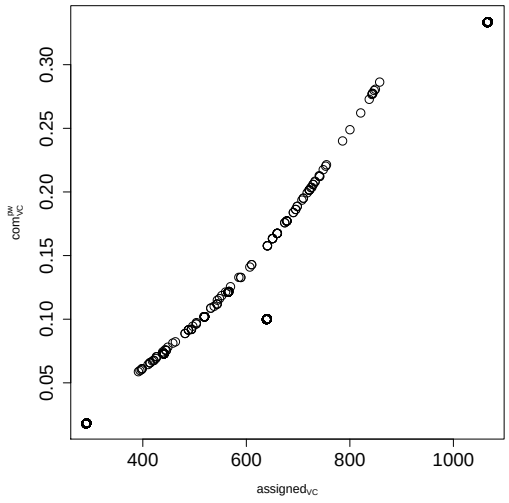
(a) Relationship between the number of communities and the clustering coefficient.



(b) Relationship between the number of communities and the modularity.



(c) Relationship between the clustering coefficient and the modularity.



(d) Relationship between the number of assigned elements and the pair-wise commonality.

Figure 21: Different measures of the iterative algorithm.

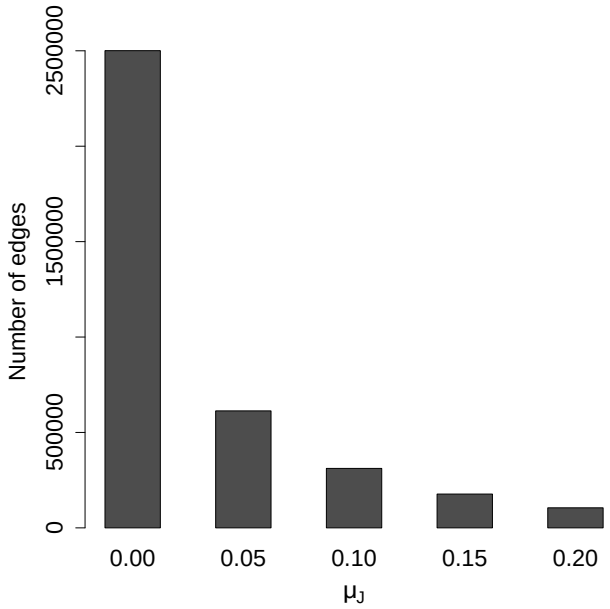


Figure 22: Size of the condensed graph attending to the weight threshold value.

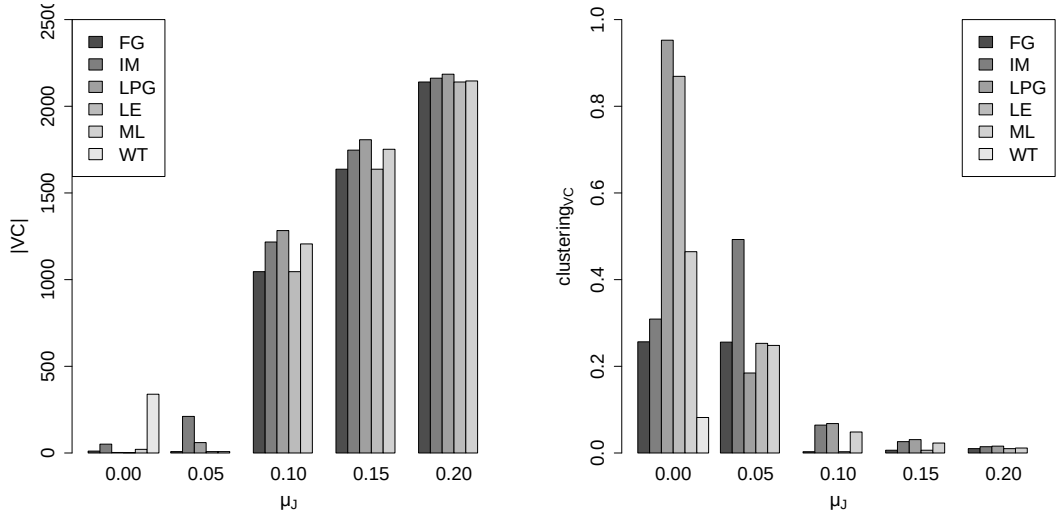
In summary, the results demonstrate that our proposed iterative community detection algorithm is able to shape the resulting virtual communities to the user specifications.

4.4.6 *Effect of the weighting threshold*

In this section we study the effect of changing the weighting threshold (μ_J) on the resulting graphs. The threshold permits to determine the minimum value of the Jaccard measure to consider an edge to be meaningful. In this way, all edges whose weights are smaller than μ_J are not added to the condensed graph. In practice, this reduces the number of edges in the graph and facilitates the execution of the community detection algorithms in terms of computational complexity. For clarity purposes, our study focuses on a file graph with 1,600 users and 3,200 files. However, the results can be extended to other input graphs of the data set. To show the impact in the size of the graph, Figure 22 shows the number of edges in the condensed graphs for values of $\mu_J = 0.00, 0.05, 0.10, 0.15, 0.20$.

The results show a significant decrease in the number of edges compared with the initial condensed graph. The use of $\mu_J = 0.05$ results in a reduction of 75% in the number of edges, reaching up to 95% for $\mu_J = 0.20$. This decrease results in a reduction in the time required by the algorithms to produce the partitions as the number of edges to be explored is reduced.

However, it is important to understand what factors may be affected by this situation. In particular, we analyze the number of communities (Figure 23a) and the clustering coefficient (Figure 23b) using the different community detection algorithms for various values of μ_J.



(a) Number of communities.

(b) Clustering coefficient.

Figure 23: Number of communities and clustering coefficient attending to the weight threshold value.

Regarding the number of virtual communities, the results show a significant increase in the number of resulting communities. This effect is not desired as it also implies that the number of entities per community is drastically reduced. For example, with $\mu_J = 0.10$, the number of communities is multiplied a factor between 23 to 427 depending on the algorithm. Additionally, we notice that the *WL* algorithm was not able to produce any partition for non-zero values of μ_J. To explain this outcome, we must consider other effects of increasing the threshold value. In particular, as the value is increased, the likelihood of finding non-connected components in the graph is increased. This situation leads to two problems. First, it demonstrates that non-zero values of the weighting threshold may result in a loss of information in the resulting graph. As a consequence, some entities may be reassigned to non-optimal communities. Second, it limits the use of algorithms that require a connected graph such as *WL*.

To support our observations, we analyze the clustering coefficient of the resulting communities in the refined condensed graphs. The results of the figure are in concordance with the previous observations as the clustering coefficient is dramatically reduced as the values of μ_J increase. Taking into account the effects of increasing the value of μ_J, we conclude that non-zero values reduce the amount of useful information in the conduced graph and increase the risk of obtaining non-connected components. Therefore, its use should be limited to situations where the removed information is not relevant.

4.5 SUMMARY

In this chapter, we have presented the community detection module of the architecture. The module analyzes the relationships between the users of the system and the

files they access, and generates a set of virtual communities. The main objective of this module is to find a method that provides partitions that satisfy three characteristics: 1) contain a minimum number of users, 2) have a maximum number of files assigned to the partition, and 3) the set of files shared on all partitions should be minimal.

In order to obtain the virtual communities we use a set of well-known community detection algorithms. As the existing algorithms only support input graphs with a single type of entity, in this thesis we propose a technique to condense the input data distribution graph to contain a single type of entity (e.g., files or users). In this respect, we experiment with two similarity metrics to establish the weights on the condensed graph, namely intersection and Jaccard, and we have found that the Jaccard measure produces the best results.

The existing community detection algorithms automatically define the partitions based on the resulting structure of the graph. However, they do not allow any type of user input to guide the partitioning process. To address this limitation, we propose a new iterative community detection algorithm. Our algorithm uses any of the existing community detection algorithms, and guides the partitioning process based on the number of entities and assigned items in the resulting communities.

Afterwards, we concentrate in the evaluation of the performance of the selected algorithms. In this respect, we use a set of metrics related with community detection methods and data distribution systems: number of communities, modularity, clustering coefficient, number of vertices, number of assigned items, and commonality. Several conclusions can be drawn from our evaluation. First, we show that the existing algorithms perform poorly when partitioning highly connected graphs. Our results suggest that this difficulty is influenced by the underlying modularity metric used to guide the partition process. Second, the analysis of the partitions produced by the different algorithms show that the results are influenced by the number of edges and vertices of the underlying graph, and the quality ranking is not maintained for different configurations. Third, we study the common items found in the partitions produced by different algorithms and the results suggest that the virtual communities obtained can be categorized into two groups: a set of communities that are mirrors of other in terms of assigned items, and a set of communities that partially share some content with a small number of communities. Fourth, we show how our proposed iterative community detection algorithm allows the user to guide the partitioning process according to the desired configuration of the resulting communities.

Finally, we study the effect of filtering the edges of the condensed graph according to a minimum weight threshold. In this respect we find that the weighting threshold has a significant impact on the resulting communities, and the loss of useful information can lead to the appearance of non-connected components in the data distribution graph.

5

DATA TRANSFER SCHEDULING

The last years have shown a continuous increase in the volumes of data to be stored and retrieved on-line. As this evolution is predicted to continue, large-scale data distribution systems face considerable challenges in assuring quality of service, while keeping the infrastructure costs at profitable levels. Consequently, achieving a high resource utilization has become of utmost importance for every large scale data provider. In this respect, data transfer scheduling appears as a promising solution to control the traffic found distribution scenarios. The scheduling of transfers brings two main benefits: it permits to optimize the required number of servers at any moment, and it permits to adapt the quality of service perceived by the users of the system.

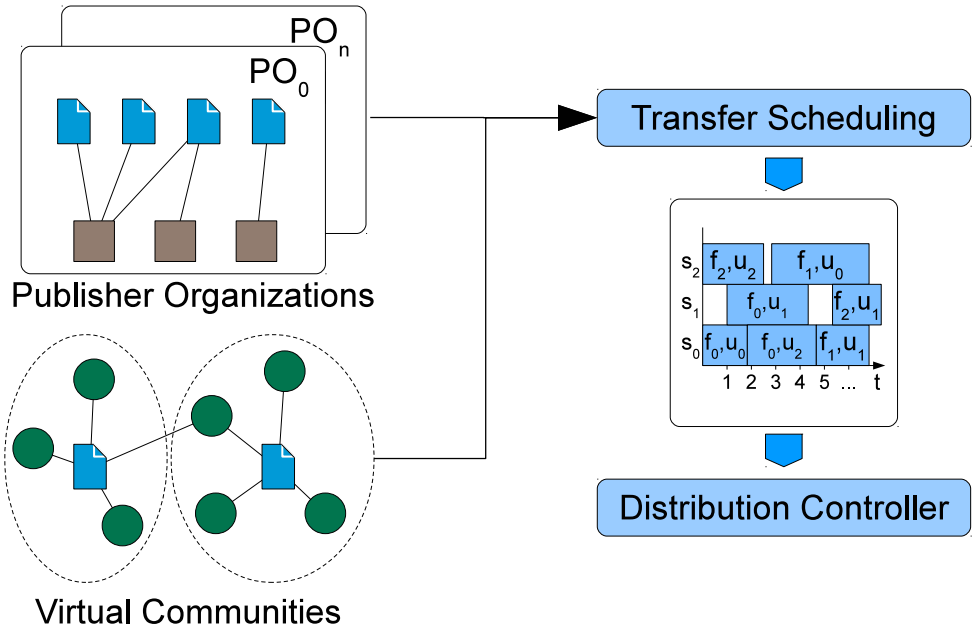


Figure 24: Generation of a transfer schedule in a data distribution scenario using the requests assigned to virtual communities, and the current file to server allocation on the publisher organizations.

In this chapter we address the problem of finding an efficient data transfer schedule for multi-server and multi-user scenario. In particular, we address the problem of transferring a set of files to a set of users, given a certain number of servers, under several constraints such as maximum available bandwidth. We propose a solution that produces a schedule that delivers all files to all users, while reducing the

schedule length and maximizing the server utilization. The component responsible of providing this functionality is the Transfer Scheduling module (Figure 24) of the architecture presented in Chapter 3.

There are three main objectives that we target: a) to minimize the schedule length, b) to maximize the file server utilization, and c) to find the schedule in a computationally tractable way. Fulfilling all these objectives is a challenging task, as only calculating the optimal schedule in a multi-server multi-user environment can be shown to be NP-complete. In order to address this issue, we seek to find a practical balance among these objectives.

Our solution is based on the relaxation of an objective-based time-indexed formulation of a linear programming problem. In order to speedup the solving process, we propose a method to distribute the computation process using the map-reduce paradigm. Additionally, we allow the architecture administrator to control the trade-off between the quality of the produced solution and the time required to produce the solution.

5.1 TRANSFER SCHEDULING PROBLEM

The transfer scheduling problem can be stated as an optimization problem as follows. Given a set of servers S, a set of files F distributed over S, and a set of requests R (as a union of tuples $< dst, f >$ representing requests from user dst to file f), what is a schedule of optimal makespan? Figure 25 shows an overview of the transfer scheduling setup. In a generic scenario we distinguish two sides of the data distribution problem. On the left-hand side, the user destinations request a subset of files. These destinations correspond to the virtual communities produced by the Community Detection module. However, if required, the user destinations may correspond to single users requesting particular files. On the right-hand side, the files are stored across the servers of the different publisher organizations. In this context, the problem consists of determining which transfers should be made between the servers and the user destinations considering the time required to transfer a file between different entities and the existing requests. Taking this scenario as the basic problem, several improvements can be added by reformulating the model to support additional constraints such as the server available bandwidth, number of requests served at a time, etc.

Different base problems can be used to model the file transfer scheduling problem: graph-coloring, job shop problem, maximum flow network, etc. In this thesis we choose to formulate our base model as an extension of the open job shop problem [82]. This problem consists of n independent jobs to be processed by m parallel machines. Each job i is composed of a set of operations. The operation O_{ij} over job i is processed by machine j for p_{ij} time units. All operations of a job have to be processed on all machines. Only one operation can be executed per machine per unit of time and the operations do not have any order constraint.

The remainder of this section describes the formulation of our baseline model based on the open job shop scheduling problem and discusses the complexity of this approach. Table 3 summarizes the variables and parameters used in the formulation of the baseline model.

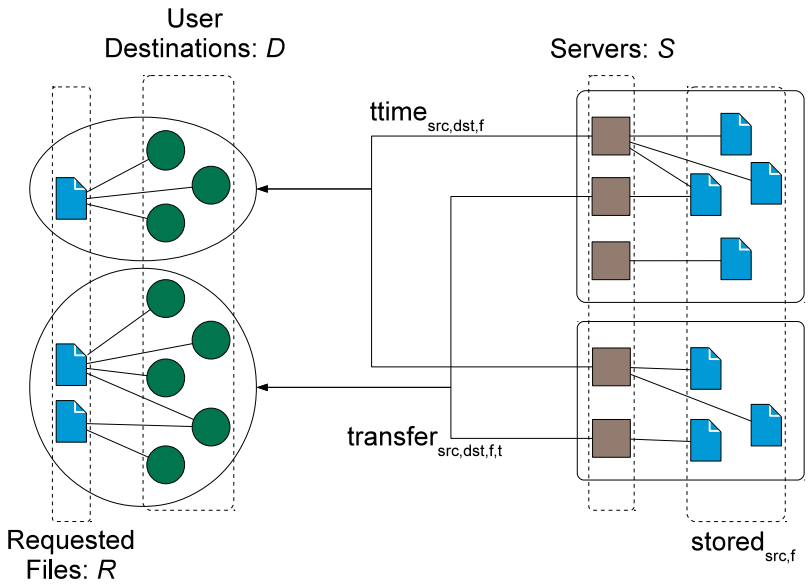


Figure 25: Elements involved in data distribution scenario.

NAME	DESCRIPTION
S	Set of servers
D	Set of requesting destinations
F	Set of files
R	Set of request tuples $< dst, f >$
M	Set of all machines $M = S \cup D$
T	Time slots from $t = 0...Tmax$
$maxtt$	Maximum transfer time for a single file.
$Tmax$	Maximum time to perform all transfers.
$ttime_{src,dst,f}$	Transfer time of file f from src to dst.
$stored_{src,f}$	Boolean representing if file f is stored in server src.
$transfer_{src,dst,f,t}$	Boolean representing if a transfer of file f is in progress at time t from src to dst.
$finish_{dst,f}$	Time when file f is available at dst.

Table 3: Baseline model variables and parameters

5.1.1 *Baseline model*

For our baseline model we adapt the open job shop scheduling problem by considering an operation as a file transfer $< dst, f >$, and a job as the total set of transfers for a given file (i.e., $< dst_1, f >$, $< dst_2, f >$, ...,$< dst_k, f >$). Our approach departs from the original open job problem definition in three respects. First, a job does not have to be processed by all the machines, i.e., a file is not to be transferred by all servers. Second, we add a constraint on the transfer time of each file based on the available bandwidth of each machine. Third, our formulation allows for scheduling a user request to an idle server that is not storing a file. In this case, the server incurs a miss and a request is scheduled for fetching the file from a server storing it.

To simplify the model representation, we define $M = S \cup D$ as the set of all machines in the system. The parameter $ttime_{src,dst,f}$ represents the time required to transfer a file f from the machine src to dst, when taking into account the available bandwidth of each machine. The binary parameter $stored_{src,f}$ is 1 if and only if the file f is stored in server src. Time constraints are introduced with the parameter $maxtt$, which is the maximum value of $ttime$. $Tmax$ represents the maximum time allowed to perform all transfers.

Finally, we introduce the set of variables that represent the solution to the scheduling problem. The transfer progress is modeled as a time-indexed formulation, a very popular representation for scheduling problems [62, 12]. The binary variable $transfer_{src,dst,f,t}$ takes the value 1, when a transfer of file f from src to dst is active at time t. The integer variable $finish_{dst,f}$ represents the time when file f is available at machine dst.

Equations (6) to (16) show the formalization of the baseline model. The objective of this optimization problem (6) is to minimize the schedule length (makespan), i.e., the total time required to perform all transfers. Equation (7) ensures that all requested files eventually become available to the requesting machines at time $finish_{dst,f}$. Files requested to a server that stores them are available locally according to Equation (8). Equation (10) prevents transfers among users. These transfers would result in permiting peer-to-peer functionality, but would also further increase the solver complexity. In (11) we assure that destination machines receive the requested files. Equation (12) expresses the fact that for any destination there is only one active transfer at a given time. Equation (13) assures the download process to take into account transfer times and file availability at the source. Equation (14) enforces that a file is transferred only when it is available at the source server. Equations (15) and (16) limit the number of transfers a machine is involved at a given time.

5.1.2 *Baseline model discussion*

The utilization of the aforementioned model provides optimal solutions for the file transfer scheduling problem. However, the model is difficult to use in practice in its original form. In particular, the open job shop problem is a known NP-complete problem [67]. If we assume a scenario, in which transfers among servers are not allowed, each server only provides one file and all users request all files, we can rewrite the transfer file scheduling problem like an open job shop problem. Therefore, we can claim that our baseline model is NP-complete. Additionally, the presented

Minimize: $$\sum_{m \in M, f \in F} finish_{m,f} \tag{6}$$

Subject to:

$$finish_{dst,f} = \sum_{\substack{0 \leqslant t \leqslant Tmax, src \in M \\ src \neq dst}} transfer_{src,dst,f,t} \cdot t$$

$$\forall f \in F, dst \in M \tag{7}$$

$$stored_{src,f} = 1 \Rightarrow finish_{src,f} = 0 \qquad \forall s \in S, f \in F \tag{8}$$

$$\sum_{t \in T} transfer_{m,m,f,t} = 0 \qquad \forall m \in M, f \in F \tag{9}$$

$$\sum_{dst \in M, t \in T} transfer_{src,dst,f,t} = 0 \qquad \forall f \in F, src \in D \tag{10}$$

$$\sum_{src \in S, t \in T} transfer_{src,r.dst,r.f,t} = 1 \qquad \forall fr \in R \tag{11}$$

$$\sum_{src \in M, t \in T} transfer_{src,dst,f,t} = 1 \qquad \forall dst \in M, f \in F \tag{12}$$

$$(transfer_{src,dst,f,t} = 1) \Rightarrow (t \geqslant (finish_{src,f} + ttime_{f,src,dst} + 1)) \qquad \forall src, dst \in M, src \neq dst, f \in F, t \in T \tag{13}$$

$$(transfer_{src,dst,f,t} = 1) \Rightarrow stored_{src,f} + finish_{src,f} \geqslant 1 \qquad \forall src, dst \in M, src \neq dst, f \in F, t \in T \tag{14}$$

$$(transfer_{src,dst,f,t} = 1) \Rightarrow \sum_{\substack{m \in M, p \in F \\ 0 \leqslant j \leqslant ttime_{f,src,dst}}} transfer_{m,dst,p,t-j} \leqslant 1 \qquad \forall src, dst \in M, src \neq dst, f \in F, maxtt \leqslant t \leqslant Tmax \tag{15}$$

$$(transfer_{src,dst,f,t} = 1) \Rightarrow \sum_{\substack{m \in M, p \in F \\ 0 \leqslant j \leqslant ttime_{f,src,dst}}} transfer_{src,m,p,t-j} \leqslant 1 \qquad \forall src, dst \in M, src \neq dst, f \in F, maxtt \leqslant t \leqslant Tmax \tag{16}$$

Figure 26: Baseline model for scheduling transfers.

formulation requires $O(|M|^2|F||T|)$ constraints and variables. Although M appears squared, the value of T is likely to be orders of magnitude larger. In practice, this means that the complexity of the scheduling model is more sensitive to the value of T. This huge space complexity adds an additional challenge to any linear programming solver, which is supposed to load the problem into main memory for the solving process.

Based on these observations, we define an alternative model, which addresses these issues in order to reduce the complexity and allow for solving the problem in a computationally tractable way.

5.2 ALTERNATIVE MODELING OF THE TRANSFER SCHEDULING PROBLEM

As presented in the previous section, using the baseline model to solve the transfer scheduling problem is computationally intractable. In this section we address this issue by transforming the baseline model in four main steps. First, we reformulate the baseline model as a relaxed feasibility problem in Subsection 5.2.1. Second, in Subsection 5.2.2 we propose a heuristic that acts as an approximation algorithm, which allows to relax the optimization problem based on a customizable factor. Third, we discuss how the solving process can be distributed in Subsection 5.2.3. Finally, we discuss the merging of the final solution in Subsection 5.2.4. Table 4 presents the new set of parameters and variables for the alternative model.

5.2.1 *Reformulation as a feasibility problem*

The complexity of the baseline model makes it non-suitable for real applications. To address this problem, we opt for reformulating the model as a feasibility problem. A feasibility problem consists of finding any feasible solution for a given problem, without regard to the objective function. Using a feasibility problem instead of an optimization problem reduces the total computation time as the solver only needs to return the first solution that satisfies the set of constraints.

The reformulation consists in three main parts. First, we remove the objective function (Equation 6). Second, we remove the time dimension from the model variables. Third, we generate one model per file instead of generating one global model for all files. This approach has the advantage that it allows to distribute the solving process, and, subsequently, merge the partial solutions into the final global schedule.

Table 4 summarizes the variables used to represent the feasibility model. In the feasibility model, the output for a single file schedule is represented by the variables $transfer_{src,dst}$, $start_{src,dst}$ and $finish_{src,dst}$. All integer variables can take values from 0 to $Tmax$. Expressions (17), (18), (19) are intermediate representations used for simplification. They represent the start and the finish of a transfer, and the availability of a file in a server, respectively.

Equations (20) to (28) describe the formulation of the feasibility problem. The constraint (20) enforces that all destinations receive the file they have requested. Constraint (21) sets the finish variable to 1 in servers that have already stored the file. Constraint (22) introduces precedence in the transfer order, i.e., no transfer can start

NAME	DESCRIPTION
S_f	Set of servers involved in the distribution of f
D_f	Set of destinations requesting f
M_f	Set of all machines $M_f = S_f \cup U_f$
$start_{src,dst}$	Start time of transfer between src and dst
$finish_{src,dst}$	Finish time of a transfer between src and dst.
$transfer_{src,dst}$	Boolean representing if a file has been transferred from src to dst.
$tstart_{dst}$	Start time of a transfer to dst.
$available_{dst}$	Finish time of a transfer to dst.
$cached_{dst}$	Boolean representing if the file is cached at dst.
$stored_{src}$	Boolean representing if file is initially stored in server src.
$order_{srv}$	Preferred transfer order.

Table 4: Feasibility model variables and parameters

$$tstart_{dst} = \sum_{src \in M_f, src \neq dst} start_{src,dst} \qquad \forall dst \in M_f \qquad (17)$$

$$available_{dst} = \sum_{src \in M_f, src \neq dst} finish_{src,dst} \qquad \forall dst \in M_f \qquad (18)$$

$$cached_{dst} = \sum_{src \in S_f, src \neq dst} transfer_{src,dst} \qquad \forall dst \in M_f \qquad (19)$$

Figure 27: Expressions used in the feasibility problem formulation.

$$\sum_{src \in S_f} transfer_{src,dst} = 1 \qquad \forall dst \in D_f \quad (20)$$

$$stored_{srv} = 1 \Rightarrow finish_{srv,srv} = 1 \qquad \forall srv \in S_f \quad (21)$$

$$(transfer_{src,dst} = 1) \Rightarrow (start_{src,dst} \geqslant available_{src}) \qquad \forall src \in S_f,\ dst \in M_f,\ src \neq dst \quad (22)$$

$$tstart_{m_1} \geqslant tstart_{m_2} \qquad \forall m_1, m_2 \in M_f,\ order_{m_1} > order_{m_2} \quad (23)$$

$$tstart_{m_1} \leqslant tstart_{m_2} \qquad \forall m_1, m_2 \in M_f,\ order_{m_1} < order_{m_2} \quad (24)$$

$$(transfer_{src,dst} = 1) \Rightarrow \Rightarrow (start_{src,dst} = finish_{src,dst} - ttime_{src,dst}) \qquad \forall src \in S_f,\ dst \in M_f,\ src \neq dst \quad (25)$$

$$(transfer_{src,dst} = 1) \Rightarrow (stored_{src} + cached_{src} \geqslant 1) \qquad \forall src \in S_f,\ dst \in M_f,\ src \neq dst \quad (26)$$

$$(transfer_{src,dst} = 0) \Rightarrow (start_{src,dst} + finish_{src,dst} \leqslant 0) \qquad \forall s \in S_f,\ dst \in M_f,\ src \neq dst \quad (27)$$

$$(transfer_{src,dst_1} = 1 \wedge transfer_{src,dest_2} = 1) \Rightarrow \Rightarrow \Big((order_{dst_1} \leqslant order_{dst_2}) \Rightarrow (tstart_{dst_2} \geqslant available_{dst_1})\Big) \qquad \forall src \in S_f,\ dst_1, dst_2 \in M_f,\ src \neq dst_1 \neq dst_2 \quad (28)$$

Figure 28: Constraints used in the feasibility formulation of the transfer scheduling problem. A single file initially stored in a set of servers is requested by destinations D_f.

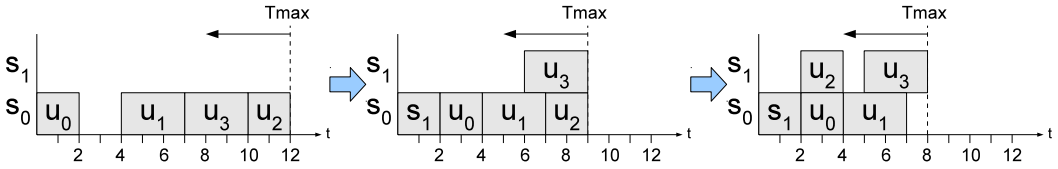


Figure 29: Effect of reducing the α value in the calculation of $Tmax$.

before the file is available in the source server. The constraints (23) and (24) enforce that two transfers for the same file are ordered. Constraint (25) ensures that transfer times between machines are taken into account. Constraint (26) enforces that files are transferred only when they are available at the source. Constraint (27) ensures that the start and finish variables are consistent with the transfer variable. Constraint (28) ensures that whenever two transfers share a common source, the order is maintained.

Using this formulation, the number of variables and constraints for each file is reduced to $O(|S||M|^2)$. This is a significant improvement over the baseline model, as time T is no longer a factor of the complexity. This permits to have a transfer scheduling complexity, which is independent of the schedule length.

5.2.2 *Approximation heuristic*

The quality of the solution produced by the feasibility model affects the fragmentation of the schedule. Optimum schedules have the advantage of reducing server idle times to the minimum, but they require a significantly higher solving time. In order to help finding a trade-off between solutions provided by feasibility model and optimal solutions, we propose an heuristic that permits to regulate how much margin the intermediate schedules may show.

Our approach is based on relaxing the admissible value for $Tmax$, representing the maximum makespan available for finishing all transfers (see Table 4). Determining the $Tmax$ value for each task depends on the number of servers and destinations, and their interconnection characteristics. In the worst case scenario, if we assume that all the interconnections have the same bandwidth, and all servers have the file in their caches, a lower bound for this parameter is $\lceil |D|/|S| \rceil \cdot maxtt$. In our case, we define our heuristic to compute the $Tmax$ value for each task as:

$$Tmax = \alpha \cdot \left\lceil \frac{|D_f|}{|S_f|} \right\rceil \cdot maxtt \tag{29}$$

where α is a weighting factor, $|D_f|$ is the cardinality of the set of destination machines, and $|S_f|$ is the cardinality of the set of servers. This model represents an α-approximation of the scheduling problem.

To graphically illustrate the effect of reducing the α value, Figure 29 shows a example scenario with two servers and four users interested in the file. Users u_0 and u_2 require two time units to retrieve the file while users u_1 and u_3 require three time units. Initially the file is only cached in server s_0 and transferring the file to s_1 will require two time units. For the purpose of discussion consider an initial value of $Tmax = 6$. In this configuration, the feasibility model could be solved involving only

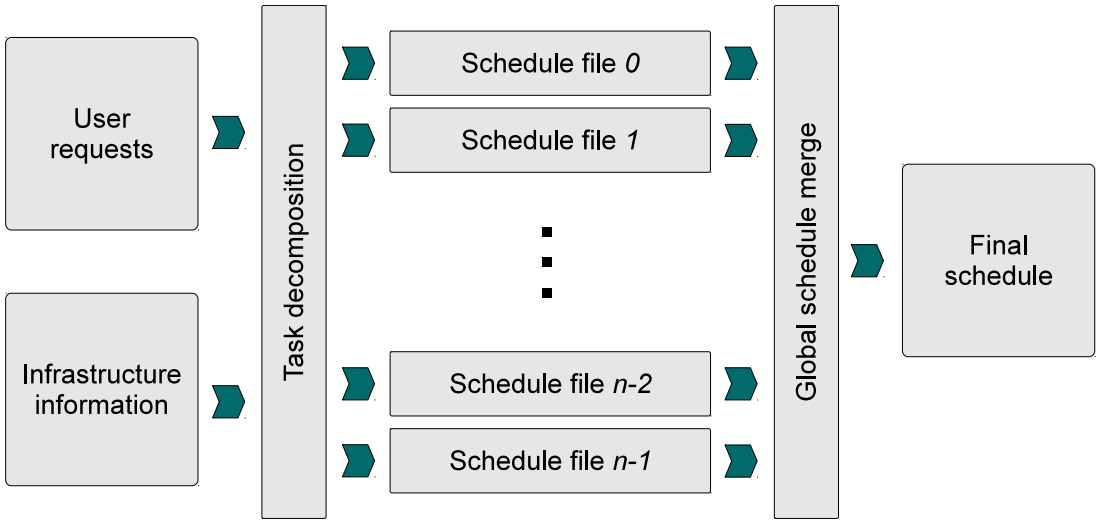


Figure 30: Distributed solving of the file transfer scheduling problem.

s_0 in the transfer. As Tmax is reduced to 9, s_0 is no longer able to transfer the file to all users in the maximum time and therefore s_1 must participate in the transfer process. After solving the model, we observe that the first operation is to transfer the file from s_0 to s_1. After that, the distribution of the file to user u_3 can overlap in time with the transfers to users u_1 and u_2 as two different servers are used. If we continue reducing the value of Tmax to 8, the schedule makespan is tightened. Eventually, the smallest value of Tmax that will produce a schedule (i.e., the problem has at least one feasible solution) will match the optimal schedule produced by the classical minimization approach.

5.2.3 *Distributed transfer scheduling*

This section describes the different components of the architecture that permits to distribute the solving process. Next section shows how partial schedules can be merged into the final schedule. Figure 30 presents a general overview of the involved components. The user requests and the current infrastructure information are the inputs for the task decomposition module. The user requests simply represent a list of which files are requested by which users. The infrastructure information contains the current status of the system: a list of active servers, the distribution of files over the servers, and the interconnection bandwidths. Using this information, the task decomposition module divides the problem into a set of subtasks. Each subtask receives as input the information required to solve a single file transfer problem:

$$< S_f, U_f, \mathtt{stored}[\,], \mathtt{ttime}[\,][\,], \mathrm{Tmax} >$$

where S_f and U_f are the subset of servers and users involved in the transfer, $\mathtt{stored}[\,]$ is the distribution of the file over the servers (i.e. an array of booleans of size S_f indicating which server is storing the file initially) , $\mathtt{ttime}[\,][\,]$ represents the time required to transfer the file between src and dst. These input parameters are fed to a linear programming solver, which generates and solves the model associated with f.

To reduce the complexity of the problem and speed up the solving process, the task decomposition module can limit the number of users per problem, such that large problems are also split into smaller subsets (e.g. a problem with 100 users, can be split into two subproblems of 50 users each).

The generated subtasks are independent among themselves, making it possible to solve several subtasks in parallel. Task assignment can be done based on different policies such as round-robin, dynamic assignment, etc. The result produced by each task is a partial schedule of transferring the file f to destinations D_f. A single file transfer is represented by the tuple $< src, dst, f, start, finish, dependency >$, where the $dependency$ value indicates that src does not store any copy of f, and this has to be retrieved prior transferring the file to dst. To obtain the final schedule, the global schedule merge module combines all partial solutions, as discussed in the next subsection.

5.2.4 *Merging partial schedules*

The problem of merging the partial schedules is similar to a constrained bin packing problem [119]. The bin packing problem consists of optimally assigning a number of items with given costs or weights into a set of bins with a maximum capacity. In our case, we have a set of server timelines with idle periods, where we try to assign the scheduled transfers. We model this problem by defining bins as time intervals, where the capacity is the idle time. Algorithm 2 is the pseudocode of the merge function. Transfers from server to server (in case a server is requested a file that does not store locally) are scheduled before transfers from servers to users in order to mitigate the dependency problem.

For each transfer, the algorithm calls the function $addTransfer$ (Algorithm 3), which searches for an interval, in which a transfer can fit. The algorithm iterates over the list of available intervals until it finds fitting intervals for both source and destination.

Algorithm 2 Global schedule merge algorithm.

Input: serverTransfers, userTransfers
  $globalSchedule = \emptyset$
  //Schedule server-to-server transfers
  for each $t \in serverTransfers$ **do**
    addTransfer(t, globalSchedule)
  end for
  //Schedule servers-to-users transfers
  for each $t \in usersTransfers$ **do**
    addTransfer(t, globalSchedule)
  end for
  return globalSchedule

Figure 31 shows the merge of two partial schedules, in a scenario in which 3 users request 2 files from 3 servers. Interval tables are shown on the right of the computed schedule. Initially, all machines in the system have the interval $[0, \infty)$ available. After three iterations, the partial schedule is shown. Notice how the transfer of file 1 to

Algorithm 3 addTransfer algorithm

Input: gs: Current global schedule
// Transfer to add to the global schedule
Input: t: Transfer $<$ src, dst, f, start, finish, dependency $>$
srcInterval = getNextAvailableInterval(t.src)
dstInterval = getNextAvailableInterval(t.dst)
assigned = False
while not assigned **do**
  if fit(t, srcInterval) $\wedge$ fit(t, dstInterval) **then**
    assign(t, srcInterval, dstInterval, gs)
    updateInterval(srcInterval, t)
    updateInterval(dstInterval, t)
    assigned = True
  else
    srcInterval = getNextAvailableInterval(t.src)
    dstInterval = getNextAvailableInterval(t.dst)
  end if
end while

user u_1 has been shifted in time, as there was a collision with the transfer of file o from server s_1. This causes the original interval of server s_2 to be split into $[0, 2]$ and $[3, \infty)$. The final schedule is obtained after all transfers have been assigned.

5.3 IMPROVEMENTS ON THE FORMULATION

The model presented in the previous section considers a scheduling problem whose objective is to find a solution within an established makespan. In this section, we explore different modifications of the model in order to support the multiplexing of server bandwidth and the fault-tolerance capabilities of the resulting schedule.

5.3.1 *Multiplexing server bandwidth*

Multiplexing the server bandwidth permits to schedule parallel transfers from a server based on a multiplexing degree m. To introduce server bandwidth multiplexing in the previous model, we transform the previous definition of the scheduling problem. To facilitate the transformation, we refer with $S(S_f, D_f)$ to the scheduling problem presented in Section 5.2.1, where S_f is the set of servers and $D_f = \{d_0, ..., d_n\}$ is the set of destinations.

To assign the requesting destinations to the new multiplexed channels, we distribute the destinations uniformly among them. Consequently, our original problem S becomes:

$$S = S_0(S_f, D_{f,0}) \bigcup ... \bigcup S_m(S_f, D_{f,m}) \tag{30}$$

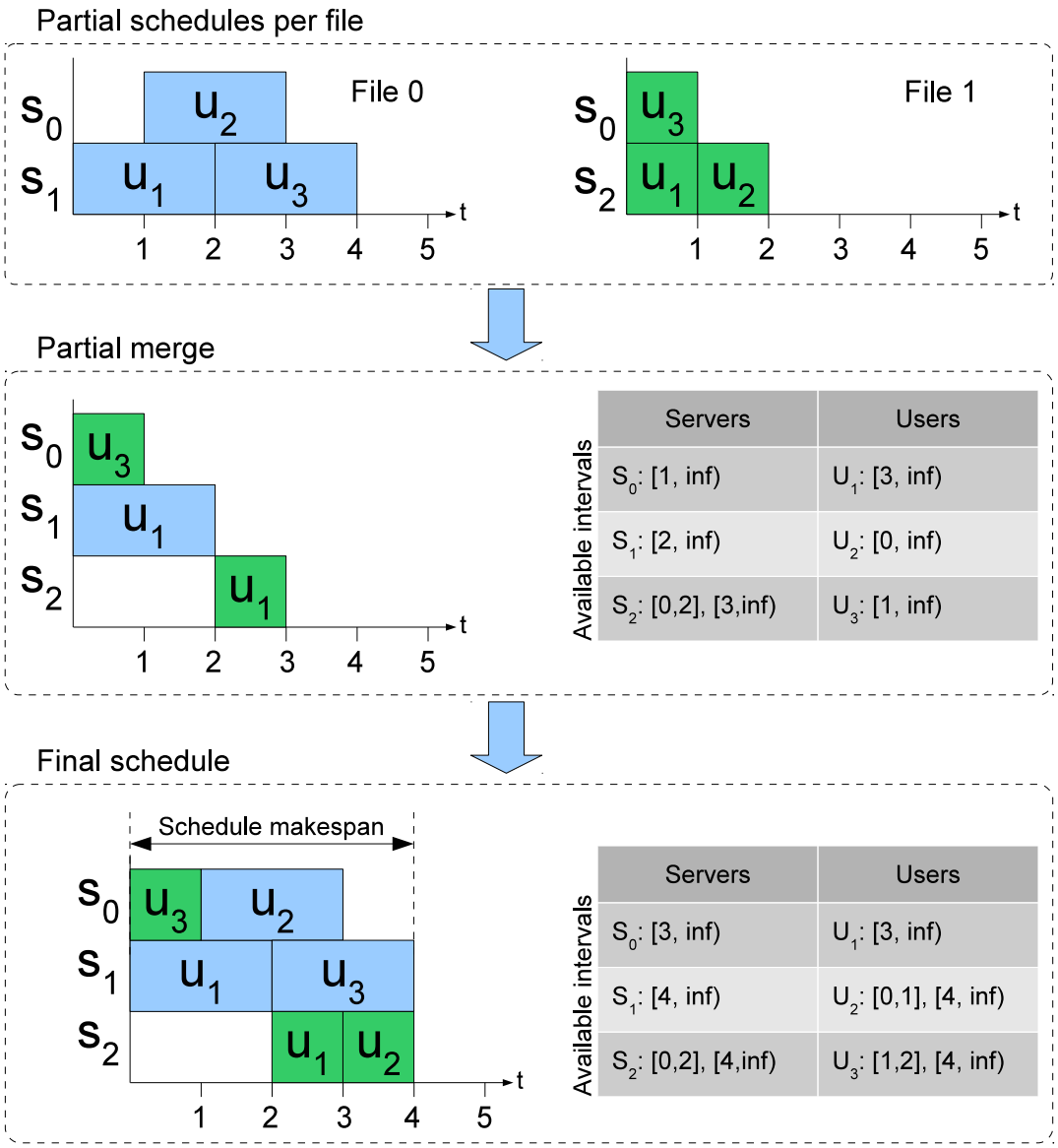


Figure 31: Merge of two file schedules to obtain a global schedule. Transfers are represented by boxes whose color represents different files. The length of the boxes indicates the duration of the transfer. The source of the transfers appears in the Y axis, and the destination is indicated in each box. The intervals are updated during the merge as shown in the right tables.

where the initial scheduling problem is now decomposed into a set of reduced scheduling problems. Each problem shares the same set of servers, with a redefined set of users:

$$U_{f,p} = \bigcup_{0}^{n} u_j : channel(u_j) = \left\lceil \frac{j \cdot m}{|D_f|} \right\rceil = p \tag{31}$$

In this manner, each sub-problem is assigned a portion of the users found in the original problem. Additionally, it is necessary to multiplex the value of the existing bandwidth between the servers and the users. For this purpose, and considering a uniform distribution of the bandwidth on the channels, the time to transfer a file between two sites is redefined as:

$$ttime_{src,dst} = \frac{ttime_{src,dst}}{m} \tag{32}$$

The presented bandwidth multiplexing strategy does not require any changes in the presented approximation model and can be employed to improve the bandwidth utilization of the existing servers. Additionally, it is also possible to use the initial part (Equation 30) of this approach to regulate the maximum number of users per problem to reduce the overall complexity of the problem and speedup the solving process.

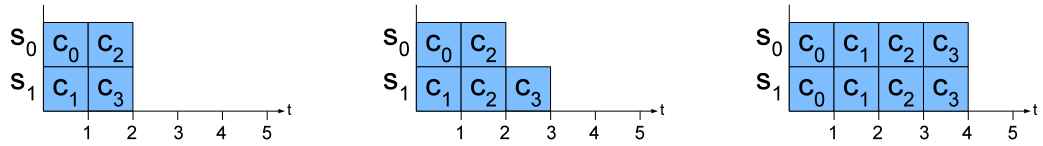
5.3.2 *Adding user-driven fault-tolerance capabilities with parallel downloads*

The previous model considers a scenario where each user destination transfers its files connecting to a single server. In this section, we are interested in adding fault-tolerance capabilities to our model in order to support the transfer of files using several servers in parallel.

We distinguish three types of download parallelism that could be added to the model. To explain the differences between them, Figure 32 depicts the resulting schedule of applying each type of parallel download considering a file f composed of 4 chunks. For clarity purposes, the file is distributed to a single user and both servers have the same available bandwidth.

First, in Figure 32a we observe the schedule produced when two servers are used in parallel to obtain different chunks of the file. This approach is throughput-oriented which translates into a significant speedup of the transference process. However, the resulting schedule is not fault-tolerant making necessary to recalculate it in case of a server failure as the file can not be transferred in the allocated time and/or some chunks may not be available. To address this situation, a possible approach (Figure 32b) is to introduce some level of overlapping. In this way, a part of the file is scheduled to be transferred from several available servers. However, this approach is also susceptible of mirror failures. To solve this problem, the next approach (Figure 32c) is to schedule the complete transfer of a file from several servers at the same time. In case of failure of one server, the makespan of the transfer is not affected.

It is important to notice, that in this section we are interested in providing a scheduled fault-tolerance, nor reducing the makespan by means of using non-overlapped



(a) Parallel download of different chunks.

(b) Parallel download with partial overlapping.

(c) Parallel download with full overlapping.

Figure 32: Different types of parallel downloads considering a file f with 4 chunks c_0, c_1, c_2, and c_3. From left to right, parallel download using two servers for different chunks, parallel download with partial overlapping, and full overlapped parallel download.

SERVER AVAILABILITY	NUMBER OF SERVERS INVOLVED (η_{src})
$availability \geqslant 90\%$	1
$60\% \leqslant availability < 90\%$	2
$40\% \leqslant availability < 60\%$	3
$availability < 40\%$	0

Table 5: Example configuration to determine the number of servers (η_{src}) involved in a transfer using the server availability.

parallel transfers. In this respect, we opt for the last configuration (Figure 32c), and modify the model to support parallel transfers.

Before adding or modifying any equations, it is important to clarify the mechanism used to determine the level of parallelism required for a transfer. An initial approach would be to define a parameter η that represents the number of servers required per transfer. Considering the fact that some servers may be more reliable than others, we opt for a fine grain configuration permitting to specify the number of required servers based on which server participates in the download. Using this approach, we are able to permit different levels of parallelism based on the involved servers. In order to do that, we introduce a new parameter η_{src} that controls the number of required servers in a transfer when server src participates in it. In practice, this parameter is configured by the system administrator based on the specific requirements of each application scenario. As an example, Table 5 shows a possible configuration of the parameters. In this case, a server with an availability of 85% requires an additional server in order to participate in a transfer. In case the availability drops below 40%, the server is no longer used as the number of total servers involved is zero.

By configuring the values of η_{src} using the aforementioned table the infrastructure administrator defines the type of fault-tolerance to be applied: optimistic, high-availability, and probabilistic.

- **Optimistic**: If $\eta_{src} = 1, \forall src \in S_f$, we employ an optimistic approach as the schedule must be recalculated in case of a mirror failure during a transfer.

- **High-availability**: If $\eta_{src} > 1, \forall src \in S_f$, we assure that all transfers from a server have at least one backup server in case the transfer fails.
- **Probabilistic**: If $\eta_{src} \geqslant 1, \forall src \in S_f$, we employ a similar approach to the high-availability but considering that some server may be more prone to failures than others. This approach is more flexible as it permits to adapt the level of fault-tolerance depending on the characteristics of the servers.

Using the model presented in Section 5.2.1 as base, we introduce several modifications. The first modification with respect to the feasibility model presented in Section 5.2.1 is to transform the expression in Equation 17 into a new constraint (Equation 33). This substitution is motivated by the fact that all parallel transfers must start at the same point.

$$(transfer_{src,dst} = 1) \Rightarrow (tstart_{dst} = start_{src,dst}) \quad \forall src \in S_f, dst \in D_f, src \neq dst \tag{33}$$

Second, we modify the expression shown in Equation 18 with a new expression (Equation 34) to capture the new semantics of problem. As several transfers may be active at the same time, we take the slowest time to mark the time the download is available at the user. Using this approach, in case one of the mirrors fails during the transfer, there is enough allocated time in the schedule to retrieve the complete file.

$$available_{dst} = \max_{src \in M_f} finish_{src,dst} \quad \forall dst \in M_f \tag{34}$$

Third, we replace Equation 20 by Equation 35 in order to assure that the number of transfers received by a user is bounded between one and the maximum value of η_{src}.

$$1 \leqslant \sum_{src \in S_f} transfer_{src,dst} \leqslant \max_{src \in S_f} \eta_{src} \quad \forall dst \in D_f, src \neq dst \tag{35}$$

Fourth, we introduce Equation 36 to force parallel transfers from different servers to be active at the same time for a common target destination. Similarly, we add Equation 37 to assure that transfers from one server to several users do not start at the same time.

$$(transfer_{src_1,dst} + transfer_{src_2,dst} >= 2) \Rightarrow \Rightarrow (start_{src_1,dst} = start_{src_2,dst}) \quad \forall src_1, src_2 \in S_f, dst \in M_f,\ src_1 \neq src_2 \tag{36}$$

$$(transfer_{src,dst_1} + transfer_{src,dst_2} >= 2) \Rightarrow \Rightarrow (start_{src,dst_1} \neq start_{src,dst_2}) \quad \forall src \in S_f, dst_1, dst_2 \in M_f,\ dst_1 \neq dst_2 \tag{37}$$

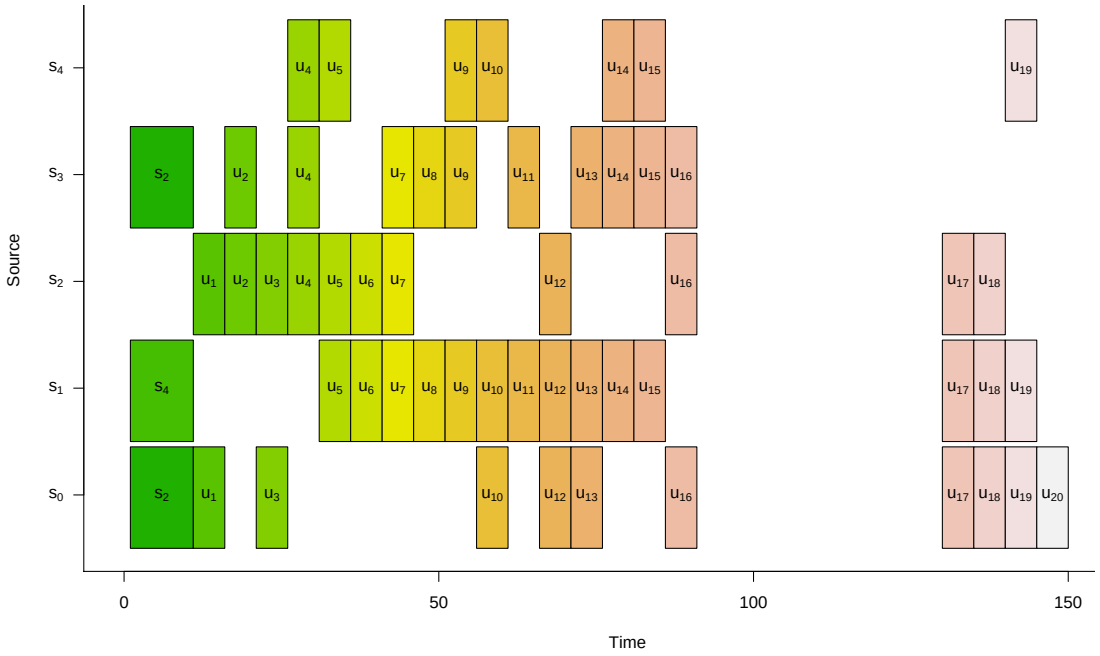


Figure 33: Example schedule produced by the model with parallel downloads considering a scenario with 5 servers and 20 users with a parallelism degree $\eta = [1, 1, 2, 2, 3]$.

Finally, we add Equation 38 to enforce that at least η_{src} are involved in a transfer whenever server src is involved.

$$(transfer_{src_1,dst} = 1) \Rightarrow \eta_{src_1} \leqslant \sum_{src_2 \in S_f} transfer_{src_2,dst} \leqslant \max_{src \in S_f} \eta_{src}$$
$$\forall src_1 \in S_f, dst \in M_f \qquad (38)$$

Using this modifications it is possible to introduce parallel downloads in the model. To illustrate its behavior, Figure 33 shows the application of the modified model to a scenario with 5 servers and 20 users with a parallelism degree $\eta = [1, 1, 2, 2, 3]$.

At first, we notice a transfer from server s_1 to to server s_4 as that server did not have the file. At the same time, servers s_0 and s_3 transfer the file to server s_2. Notice that two parallel transfers are scheduled as $\eta_3 = 2$. From that point, we observe different parallel transfers depending on the parallelism level required by the servers participating in the transfer. Notice that as a large value $Tmax$ is being used, some time slots are not occupied (interval [92, 129]). In addition, it is important to highlight that depending on the value of $Tmax$, the number of parallel transfers in the solution will change. As $Tmax$ decreases, we expect less parallel transfers to be scheduled due to the fact that more servers need to be involved in a transfer and the total time may increase as a consequence.

5.4 EVALUATION

In this section we present the evaluation of the proposed solution. First, we discuss the implementation of our solution based on a linear programming solver and Hadoop. Second, we study the quality of the solution provided by our feasibility

model using a single file schedule. Third, we compare the results produced using our distributed approach with two greedy transfer assignation policies, for scenarios requiring between 28K and 228K transfers.

5.4.1 *Implementation*

The implementation of our solution is based on two main components: a linear programming solver and a distributed computing framework. While various technologies can be employed, we have settled on IBM ILOG Suite [1] for linear programming and Hadoop framework for distributed computing.

The selection of the ILOG Suite is motivated by the existence of a programming API and the fact that is one of the best performing [75, 64] state-of-the-art solvers available at the moment. Similarly, the selection of the Hadoop framework has been motivated by it extensive documentation and the fact that is a widely extended solution for commercial and non-commercial environments.

Our model, presented in Sections 5.2.1 and 5.2.2, has been fully coded in CPLEX. The distributed computations described in Sections 5.2.3 and 5.2.4 have been implemented as a Map/Reduce program in the aforementioned Hadoop framework. Hadoop [2] is a map-reduce framework based on the Google MapReduce model [32]. Hadoop jobs consist of two main task types: *map* and *reduce*. A map task (Equation 39) takes one key/value pair (k_1, v_1) and applies a function producing a set of key/values (k_2, v_2).

$$\mathtt{map}(k_1, v_1) \rightarrow \{(k_2, v_2)\} \tag{39}$$

The reduce tasks (Equation 40) take the output of the map tasks and apply a given operation to produce a combined result.

$$\mathtt{reduce}(k_2, \{v_2\}) \rightarrow (k_3, v_3) \tag{40}$$

In our Hadoop solution, `map` tasks solve single file schedules using the IBM ILOG Suite, while `reducer` tasks are mergers of partial schedules produced by map tasks, as described in Section 5.2.4. The result of the Map/Reduce program is the global data transfer schedule.

The reduction process can be iteratively executed in order to further exploit the parallelism in the system. However, this approach would require the use of a map/reduce framework that supports iterative reduce phases [115, 38] without the execution of the map phase. This implementation improvement remains out of the scope of the thesis, as we are interested in testing the feasibility of the distributed solving process.

5.4.2 *Computational complexity in practice*

In this section, we study how different parameters of the model affect the computational complexity. To perform this study, we evaluate the time required to produce

1 IBM Ilog Suite: www.ibm.com/software/websphere/ilog/
2 Apache Hadoop hadoop.apache.org

a solution and the associated schedule makespan. To obtain the data, we solve two configurations: one with a fixed number of 15 servers with a varying number of users between 1 and 15; and other with a fixed number of 15 users and a varying number of users between 1 and 15. In both cases, the $Tmax$ value ranges between 1 and 1,000. As some values of $Tmax$ may not produce a solution (the minimum time required is greater than $Tmax$), we establish a timeout of 5 minutes for each configuration. If a solution is not found in that time, we consider that the problem no longer has a solution and therefore smaller values of $Tmax$ are not further explored. The objective of this test is to understand how the complexity is affected by the different parameters of the model.

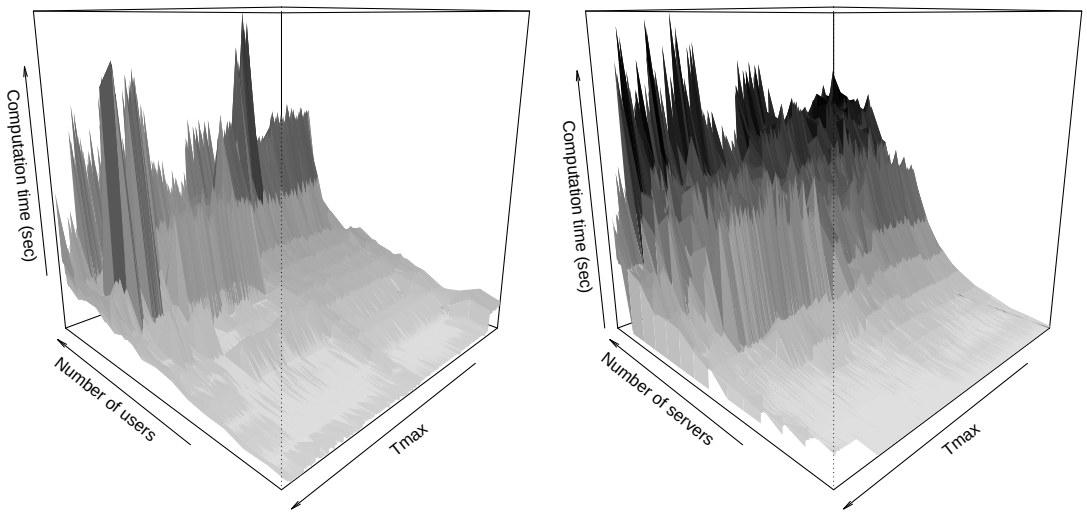
Figures 34a and 34b show the computational time required to obtain a solution using the aforementioned configurations in scenarios with a fixed number of servers and users respectively. Initially, without observing the figures, we expect that the computational time required to solve a scenario increases with the number of users and servers. However, our results demonstrate that the computational time is not only affected by those parameters. In both figures we observe that the computational the existence of "valleys" where the computational time is significantly lower than in its proximity.

Based on our experience, we explain the existence of those valleys due to the operation of the underlying solver. As our solver uses a modified version of the Branch-and-cut heuristic, a tree-like structure is deployed in memory to track the different configuration of the variables that produce a viable solution. The way this tree is defined is based mainly on the structure of the instance of the model and some startup parameters determined by the heuristics of the solver. Therefore, a small change in the parameters of the model may produce a structure that is slightly easier for the solver.

The implications of this behavior are quite important to optimize the execution of this (and possible others) type of models. By understanding which configurations represent a reduced complexity for the solver we could determine which is the optimal number of users and servers per problem, which heuristics produce a reduced computational time, etc.

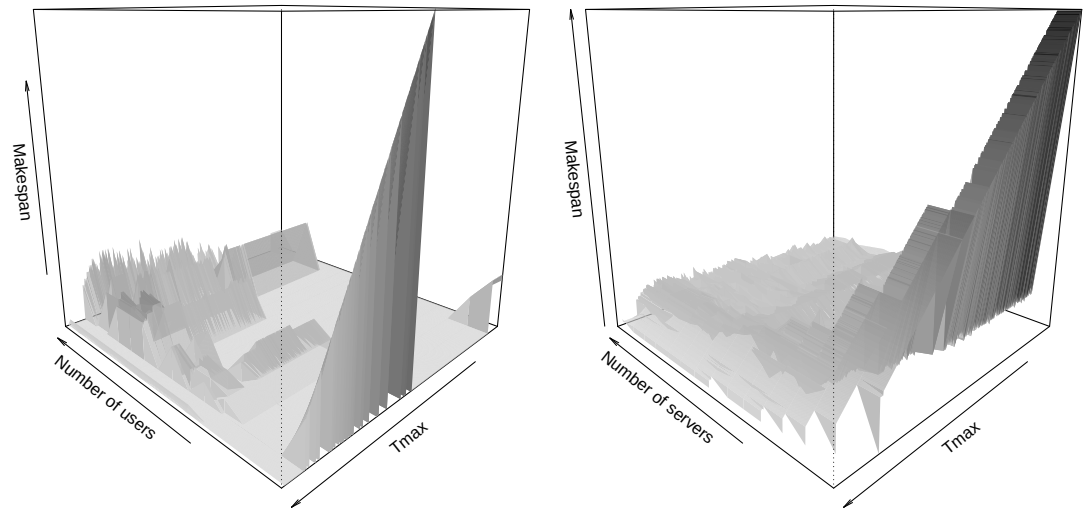
Figures 34a and 34b depict the resulting schedule makespan for the same cases. Similarly to the previous results we also observe the existence of configurations that produce a smaller makespan. This behavior has two main implications. First, we observe that some values of $Tmax$ produce a reduced makespan. This demonstrates the viability of using a feasibility formulation as tight schedules can be produced without drastically limiting the search space with the $Tmax$ value. Second, by combining the results of the produced makespan with the computational time, we could optimize our heuristic to obtain solutions with a reduced makespan and computational time.

To summarize, our evaluation demonstrates that the computational complexity highly depends on the specific problem configuration. The intuitive behavior would suggest that the complexity will increase with the number of users and servers, and decrease with the maximum makespan. However, our results demonstrate that the computational complexity in practice does not follow the expected behavior. We find that there are some sweet spots around some configurations leading to situations where it is easier to solve a problem for a large amount of servers and/or users, than



(a) Fixed servers. (b) Fixed users.

Figure 34: Impact of varying $Tmax$ in the computational time required to solve scenarios with a fixed number of servers and users respectively.



(a) Fixed servers. (b) Fixed users.

Figure 35: Impact of varying $Tmax$ in the schedule makespan in scenarios with a fixed number of servers and users respectively.

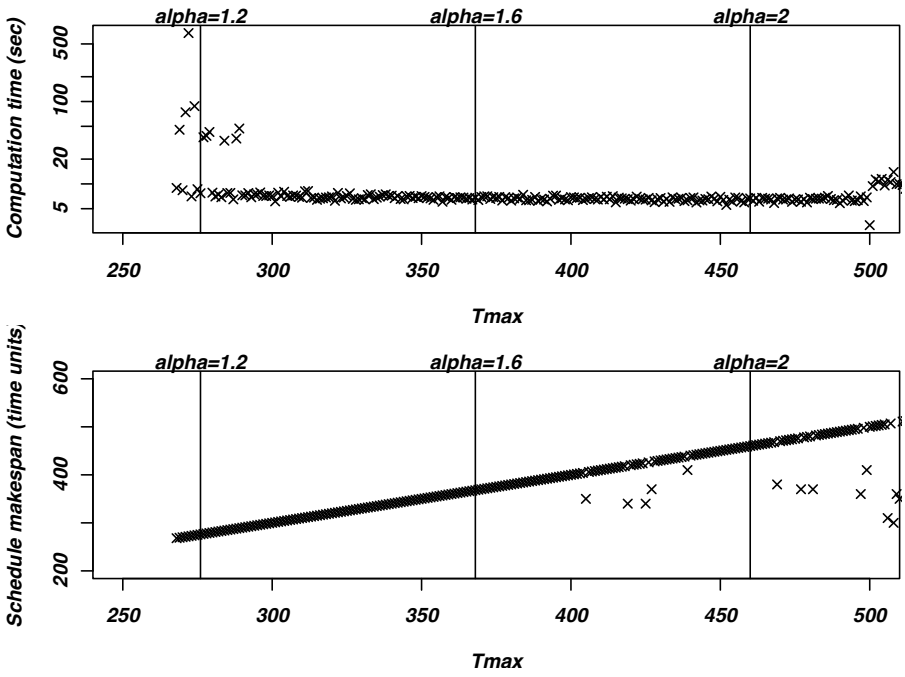


Figure 36: Effect of changing α in terms of computation time and schedule makespan.

solve it for a reduced set. We believe that this behavior is highly related with the way the solver covers the solution space, and how the branches of the internal tree structure that supports the solving process are explored and cut.

5.4.3 *Sensitivity analysis of scheduling solutions*

In the previous section, we have explored how the computational complexity evolves depending on the parameters of the problem. In this section, we evaluate our heuristic and study the effect of varying the α value that is used to compute the effective $Tmax$. In this respect, we are interested in determining whether the heuristic is able to find effective configurations of the problem, so that the computational time is reduced.

We run experiments to empirically determine which is the impact of varying the value of α in practice. We measure the computation time and the total scheduled makespan for different values of $Tmax$ ranging between 550 time units to the lowest value where a solution can be found. For clarity purposes, we increase the timeout until the problem is considered to be unsolvable in that configuration to 15 minutes. The lowest $Tmax$ value that produces a solution in the feasibility model matches the optimum schedule makespan for the baseline model (for $\alpha = 1$). For experimental purposes, we consider a scenario with two servers and 45 users per problem. However, as shown before, these effects are observable in all configurations. The experiments have been executed on an Intel Xeon E4505 machine with 4GiB of RAM.

Figure 36 shows the computation time required by the solver (top), and the schedule makespan obtained for each value of $Tmax$ (bottom). As $Tmax$ decreases, the scheduled length decreases with an increment of the computation time. Using this heuristic to compute $Tmax$ allows to find a feasible schedule in a fraction of the time required to find the optimum.

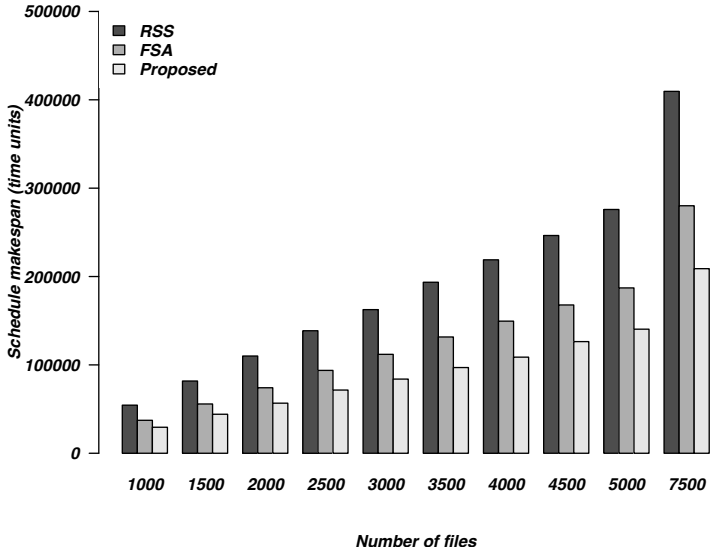


Figure 37: Schedule makespan in time units for the different configurations.

In order to see this effect, we explore the possible outcome when setting α to 1.2, 1.6, and 2. For $\alpha = 1.2$ we get a schedule makespan of 277 time units in 36.80 seconds. If we relax α to 1.6 the makespan is 369 time units calculated in 6.37 seconds. Finally, $\alpha = 2$ produces a makespan of 461 time units in 6.25 seconds.

The solution in the second case with $\alpha = 1.6$ is 1.33 times larger than the one with $\alpha = 1.2$, but it requires 83% less computation time. Depending on the application scenario, different heuristics can be explored. In a scenario with a small amount of files, small values of $Tmax$ reduce server idle times. Large values of $Tmax$ increase server idle time, being suitable for scenarios with large amounts of files. These idle times are partially removed during the merging phase (section 5.2.4).

5.4.4 *Evaluation of distributed transfer scheduling*

In this section we evaluate the quality of the scheduled produced by our distributed methodology and we compare two frequently used approaches in server-based request distribution [93]: RSS (Random Server Selection) and FAS (First Available Server). In RSS the schedules are produced by randomly assigning requests to servers. In FAS the schedule is built by assigning the first available server to the current request to be served.

The evaluation setup is the following. A variable number of files are stored randomly over 8 servers. Each file is requested by 28 users (destinations) randomly selected from a set of 60. We report results for different number of files from 1,000 to 7,500 (i.e. the total number of requests vary between 28K to 228K). The α value is set to 2 (i.e. the schedule makespan for each file is at most twice the optimum value).

First, we report the schedule makespan produced by the different approaches. Figure 37 shows the makespan for the different transfer scheduling policies. In all cases the schedule produced by our solution is considerable shorter than RSS and FAS. For 7500 files FAS and RSS produce schedules which are 34% and 96% larger than our solution, respectively. The standard deviation is less than 3% for all approaches. Additionally, our approach scales well with the number of the files.

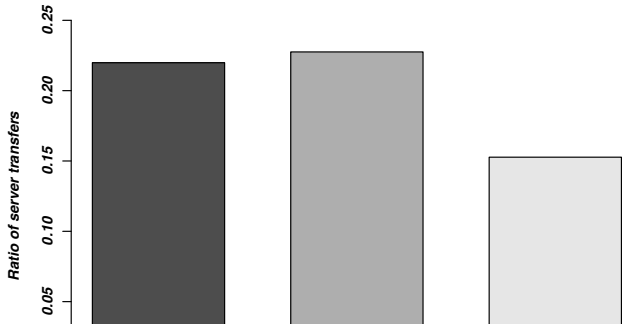


Figure 38: Aggregate server miss rate.

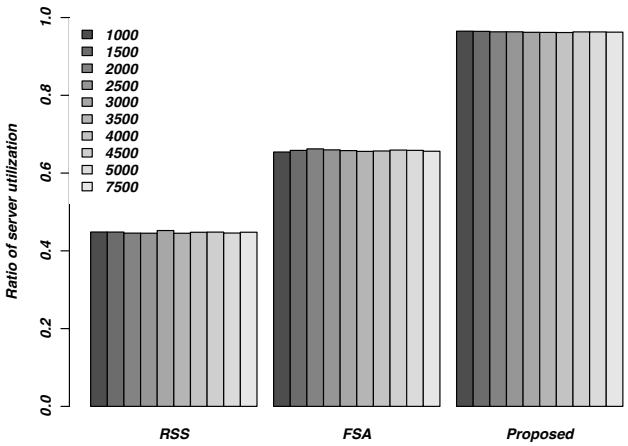


Figure 39: Ratio of server utilization.

Second, we evaluate the aggregate server miss rate caused by our policy, RSS and FAS. This value is calculated by dividing the number of requests that cause transfer between servers over the total number of requests. This measure is a good indicator of locality. Figure 38 shows the aggregate server miss rate for our proposed solution, RSS and FAS. The miss rates are 22% for RSS, 23% for FAS, and 15% for our solution. Thus, our solution improves by 7% and 8% on the number of transfers between servers, which results in a lower contention inside the server infrastructure.

Third, we evaluate the server utilization caused by RSS, FAS and our solution. We calculate the average server utilization as:

$$server_{utilization} = \frac{1}{|S|} \sum_{srv \in S} \frac{makespan_{srv} - idle_{srv}}{makespan_{srv}} \tag{41}$$

where $makespan_{srv}$ is the total schedule length and $idle_{srv}$ is the inactive time of server srv (i.e. server srv has no scheduled request). Intuitively, this metric evaluates the amount of fragmentation of the produced schedules. A high amount aggregated idle times causes low resource utilization and vice versa. A high utilization (close to 100%) indicates that the proposed solution is closed to optimum.

Figure 39 shows average server utilization for three policies (RSS, FSA and our solution), and for a number of files varying between 1000 and 7500. We note that, for each policy, the server utilization does not vary considerable with the number of files. Our solution achieves around 95% server utilization in all cases, which indicates that

the produced schedules are close to optimum. This value represents a considerable improvement over FSA and RSS, which show a utilization rate of 30% and 51%, respectively.

5.4.5 *Energy saving considerations*

The utilization of a transfer schedule has several ramifications. In particular, there is a direct relationship between the quality of a schedule and the energy consumption of applying that schedule in a real scenario.

In this respect, we can assume that idle server times are periods of time where the server may be powered down in order to save energy. However, we need to consider two factors. First, continuos fast cycles of start and stop may impact the reliability of the infrastructure. Second, shutting down and booting up machines involves an overhead (ρ) due to the time cost of gracefully stoping all services, and requesting an on-demand resource and configuring it when a new machine is required.

Therefore, an optimal schedule from the point of view of the energy consumption should concentrate the utilization on the servers reducing idle periods. Notice, that obtaining a minimal makespan schedule does not necessarily imply obtaining a schedule with reduced idle periods in all involved servers.

In our case, we define the energy that could be saved by a good scheduling method as:

$$E = \sum_{srv in S} (idle_{srv} - \rho_{srv}) \cdot energy_{srv} \tag{42}$$

where $idle_{srv}$ is the amount of time server srv is idle, ρ_{srv} is the required delay to stop and start a machine with the characteristics of srv, and $energy_{srv}$ is the energy consumed per unit of time by server srv.

Using the results presented in the previous section, our solution is the most energy efficient requiring less than 5% of overhead energy, while RSS and FSA require 70% and 49% extra energy. These results demonstrate the importance of employing efficient dynamic infrastructures. This type of infrastructure should be able to react to the ongoing demand, but at the same time maintain the level of overprovisioning at a minimum level. Of especial interest are cloud-based infrastructure where the time required to start new machines may take up to several minutes and booted machines are pay by the hour usage. In this respect, other works [109, 110] outside the focus of this thesis have been carried out to propose novel techniques for adaptive control of elastic server infrastructures.

5.4.6 *Performance impact of the underlying hardware*

In this section, we are interested in analyzing the impact of changing the underlying hardware architecture the solver is executed on. Our objective is to determine what could be the possible performance gains of executing the solving process in a different hardware architecture.

Our analysis consists of executing a scenario with 5 servers, 20 users and a $Tmax$ value between 1,000 and the minimum value that produces a solution. We execute

CODE	NAME	CLOCK SPEED	CACHE SIZE	CORES	THREADS
A	Intel©Xeon X7350	2.93 GHz	L2 8 MiB	4	4
B	Intel©Xeon E7-4807	1.87 GHz	L3 18 MiB	6	12
C	Intel©Xeon E5-2620	2.00 GHz	L3 15 MiB	6	12

Table 6: Hardware characteristics

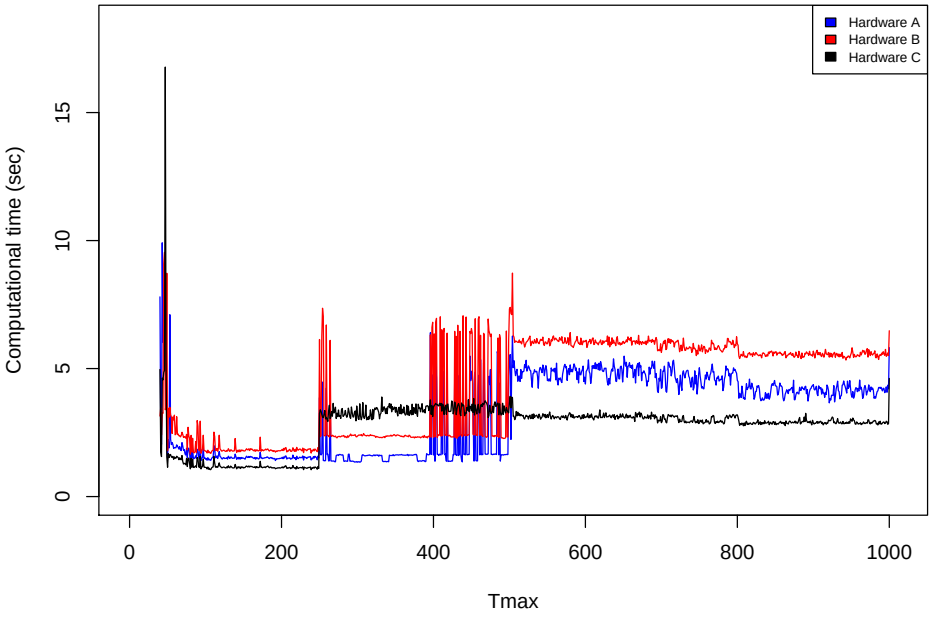


Figure 40: Computational time depending on the underlying hardware.

this scenario in three different hardware configurations (A, B, and C) whose characteristics are summarized in Table 6.

Figure 40 shows the computational time for the test scenario. The first conclusion that can be drawn from this figure is the existence of different complexity levels on the scenario. This situation matches the observations previously noticed in Figure 34 (Section 5.4.2). However, in this case, we notice that the number of levels and the length of the valleys depends on the underlying hardware. For example, a heuristic producing a $Tmax$ value of 800 will produce faster results in hardware C, whereas if the heuristic value is 400, hardware A would be the most appropriate for the task. Using this figure, we can also observe how larger values of $Tmax$ do not necessarily produce faster solutions.

Second, with regards to which hardware would be better to solve a particular problem, we observe that the computational time is influenced by many factors. First, some results ($Tmax$ value of 300) suggest that in some cases the clock speed may be a critical factor as hardware A outperforms hardware B. However, in other cases

($Tmax$ value of 200), a combination of L3 cache size and clock speed produce better results as hardware C outperforms A and B. In general, our results suggest that the underlying hardware produces a significant impact on the time required to solve a particular configuration of our model. Moreover, the heuristic selected for determining the $Tmax$ value should be tailored to the underlying hardware as a general purpose heuristic does not produce proportional results when presented with different hardware specifications. Finally, the number and size of the different levels of the cache hierarchy may have a critical impact on the time required to obtain a solution. As the focus of the section was to perform an initial analysis of the hardware impact, it remains for future work the realization of a detailed study of these observations.

5.5 SUMMARY

In this chapter we have presented the Transfer Scheduling module of the architecture. Using the virtual communities detected by the Community Detection module, the subscriptions and the system state, the module is able to produce an optimized transfer schedule. To achieve this objective, we presented a linear programming model that captures the requirements of a data distribution system. We modify the model so that it can be solved in parallel using the map/reduce approach and provide a detailed evaluation.

Linear programming models for these type of scenarios are known to be complex to solve. To address this issue, we introduce two major modifications. First, we transform the original minimization problem into a feasibility problem. In this way, the result of solving the model is the first solution found satisfying all requirements and not the minimum possible schedule. In order to regulate how far from the optimal solution is our schedule we introduce a heuristic. Our heuristic regulates the maximum time slots available to perform all transfers in the feasibility problem.

Our feasibility problem is focused on solving the scheduling problem for a single file. Therefore we define a method based on the map-reduce approach that permits to distribute single file problems to be solved in parallel and gather the final solution in the reducing steps. Additionally, we also defined several modifications of the problem that permit to optimize other aspects of a data distribution scenario. First, we permit to regulate the number of users per problem. If a problem involves a large number of users, it is possible to split the problem into several subproblems sharing the same servers, as the reduce step will produce a complete schedule. This permits to regulate the complexity of each subproblem. Second, we permit to multiplex server channels, so different users can be served in parallel from the same servers. Finally, we modify the model to permit parallel transfers of the same file from a set of servers with a specific parallelization degree. In this way, if a server fails in the middle of transfer, it is possible to continue the process from the remaining scheduled servers.

Our evaluation demonstrates that the computational complexity of such a model highly depends on the input parameters and does not scale linearly. This motivates the introduction of the heuristic that has been demonstrated to reduce the computational time required to obtain a schedule. Moreover, we also discover that the computational complexity also depends on the underlying hardware configuration. In

particular, we believe the cache hierarchy and size has a significant impact in the solver performance.

Regarding the quality of the produced schedule, we compare the makespan produced by our distributed solving of a linear programming problem with the solutions produced by two well-known policies: Random Server Selection and First Server Available. The results demonstrate the advantages of our proposal in several aspects: (1) the resulting schedule has shorter makespan when compared with the alternative methods; (2) our solution reduces intermediate server to server transfers and the associate miss rate in the selected servers; (3) our solution achieves a significant 95% server utilization of the resources, and (4) our solution is more energy efficient having an idle time of 5%.

6

FAST DATA TRANSFERS

In the recent years, the amount of information moved through the Internet has experienced a spectacular growth. Some aspects such as computing and storage capabilities had followed the increase; however, a fundamental key element is lagging: the networking infrastructure. While it is possible to maintain an up-to-date interconnection inside an organization, the connectivity with other external organizations involves multiple hops, providers and technologies (e.g., ethernet, fiber channels, etc.). In order to update these types of infrastructures, all intermediate links must be updated which requires a significant investment from different providers and organizations. Due to the practical complexities, in this thesis we address the objective of providing fast data transfers without requiring changes in the underlying interconnection infrastructure.

In this thesis, we focus our study on applications that require the transfer of large amounts of data and could benefit from our proposal. In this respect, scientific applications are a good candidate for this type of study. As a practical example, the LHC (Large Hadron Collider) project is expected to produce [1] around 15 petabytes of information per year. The VLT (Very Large Telescope) survey is expected to produce [2] around 100 Terabytes of data per year. In this type of applications, there is usually an important gap in terms of connectivity between the sites where data is actually produced and the institutions around the world that want to access it.

In this scenario, our proposal fills the gap providing automatic data transfers between different sites with a limited infrastructure. The work presented in this chapter is based on the HIDDRA architecture [55]. In that work, we presented the main components and general objectives of a data transfer architecture tailored to space science applications and a basic evaluation of its capabilities.

In this thesis, we present an extended architecture that shares the same design objectives of our previous work, but provides an improved scenario-independent solution. In this chapter, we focus on the Distribution Controller module which is responsible of 1) managing the distribution infrastructure, 2) processing events that trigger download notifications on the interested communities, and 3) efficiently downloading data from the available servers. As shown in Chapter 3, this module is now part of a general architecture that offers capabilities to detect virtual communities and to schedule the data transfer in order to improve the performance of the whole data distribution infrastructure.

Figure 41 shows a general overview of the distribution controller module. This component receives events such the availability of new data and uses the information from the other modules of the architecture to distribute it to interested users.

1 Large Hadron Collider expected data per year accessed on October, 2012. http://public.web.cern.ch/public/en/lhc/Computing-en.html

2 Very Large Telescope expected data per year accessed on October, 2012. http://www.eso.org/public/teles-instr/surveytelescopes.html

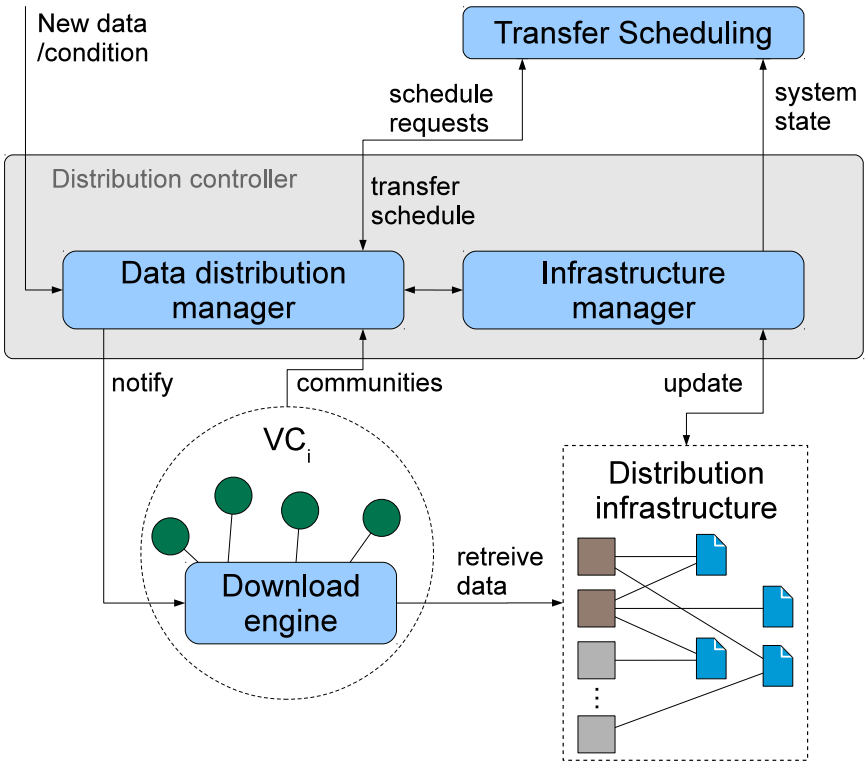


Figure 41: The distribution controller module uses the Transfer scheduling and Community detection modules to produce an efficient data transfer schedule once new data becomes available in the system.

The controller interacts with the Community Detection module (Chapter 4) in order to have an updated version of the community structure required during the matching of the subscriptions and creation of the notification messages to be sent. It also uses the Transfer Scheduling module (Chapter 5) to produce an optimized schedule of the data transfers to be performed based on the matching of the existing subscriptions. The distribution module is composed of three main components that are briefly described in the following paragraphs: *Data Distribution Manager*, *Infrastructure Manager*, and *Download engine*.

- **Data distribution manager**

  This event-driven component is responsible of managing the different steps involved in a data distribution scenario. The component awaits the arrival of new events or conditions and depending of the type of each of them triggers operations in other components of the Distribution Controller. Two main types of events are processed by this component: new data notifications and system messages.

  The data notification messages are sent by the publisher organizations to signal new data available in their servers. These data must be distributed to the subscribed users and to do that, several steps must be taken into account. First, based on the virtual community information and the subscription information, the target users are selected. This information is then sent to the Transfer Scheduling module in order to produce an optimized transfer schedule.

With this information the data distribution manager extracts the server-related information from the schedule and informs the Infrastructure manager whether new additional on-demand resources should be booted, and if intra-server transfers are required. After that, the notifications are sent to the download engines associated with the user communities following the defined transfer schedule.

The component is also responsible for processing other types of events such as system messages. These types of messages are sent by other components to inform about system changes such as the discovery of new virtual communities, new subscriptions, new publisher organizations, infrastructure changes, etc. Upon receiving these messages, the distribution manager may decide to trigger some actions or store the new information for future updates (e.g., new subscriptions).

- **Infrastructure manager**

  This component is responsible of managing the distribution infrastructures in three key areas. The main responsibility of the component is to monitor the state of the system. This includes the monitorization of the active server infrastructure and maintenance of an up-to-date catalog of which file are available at which servers. This information is used by the Transfer Scheduling module whenever a new distribution process is scheduled.

  The second responsibility of this component is to monitor the status of the current infrastructure. The component actively monitors the existing server infrastructure to have status information regarding server availability, server load, etc. For instance, in case of a significant server failure during a distribution process, this component may send an event to the data distribution manager to reschedule the transfers. Finally, this component is also responsible of performing the required intra-server transfers. These transfers are usually part of a transfer schedule and are performed before transferring data to the users. Additional intra-server transfers may be done in case of mirror failure or resource consolidation (e.g., shutdown of an on-demand resource).

- **Download engine**

  This component is responsible for downloading data on behalf of final users. It receives the notifications from the data distribution manager component and starts the required downloads. The engine is able to download files for different users, and therefore can be seen as a local proxy. The motivation behind having a local proxy is the fact that the connectivity inside an organization is up to several orders of magnitude faster than with the external sites. From an organization point of view, the cost of transferring files within the organization is significantly smaller than transferring data from the outside. Thus, the files are transferred once from the external sites to the organization, and then distributed to local users using a faster connection.

It is important to highlight that our Distribution Controller uses a publish/subscribe mechanism to determine which download engines will receive notifications based on the associated user subscription. The underlying implementation manages

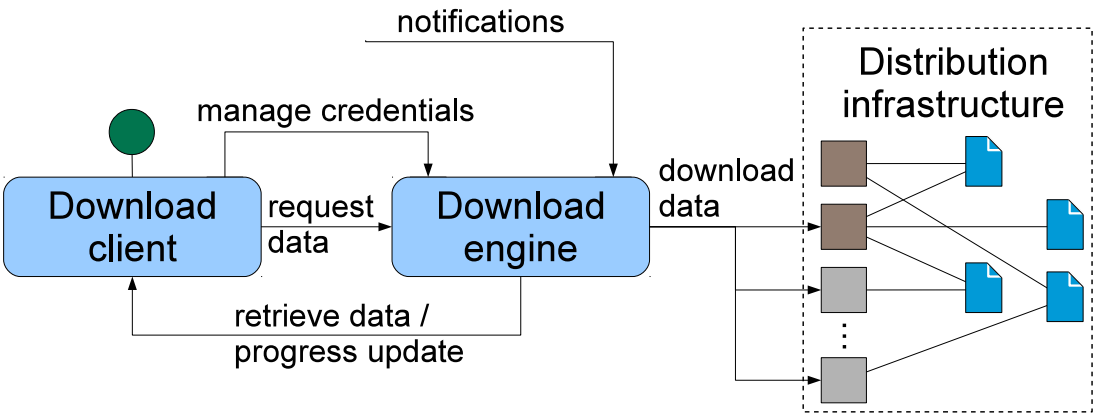


Figure 42: Detailed architecture of the Notification Module showing the two main components: Download Client and Download Engine.

the subscriptions combining two sources of information. First, the data distribution manager maintains the subscription information at user level (i.e., which type of files a particular user is interested in). Second, the download engines maintain a list of their local users. This information is sent to the data distribution manager in order to determine the subscription mapping. Notice that in this way, when two users are interested in the same file and they share the same download engine, only one notification is sent.

The remainder of the chapter is organized as follows. First, we present the functionality of the Download engine components in Section 6.1. The section details the multiprotocol parallel download mechanism, the internal architecture of the engine, the download lifecycle, the security aspects, and the method used to specify several download sources. After that, in Section 6.2 we show the evaluation of the proposed solution in terms of download performance and fault-tolerance capabilities.

6.1 DOWNLOAD ENGINE

The download engine is the component responsible for receiving notifications and downloading the requested data for its registered users. The download engine is a resource shared among the users of a virtual community. It behaves as a proxy for all downloads permitting to reduce the total number of transfers when two or more users request the same content. To access the download engine, each user has a *Client* component installed in their system. In this manner, users are not required to be online for their data to be downloaded in the proxy.

The interaction between the Download Engine and Download Client is shown in Figure 42 and can be summarized as:

- **Notification**: When new data become available in the system, the Data Distribution Manager matches the existing subscriptions and sends notification messages to the interested Download Engines. Upon receiving this type of message, the download engine starts the transferring process.

- **User requests**: The Download Client, even though our primary use case is the subscription based downloads, can send the engine interactive download requests. These requests specify the requested URL of the data to be downloaded and only affect the target Download Engine.
- **Data transfer**: Upon receiving a valid download request, coming either from the Data Distribution Manager or from the Download Client, the engine starts the download process described in Section 6.1.4. Before starting the download process, the download engine checks the validity of the request (Section 6.1.4.1) and its contents (Section 6.1.4.2).
- **Credential management**: The client interaction with the download engine requires a basic credential (i.e., username and password) in order to manage her downloads. Additional credentials, either basic or based on user certificates can be stored in the Download Engine in order to retrieve data from secure sites.
- **Progress update and data retrieval**: During a transfer, if the user connects to the Download Engine it will receive progress updates on the status of her downloads. Once the downloads are finished, the download client retrieves the required data from the download engine.

Based on the previous interactions, the lifecycle of a download process from a high level point can be summarized as: 1) a notification is received by the Download Engine, 2) the Download Engine access the required servers to transfer the data to its repository, and 3) the users connect to the Download Engine and perform local transfers of their data. The following sections provide detailed views of the inner workings of the Download Engine describing the main design principles and objectives, the benefits of using multiprotocol parallel downloads, its internal architecture and how different protocols are supported, the management of the download lifecycle, the security aspects surrounding the components and the specification of the downloads.

6.1.1 *Design objectives*

The design of the notification module is based on four design principles as our objective is to propose a distributed, autonomous, dependable, multiprotocol and extendable system. The following paragraphs describe our view of each principle.

- **Distributed**: The module is composed of different components designed to work independently of their location, performing the required communications using the existing network. A distributed deployment permits a better utilization of resources as each component can be located at its optimal location attending to its requirements (e.g., CPU, memory, available bandwidth). As an example, one Distribution Controller can manage the subscriptions of several virtual communities; each community has at least one Download Engine that communicates with several Download Clients installed in the machines of the users belonging to that virtual community.

- **Autonomous and Dependable**: All the components have been designed to work autonomously. This design reduces the administration overhead and improves the system availability. To provide this feature, different types of errors are supported by the system, and upon automatic reboots, the system recovers from the previous known state. This is particularly useful for the Download Engine. For example, in the event of a complete machine failure, upon rebooting, the engine recovers the previous download states and determines which portions of the file must be redownloaded to complete the transfer. Using this approach, we avoid redownloading the whole file when the failure affects only a small portion of the file. Additionally, we introduce a mirror scoring system, permitting the download engine to dynamically switch between different mirrors according to their current error rate. In this way, we reduce the impact of external errors on our system.

- **Multiprotocol**: Current solutions focus on using a single protocol to transfer a file. In our work, we introduce multiprotocol support which permits to concurrently use different available protocols (e.g., FTP, HTTP, HTTPS) to transfer different parts of the same file. This characteristic has two main advantages: it increases the compatibility with external systems and it removes the problem of installing custom protocols in the publisher organizations. In our proposal, the Download Engine uses a plugin system to facilitate the addition of new protocols. Notice that this approach also improves the backward compatibility as protocol updates in a organization can be introduced incrementally.

- **Extendable**: The architecture has been designed to be extendable through the use of plugins and new connectors. In this way, the effort required to support new protocols is reduced and only involves adding a new connector to the system. The main functionality of the Download Engine is abstracted from the particularities of each connector and does not need to be modified or updated in these situations.

These characteristics have been reflected in the design of the main components of the architecture: Distribution Controller, Download Engine and Download Client. Section 6.1.3 details the internal architecture of the Download Engine describing the aforementioned abstraction layer at protocol level.

6.1.2 *Multiprotocol parallel downloads*

The main idea behind the design of our Download Engine is to combine the use of parallel downloads with multiprotocol support. This combination permits to benefit from both techniques at the same time without sacrificing performance or flexibility.

The use of parallel downloads permits to increase the available throughput of a download process between two ends improving the fault-tolerance during the transfer at the same time. To demonstrate the benefits of using parallel connections, we study the maximum available throughput of a connection with and without parallel connections. In order to do that, we need to take into account the underlying transport protocol, as it is the key element to maximize the existing bandwidth provided by the technology of the link.

In our case, we study the performance of TCP as we are interested in using reliable protocols. In this respect, we use the estimation defined by Padhye et al. [84] which is based on the work of Mathis et al. [74]. To estimate the effective throughout of a TCP connection, the authors use the following expression:

$$B_{window} \approx \frac{W_{max}}{RTT} \tag{43}$$

$$B_{transfers} \approx \frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 \cdot \min(1, 3 \cdot \sqrt{\frac{3bp}{8}}) \cdot p(1 + 32p^2)} \tag{44}$$

$$B_{TCP} \approx \min(B_{window}, B_{transfers}) \tag{45}$$

where MSS is the maximum segment size, $Wmax$ is the maximum window size, RTT is the round trip time, b is the number of packets acknowledged per ACK, and p is the constant probability of a packet lost. Equation 45 estimates the bandwidth using two auxiliary equations. Equation 43 estimates the bandwidth considering the maximum window size and the round trip time. That is, the sliding window mechanism can only sent one full window of packets before receiving the first ACK. We expect this behavior to dominate Equation 45 for small values of RTT. On the other hand, Equation 44 estimates the bandwidth based on the transfer performance considering a packet lost probability.

In order to apply this formula to a scenario with η parallel connections, we modify the previous expression as:

$$B^{parallel}_{window} \approx \sum_{src \in S} \frac{\eta_{src} \cdot W_{max}}{RTT_{src}} \tag{46}$$

$$B^{parallel}_{transfers} \approx \sum_{src \in S} \frac{\eta_{src} \cdot MSS}{RTT_{src}\sqrt{\frac{2bp_{src}}{3}} + T_0 \cdot \min(1, 3 \cdot \sqrt{\frac{3bp_{src}}{8}}) \cdot p_{src}(1 + 32p^2_{src})} \tag{47}$$

where RTT_{src} is the round trip time with server src, and p_{src} is the probability of a packet loss when transferring data from src. In the previous formula we assume a common MSS as it is determined by the local end of the connection. Moreover, we need to take into account the limit imposed by the underlying interconnect B^{link}_{max}. Therefore, the effective bandwidth (Equation 48) could be estimated as:

$$B^{parallel}_{TCP} \approx \min(B^{link}_{max}, B^{parallel}_{window}, B^{parallel}_{transfers}) \tag{48}$$

Assuming a common server src it can be easily seen that:

$$B^{parallel}_{TCP} \geqslant B_{TCP} \tag{49}$$

which confirms that the upper bound for a transfer using TCP is larger or equal if we use parallel connections. To graphically illustrate this effect, Figure 43 shows the bandwidth estimation for 1, 2, 4, 8, 16, and 32 parallel connections in a scenario

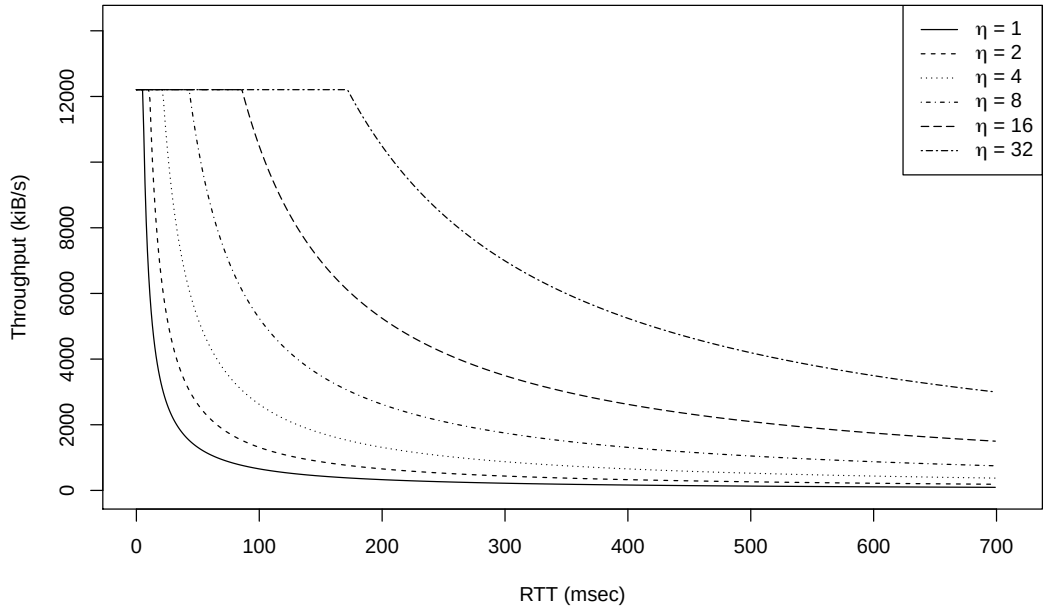


Figure 43: Maximum throughput estimation using parallel downloads using TCP.

where B_{max}^{link} corresponds to the maximum bandwidth of a Ethernet 100 Mbits connection, $W_{max} = 64\text{kiB}$, $b = 2$, $T_0 = 1$, $p_{src} = 10^{-6}$. As it can be seen in the figure, for small values of RTT, the maximum bandwidth is limited by the interconnect. As the RTT increase, the bandwidth decreases with a speed proportional to the number of parallel connections.

6.1.3 *Internal Architecture*

The Download Engine is the component responsible for managing the interaction with the clients and processing the incoming download notifications. This section details the internal architecture describing each internal component of the engine.

Figure 44 depicts the main internal components: *Download daemon*, *Workers*, and the *Local repository*. To describe the functionality of each internal component, we consider the use case of the download engine receiving a new notification and follow its processing.

The messages from the notification module are received through an available communication port that is continuously listening in the Download daemon. This component is continuously listening for new notifications or for incoming connections from the clients. Upon receiving a notification message and checking its correctness, the Download Engine decides whether the data needs to be downloaded or it is already available in the local repository.

If the requested data is available at the local repository due to being previously requested by any other users, the content is directly served from the repository to the local user. In order to grant access to the file, the user credentials are checked against

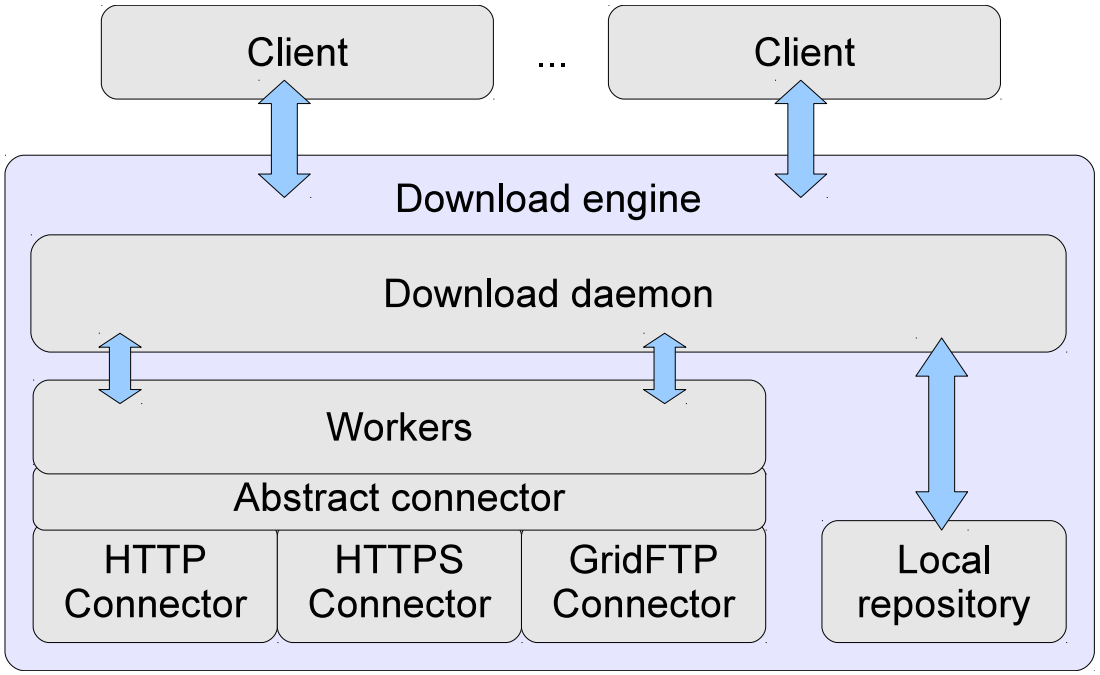


Figure 44: Download engine components.

the existing mirrors to determine whether the user can retrieve the stored copy or the data or not. The existence of a local repository also permits to reduce the number of transfers when more than one user is interested in the same data. In those cases, the data is transferred once to the local repository, and the users access the local copy assuming they have the necessary credentials. The storage space and the replacement algorithm (e.g., LRU, FIFO, etc.) can be configured by the organization administrator.

If the requested data is not available at the local repository, the Download daemon transfers the data to the repository. For this task, the Download daemon uses the download workers to retrieve the requested data. Each worker represents a set of connections to an external site with a single protocol. To manage the different protocols, an *Abstract connector* interface has been defined, so that every protocol that is supported by the download engine uses the same interface. This provides the capability of extending the number of supported protocols by adding new connectors. For example, if the data to be requested is accessible by HTTP, the download daemon will launch a worker with the abstract connector implemented by the HTTP Connector.

The Download Engine supports the utilization of several mirrors in parallel to transfer a single file (not necessarily using the same protocol). The selection of which mirrors participate in a particular transfers and which protocols are used is dynamically decided by the Download daemon. For this task, we employ Algorithm 4 to determine the number of workers that are launched connecting to a particular server. The algorithm receives three parameters: the download object (d) that contains all the associated metadata, the maximum number of workers to be launched ($max_{workers}$), and the failure threshold for a mirror ($failureThreshold$). Using this information, the algorithm retrieves the list of mirrors associated with the download ordered by their priority (line 3). Notice that the priority is specified in the notification informa-

tion sent to the Download Engine. Next, the algorithm iterates over the list of mirrors and for each mirror checks (line 8) that the number of failures does not overpass the failure threshold and that the mirror is alive at that time. Each time a chunk transfer fails, the score of that mirror is reduced. If the number of errors overpasses the established threshold the mirror is no longer used. Once the algorithm decides that a mirror can participate in a download, it adds a number of workers (lines 9-10) that corresponds to the minimum between the number of pending worker slots for the download and the number of available slots remaining for that mirror. To calculate this number, each connector specifies the maximum number of connections that can be opened with a server using a particular protocol.

Algorithm 4 Algorithm to launch the necessary download workers taking into account the mirror failure.

```
Input: d /* Associated download */
Input: max_workers /* Maximum number of workers to be launched */
Input: failureThreshold
   result = ∅
   /*Get the mirror list ordered by preference */
 3: mirrors = d.getOrderedMirrors()
   index = 0
   pending = max_workers
 6: while index < mirrors.size() ∧ pending > 0 do
      m = mirrors[index]
      if m.failures > failureThreshold ∧ m.isAlive() then
 9:     numWorkers = min(pending, m.availableConnections())
        for i ∈ numWorkers do
          result.add(newWorker(d, m))
12:     end for
        pending− = numWorkers
      end if
15:   index++
   end while
   return result
```

This approach introduces a fault-tolerance layer against server failures in the Download Engine without requiring any type of user intervention. In the event of mirror failures or maintenance situations, there is no need of resending the new list of available servers to the interested Download Engines. For example, let us consider a scenario with two mirrors transferring different parts of the same file. If one of them fails, the Download Engine will try to reconnect a configured number of times. If reconnection is not possible, the remaining pieces of the file will be downloaded from the other mirror.

6.1.4 *Download lifecycle*

The Download Engine manages the download processes using a finite state machine that covers the complete download lifecycle. Figure 45 contains all states of a trans-

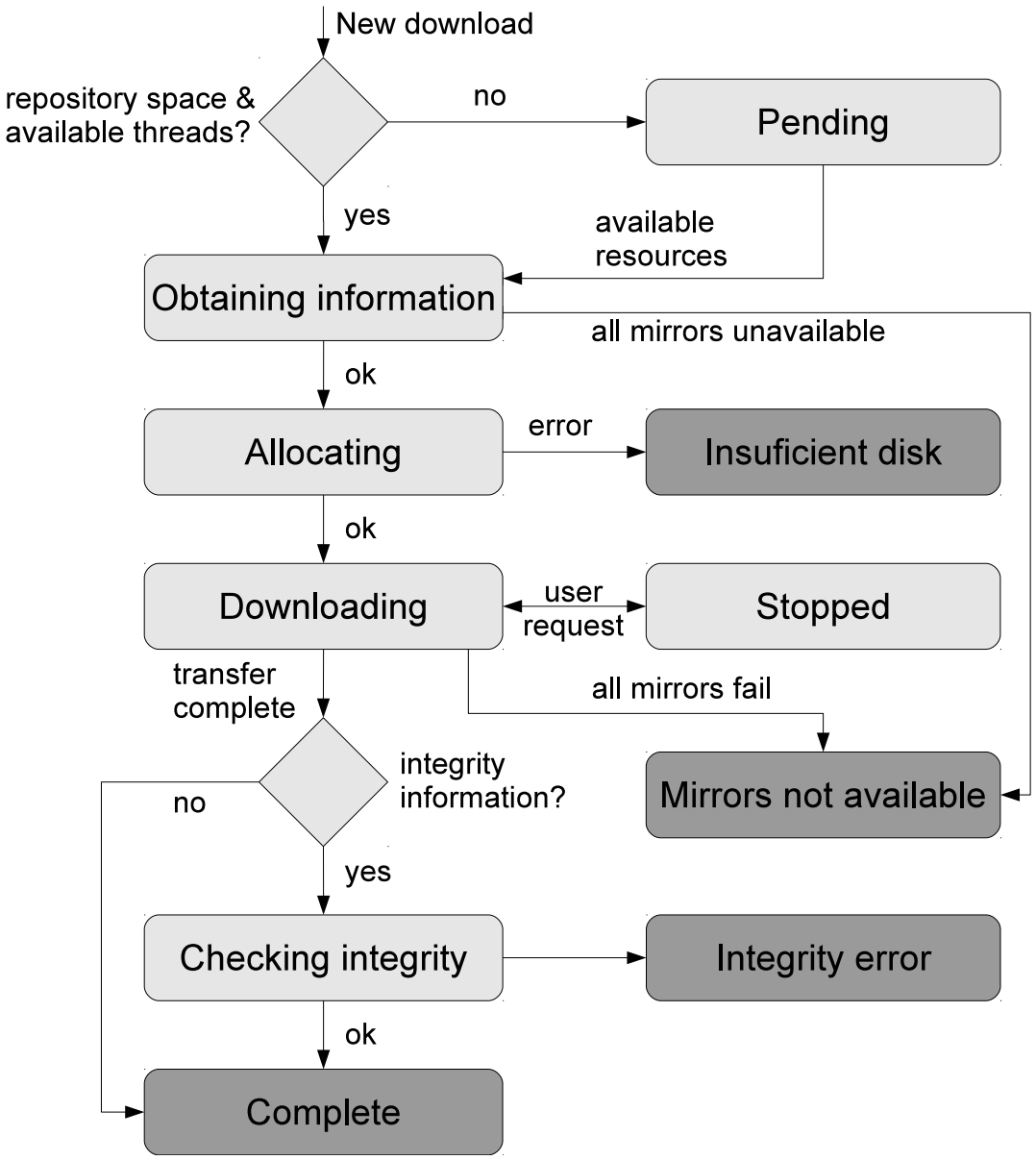


Figure 45: Download states diagram. Darker boxes represent final states of the download.

fer process including the initial, final and error states. Notice that the lifecycle of a download is abstracted from the underlying protocols and configurations that will be used to transfer the data.

The lifecycle of a download starts with the arrival of a new download request. Before starting the download, the Engine checks whether there is enough space in the local repository and there are available slots for the download to start immediately. If any of the conditions is not satisfied, the download goes to the *Pending* state. When these conditions are fulfilled, the download makes a transition to the *Obtaining information* state. At this point point, the Download Engine identifies which mirrors are available and retrieves the size of the download. If no mirror is available, the download passes to the *Mirrors not available* error state. This information could be used by the users of the system or the Download Engine itself to report errors to the publisher organizations.

If the download size can be obtained and at least a mirror is active, the Download Engine uses this information to compute the number of chunks to be downloaded. To calculate this number, a $\mathtt{chunk}_{size}$ value is configured into the Download Engine. The number of chunks (which matches the number of connections if no error appears) is then calculated as:

$$num_{connections} \geqslant num_{chunks} = \left\lceil \frac{f_{size}}{\mathtt{chunk}_{size}} \right\rceil \tag{50}$$

The assignation of which chunk will be transferred by which connection is done by defining a pool of available connections. When a connection is not transferring any data it will request another chunk to be downloaded until all chunks in a download have been transferred.

However, as the Download Engine has a maximum storage space for the repository, it is necessary first to go into the *Allocating* state, where physical disk space is reserved. If there is no sufficient space or there is a disk failure, the download makes a transition to the *Insufficient disk* error state. After a successful allocation, the download goes into the *Downloading* state. In this state, each of the chunks defined before is assigned and transferred. Upon user request, the download can be *Stopped* and resumed. If during the download process, a complete mirror failure is detected, the download goes into the previous *Mirrors not available* error state. On completion, the integrity information metadata of the download is checked. If it exists (e.g., a hash of the file), the download makes a transition to the *Checking integrity* state. If not, the download is considered successful and goes to the final *Complete* state. If the integrity check fails, the download goes into the *Integrity error* state.

The aforementioned figure contains the main state diagram. For clarity purposes, some error states have been omitted. In case any error is detected during any stage, the download will make a transition to the *Error* state. Additionally, if the download contains extended integrity information (e.g., partial hashes of the file), the downloaded chunks will be checked during the *Downloading* state to determine whether it is necessary to re-download any of them. This permits to avoid re-downloading the whole file when one chunk fails. In case of Download Engine failure (e.g., power outtake), once the engine is restarted, the downloads will continue from the previous known state.

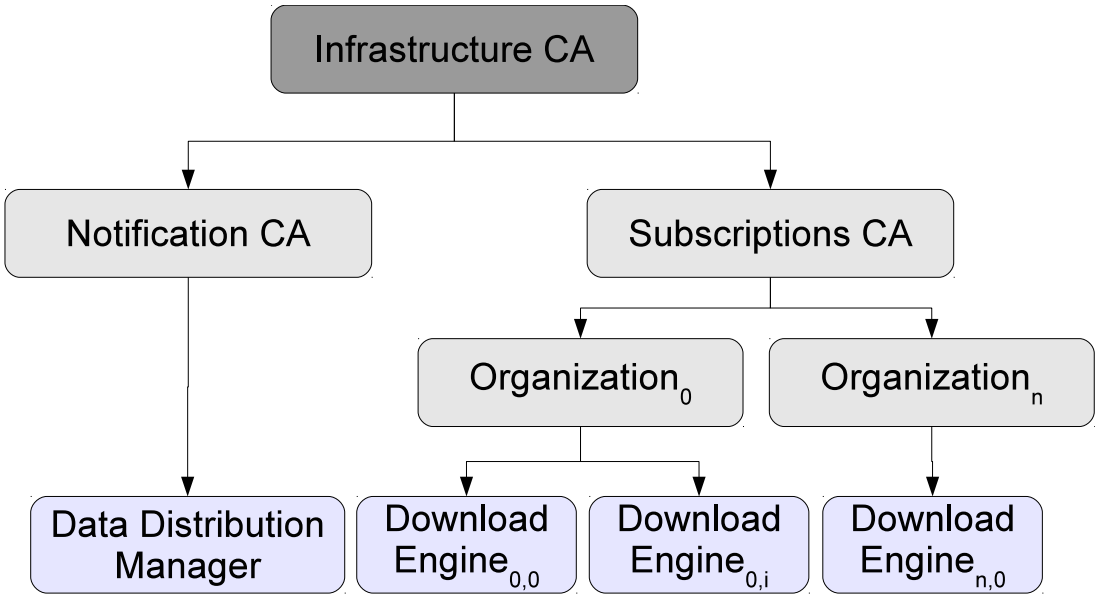


Figure 46: Public key infrastructure for secure notifications. Starting from the infrastructure certification authority, a hierarchy is built in order to delegate the management of local certificates at the final organization level.

6.1.4.1 *Security aspects*

The download engine has been modeled considering the requirement that it must be able to download data from sites that require some type of authentication. Therefore, we must provide a mechanism to manage the user credentials on the Download Engines. Moreover, as the engines rely on a notification mechanism to launch new download, it is important to authenticate the notifications. The main security aspects addressed in the initial design are:

- *Secure notifications*: Notifications send to the download engines must be send from trusted endpoints in order to avoid denial of service attacks making the Download Engine transfer non-desired files. In this type of attack, a malicious endpoint will send notifications to a Download Engine to download a set of files that the user has not requested. In this way, it would be possible to make the Download Engine use all available download slots to retrieve this non-requested data, inducing starvation in the legitimate user requests. In order to implement this mechanism, we employ a public key infrastructure. Figure 46 depicts how the certificates of different components of the system are managed. On the top of the infrastructure, a global certification authority is used to create two new certification authorities. The first one is called *Notification CA* and it is used to issue certificates to the components responsible of sending notifications. The second one receives the name of *Subscriptions CA* and it is used to issue certificates that will permit the download engines to authenticate themselves. In order to facilitate the management of the download engines certificates, a certificate is issued to the trusted organization, and it is the responsibility of each organization administrator to manage its certificates.

- *User credentials*: The users of the system retrieve the files from the download engines using their download clients. To connect to the download clients the users need to specify a username and password to establish an authentication and establish a connection. This username and password are only used by the clients to connect to the download engines. Additionally, the users are able to store other credentials in the download engines in order to access external sites. The system currently supports the storage of username and password and user certificates. The utilization of these credentials by the download engine works in a similar manner as the credentials stored in a common web browsers.

6.1.4.2 *Specifying download sources*

Our Download Daemon is able to download a file using different protocols concurrently from a set of available servers. It is therefore necessary to define a user-friendly method to specify which are the servers and protocols that can be used to retrieve a particular data. For this purpose, we employ the metalink [3] standard. A metalink is a XML document where the necessary information for a multi-source download is specified. The metalink specification considers that a download can be composed of several files. For example, the download of a new kernel release (e.g., 3.0) could involve the download of several files: System.map-3.0.0, vmlinuz-3.0.0, initrd.img-3.0.0, and config-3.0.0. The metalink specification permits to define a single metalink file (kernel-3.0.0.metalink) that contains all the information required to download all files.

An example of a metalink file is shown in the Listing 1. First, we specify the information about the contents of the download such as the content identifier, version, publisher, license, description, etc. After that, the download information is presented. As multiple files can be specified in a metalink, we will focus on the information contained in a single file.

For each file, the metalink file contains the integrity information and the list of available resources to obtain the data. The integrity information is an optional field and contains a hash per file, and a list of hashes per chunk of the file. This permits to check the integrity of the file as it is being downloaded and upon completion of the transfer. The list of resources specifies where we can obtain the data from. Each resource is characterized by its protocol (*url type*), the *location* of the mirror and the *preference*. The protocol definition permits to automatically choose which is the connector suitable for the connection. The preference and location values can be used to establish an order of preferred mirrors and how many connections can be made to each of them.

Listing 1: Sample metalink

```
<?xml version="1.0" encoding="utf-8"?>
<metalink version="3.0" generator="Metalink Editor version 0.4.1" xmlns="http://
    www.metalinker.org/">
  <publisher>
    <name>Publisher organization</name>
    <url>http://www.publisher.com/</url>
  </publisher>
```

3 Metalink standard. www.metalinker.org/

```
  <identity>Test image</identity>
  <version>1.0</version>
  <copyright>Copyright Information</copyright>
 <description>File 1 description.</description>
 <files>
   <file name="file1.dat">
     <size>5410865</size>
     <verification>
               <hash type="sha256">685a82c...f62</hash>
       <pieces type="sha1" length="262144">
         <hash piece="0">51777f8...7c2</hash>
         ...
         <hash piece="20">5075863...0cd</hash>
       </pieces>
     </verification>
     <resources>
       <url type="http"   location="es" preference="70">
            http://mirror1/dir1/file.iso</url>
        <url type="http"   location="uk" preference="70">
            http://mirror2/dir1/dir2/f.iso</url>
        <url type="gsiftp" location="es" preference="20">
            gsiftp://backup.mirror1/file.iso</url>
     </resources>
   </file>
 </files>
</metalink>
```

In this thesis, we modify the standard metalink schema to permit the use of the GridFTP protocol by introducing the *gsiftp* type. Additional protocols can be added to the metalink specification in the same way. Regarding the use of the integrity information, the Download Engine it is able to check the integrity of the file while it is being downloaded using the hashes found in the metalink. Notice that the chunk size in the metalink may not match the chunk size used by the engine. In this case, the Download Engine determines the equivalence between the metalink chunks and the download chunks and waits until the involved pieces are transferred to perform the integrity checks.

6.2 EVALUATION

In this section, we present the evaluation of the proposed Download Engine. First, we present the results obtained when using the parallel download mechanism presented in this chapter. After that, we present results regarding the fault-tolerance capabilities of the engine when presented with different types of failures. All the tests have been carried out deploying the Download Engine in a machine with a 100 Mbps ethernet connectivity using the HTTP connector. We will refer to that machine as *client* in the following sections. To capture the information, we use the well-known network protocol analyzer Wireshark [4] 1.8.3 limiting the capture to TCP packages.

4 Wireshark: http://www.wireshark.org/

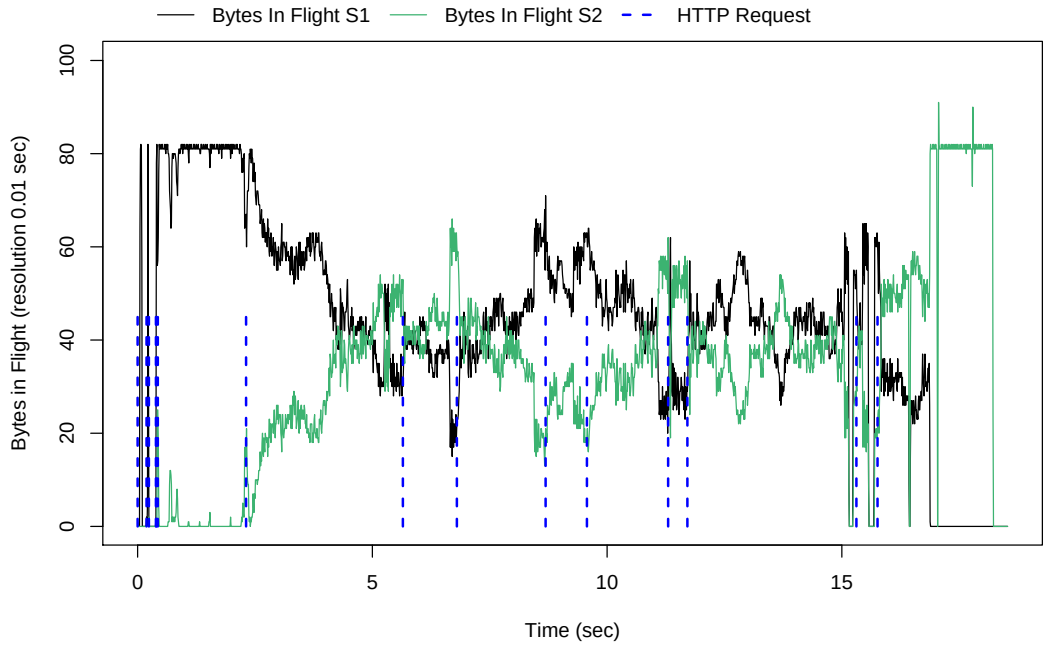


Figure 47: Parallel download performance using two servers. Notice that the sum of the bandwidth from each server matches the maximum effective bandwidth of the Ethernet link.

6.2.1 *Parallel downloads*

In this section, we explore the benefits in terms of speedup of using the proposed parallel download mechanism. Before entering in details about what could be the achievable speedup, we want to illustrate the inner workings of the mechanism. In order to do that, we capture the traffic between the client machine and two servers (S1 and S2) located in the same LAN. The maximum connectivity with the servers is 100 Mbps, and both of them can be accessed in less than 3 hops. Additionally, the system is configured to launch 4 parallel connections with 2 parallel connections per server.

To detail the timeline of the transfer, Figure 47 shows the number of bytes transmitted per server and unit of time (resolution of 0.01 sec) and a vertical line each time a new HTTP request is sent. As the download engine splits the file into chunks, ideally the number of HTTP requests should match the file size divided by the chunk size. This number may vary in reality depending on the error rate of the connection, number of retransmissions, etc. The first part of the figure shows the startup of the download process, with the Engine configured to launch connections to S1 and S2 in that order (the priority value in the metalink file). At first, all requests are sent by S1 due to the ordering. Notice how the effective bandwidth is limited only by the ethernet connectivity of the client.

Approximately, at time 2.5 seconds, the connections established with S2 start receiving data. In the interval 5 to 15 seconds, we observe how the client is accessing

both servers at the same time. It is important to highlight that the effective bandwidth is split between both servers as both have the same connectivity capabilities and the sum of the bandwidths matches the client ethernet limit. Around time 12 seconds, the threads connecting to S1 finish downloading their assigned chunks, and only connections to S2 remain active. After this, connections with S2 finish and the requested file is available at the Download Engine.

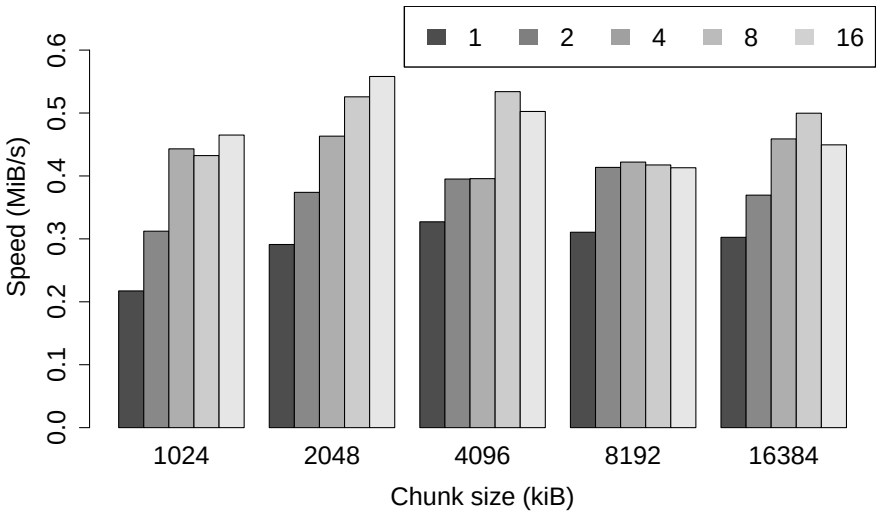
After illustrating the basic mechanism, we are interested in analyzing the possible benefits of varying the number of parallel connections and the chunk size in the Download Engine. For this purpose, we compare the average download speed when transferring the same file from three different mirrors. In the test, we vary the number of connections between 1 and 16, and the chunk size between 1024 kiB and 16 MiB. The test file corresponds to the core OpenOffice file (*OOo_3.3.0_src_core.tar.bz2*) which has a size of 105 MiB. The servers involved in the test are selected from the official mirror list [5]. The main characteristics when selecting the servers are the RTT with the client and the distance in hops between the client and the server (as measured with the traceroute tool). In our case, we choose three servers with a RTT of 265, 40, and 45 milliseconds, and a hop distance of 16, 13, and 11 hops respectively.

Figure 48 shows the average speed when accessing each of the servers. First, in Figure 48a we show the download performance when connecting to a slow server with an RTT of 265 milliseconds. The results show the benefits of using parallel downloads in this type of servers as the average speed increases in most cases with the number of connections. Some cases with 16 connections show slowdowns with a performance similar to 4 or 8 connections. Regarding the chunk size, we observe that in general medium size chunks of 4096 KiB offer the best performance. The average speedup in this server when using parallel connections is 1.74 ± 0.30 with a maximum speed of 0.55 MiB per second.

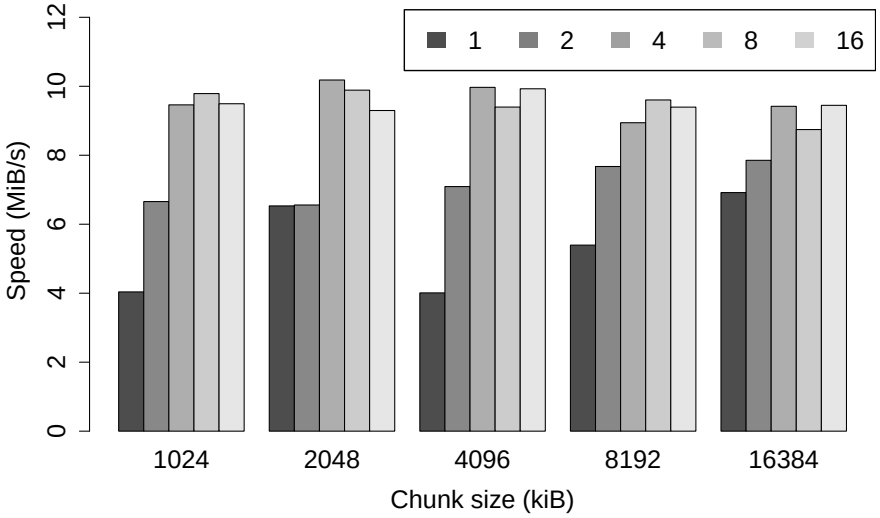
In Figures 48b and 48c we show the results when connecting to faster servers with an RTT of 40-45 milliseconds. In both cases, medium sized chunks of 4096 KiB and 8192 KiB offer the best performance. Regarding the number of parallel connections, we observe an average speedup of 1.93 ± 0.50 and 2.95 ± 1.39 respectively. As before, we also notice that with 16 connections, the speed usually decreases. It is important also to highlight that the maximum speed in both test is 9.39 and 10.18 MiB per seconds respectively, which are significantly close to the maximum speed of the ethernet local link.

In all scenarios, the results suggest that using small sized chunks do not provide the best configurations. While smaller chunks provide faster responses on the event of failures (i.e., retransmission involve less data), they impose a significant overhead in the connection. To show this behavior, Figure 49 shows the number of connections to be established depending on the size of the file and the selected chunk size. As it can be shown, smaller chunks have a significant impact in the number of connections to be opened. In this respect, when transferring large files, we are interested in reaching the steady state of the TCP connections. Therefore, medium size chunks provide the best tradeoff between the number of connections and the impact of retransmissions. With regards to the number of connections, we have also seen that 16 connections usually provide less effective bandwidth. This situation can be explained by different factors such as the overhead of opening new connections or the

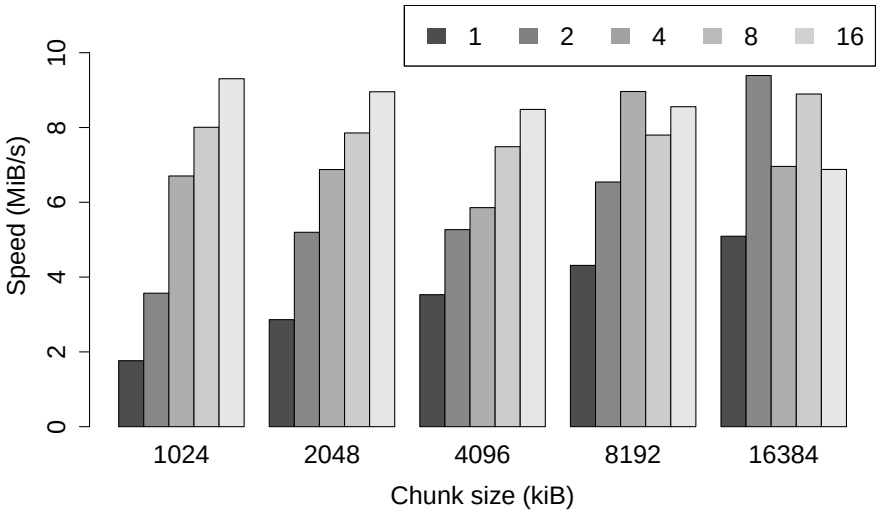
5 OpenOffice mirror list: www.openoffice.org/distribution/mirrors/master.html



(a) Slow server with a RTT of 265 milliseconds and a distance of 16 hops.



(b) Fast server with a RTT of 40 milliseconds and a distance of 13 hops.



(c) Fast server with a RTT of 45 milliseconds and a distance of 11 hops.

Figure 48: Download speed depending on the number of parallel connections and the chunk size for different servers. Each bar is colored according to the number of parallel connections launched.

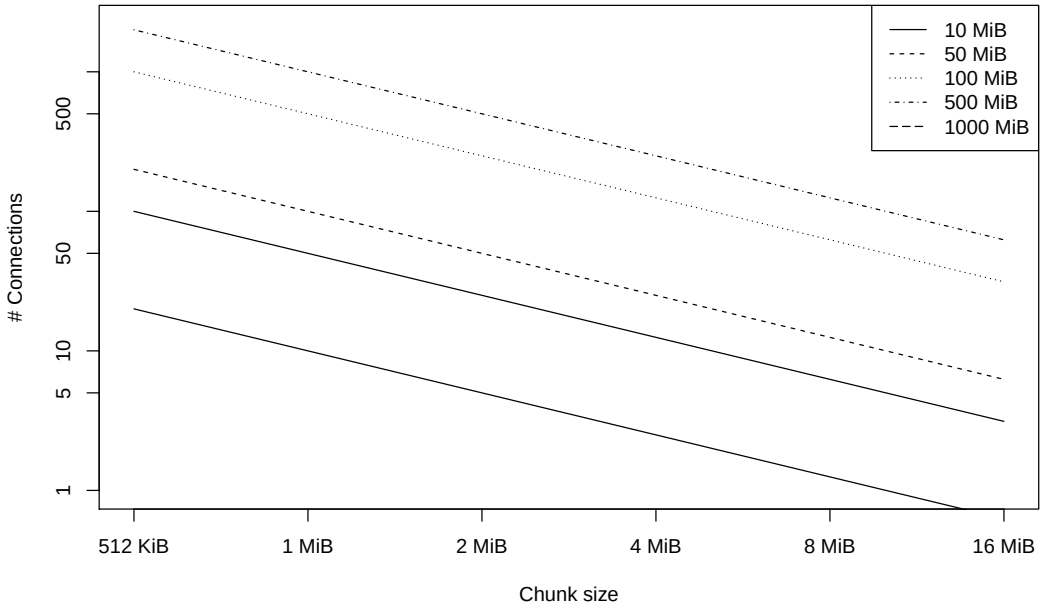


Figure 49: Theoretical number of connections depending on the chunk size for various file sizes.

effect of having a significant number of concurrent connections with a single server. Internet etiquette usually suggests not opening more than 4 parallel connections to a server. This is motivated by the fact that servers must be able to respond request to multiple users and too much connections with a single user can monopolize all resources causing starvation to other users.

6.2.2 *Analyzing fault tolerance in the event of mirror failures*

After showing the advantages of using parallel downloads for improving the bandwidth utilization, we focus on the fault tolerance capabilities of the Download Engine. For these experiments, we employ 3 scenarios with different levels of complexity. The first scenario depicted in Figure 50 consists of a single server that experiences a transient connectivity failure. For example, imagine that the server is unreachable due to an intermediate router being overloaded, or the client disconnecting momentarily from an unstable wireless connection. In this scenario, the client starts sending requests for content and the download is stable until the failure zone is reached. During this period, the client tries to reconnect to the server, and some retransmissions are sent. When the connectivity becomes normal again, the client receives retransmissions from lost packages and the download is recovered. Notice how the effective bandwidth during normal operation is bounded by the ethernet link between the client and the server.

The second experiment involves a scenario with 2 servers (S1 and S2) and during the transfer S1 suffers a permanent failure. For example, imagine that the server

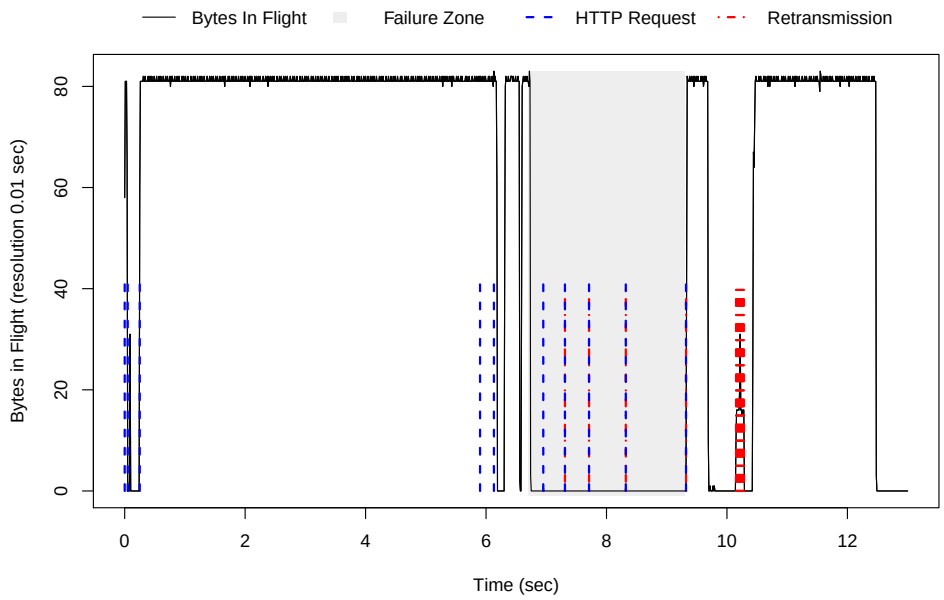


Figure 50: Fault tolerance on the event of transient failures. Mirror S1 suffers a transient failure and the Download Engine recovers the transfer process as the connectivity is restored.

suffers a hardware crash or the facility suffers a blackout. Figure 51 depicts how the Download Engine manages the connections in this situation. When the download starts, connections are established first with S1 and then with S2. When the connections to S2 start receiving data, S1 crashes. We observe that in this situation a significant amount of retransmissions is sent to try to retrieve data. During this failure, connections to S2 continue to receive data and as shown in the figure, the effective bandwidth starts being consumed by S2. Notice how the bytes in flight are at first dominated by the connections to S1, and after the failure S2 consumes all the available bandwidth.

The third experiment is a modified version of the previous one to demonstrate the use of backup mirrors when necessary. For this scenario, the Download Engine is configured to launch 4 connections per server, thus using only one of the available servers. The first part of the trace shows transfers from S1 until the system crashes. In this state, the connections are not closed by the server and the TCP mechanisms for this type of failure are activated. In particular, the TCP standard mandates to wait 2MSL (Maximum Segment lifetime) in order to determine that the connection is lost. The download engine awaits this situation and once is detected, new connections to S2 are launched to finish the download. Notice that the value of MSL changes between different implementations (between 30 seconds and 4 minutes) and we cannot avoid waiting 2MSL in case the connection is reestablished. The same mechanism is used in the first scenario and permits to support transient failures without reconnections.

In summary, the fault tolerance capabilities of the Download Engine permit to recover from external failures while maintaining a maximum effective bandwidth

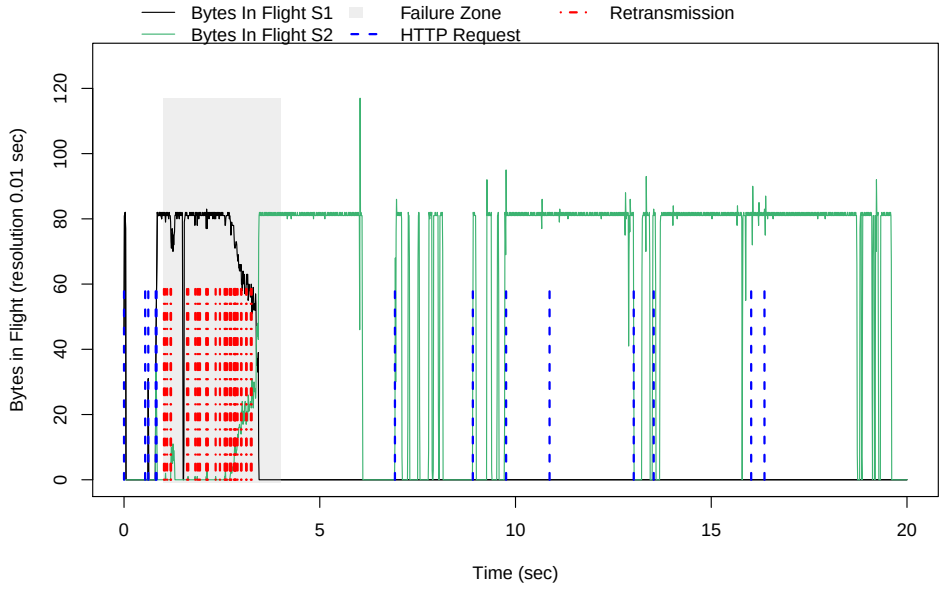


Figure 51: Fault tolerance on the event of permanent failures. Mirror S1 suffers a permanent failure once the download is in progress. As the Download Engine used two concurrent servers at the same time, the transfer is finished from the remaining connections to S2.

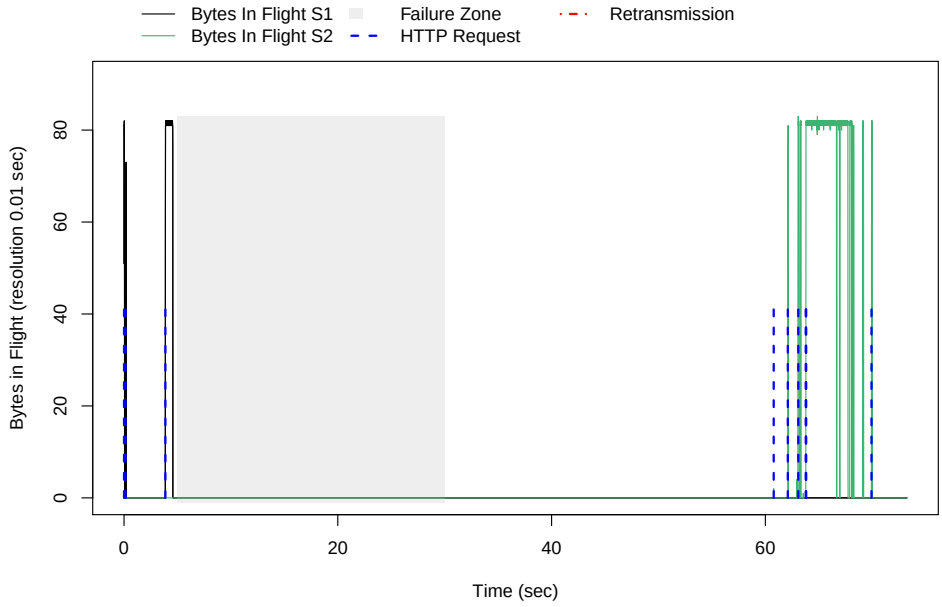


Figure 52: Fault tolerance on the event of permanent failures with a backup server. The transfer is interrupted as S1 hunts. After the TCP established timeout of 2MSL is done, new connections are launched to S2 to finish the transfer.

utilization during the downloads. Notice that no user intervention is necessary in any of the cases.

6.3 SUMMARY

In this chapter we have presented the Distribution module of the proposed architecture. This module is responsible of managing the whole distribution infrastructure and processing the events that trigger download notifications in the interested communities. It is also responsible of providing an efficient fast data transfer mechanism in the form of a Download Engine. Our engine behaves as a proxy for the users of a virtual community and it is able to perform multi-protocol parallel connections. Using this approach, we can transfer several pieces of the same file concurrently using several concurrent connections to each server, and different protocols at the same time.

To demonstrate the benefits of our approach, we first provide a theoretical upper bound of the effective bandwidth when using parallel downloads. Our evaluation shows the benefits of parallel connections in two aspects. First, our results show that the effective bandwidth of a download is equally distributed on the accessed servers. In this way, we reduce the impact on the infrastructure and maintain the quality level on the user end. Second, we evaluate different failure scenarios to demonstrate the fault-tolerance capabilities of our proposal. In this respect, we show how the download process can be restored in the event of complete mirror failures by changing to alternate servers, in the event of a mirror failure in a multi-server scenario, and in the event of transient failures with a single server.

7

FINAL REMARKS AND CONCLUSIONS

In this thesis, we have proposed an **efficient and reliable data distribution architecture that leverages the knowledge extracted from the user community in order to improve the data movement performance**. In Chapter 1.2 we decomposed the main objective into three main areas, that have been addressed throughout this document.

In Chapter 3 we introduce the data distribution problem and describe its main challenges. We present our proposed data distribution architecture from a high level point of view and describe the applicability of our proposal in several scenarios. Chapter 4 focuses on providing a method to **analyze social networks in order to extract knowledge that can be leveraged in the data distribution process**. In this respect, we propose to apply well-known community detection methods to the social networks inferred in data distribution scenarios. The different algorithms are evaluated using several metrics, and an iterative weighted community detection algorithm is proposed to overcome the detected limitations. We also address the possibility of filtering out certain relationships in the social network, and study the impact on the resulting virtual communities.

The second sub-objective of this thesis is to **provide a reliable and fast data transfer mechanism**. This goal can be addressed at different levels depending on whether we consider the point of view of a single user accessing some data or we consider the fact that we would like to improve the overall performance of the data distribution process when several users request data concurrently. In this respect, in Chapter 5 we address the problem of data transfer scheduling. Our approach consists on formalizing this problem as a linear programming model. Due to the complexity of the problem itself and the fact that solving performance is a key factor, we devise a feasibility formulation of the original problem. The new model makes use of a heuristic to limit the solution space and permits to produce a distributed solving by using a map-reduce implementation. In this chapter, we also evaluate the complexity of solving this type of problems in practice and provide an evaluation of the quality of the resulting schedules.

In Chapter 6 we address the previous objective by defining a download engine that is able to use different protocols concurrently to retrieve the same file and also provides fault-tolerance mechanisms in case of failures. The last sub-objective of **defining a novel architecture that exploits social knowledge to improve the data distribution process** is also addressed in this chapter. We show a complete architecture that uses the virtual communities and the data transfer schedule from previous chapters and defines a distribution controller that is able to orchestrate and manage data distribution scenarios.

The remainder of this chapter is organized as follows. First, in Section 7.1 we detail the contributions of this thesis. Second, in Section 7.2 we present future lines of research that are opened by the results of this thesis. Finally, Section 7.3 concludes presenting the results of this thesis in terms of publications, research internships, and research grants.

7.1 CONTRIBUTIONS

The main objective of this thesis as presented in Chapter 1.2 is **to define a new efficient and reliable data distribution architecture that leverages the knowledge extracted from the user community in order to improve the data movement performance**. In this respect, the main contributions of this thesis are:

C1 **Data distribution architecture**: Addressing the main objective of this thesis, we have defined a complete solution for data distribution scenarios. In this document, we have presented a new architecture that employs an underlying workflow to efficiently distribute data. First, we extract knowledge from the social network of users requesting data by defining an overlay of virtual communities. The definition of these communities permits to optimize the distribution process reducing the total number of transfers from the servers. Second, our architecture employs a publish/subscribe paradigm to determine which users should be informed as new data becomes available. By leveraging this paradigm, we open the possibility of assuring the service level requested by each user by creating a prioritized schedule. Third, our architecture employs a scheduling component to define the distribution plan. In this way, we can optimize the use of resources avoiding hot spots in the infrastructure. Finally, our architecture contains a fast data transfer solution that is able to concurrently employ several protocols to retrieve data.

C2 **Iterative weighted community detection algorithm**: The evaluation of the existing community detection algorithms showed that the quality of the resulting partitions is not suitable for data distribution scenarios. In particular, in this type of scenario we would like to control de maximum number of users assigned to a community as well as the number of resources that are going to be accessed per community. Our analysis of the results suggests that the main reason behind this behavior is that most algorithms rely on the modularity metric to determine how network should be partitioned. Due to the high-connectivity nature of data distribution networks, we propose an iterative algorithm that mitigates this problem. The algorithm combines the partitions defined by any community detection algorithm with user configured parameters to determine whether a partition must be iterative split into smaller ones. Our evaluation demonstrates the advantages of this solution as shown by the high quality of the produced communities.

C3 **Formalization of the file transfer scheduling problem**: The first challenge when deciding to solve a problem using a linear programming approach is its formalization. In this respect, the quality of the solution highly depends on how

complete is the model with regards to the real problem. As an abstraction process, the formalization also has the benefit of providing a better understanding of the inner workings of the problem to the author. In our case, we address the problem by building our model using elements from a well known scheduling problem. However, as the results show, successfully modeling a problem is not the only challenge, as we need solutions in a reasonable time. In order to address this issue, we refine our model by converting the original problem into a feasibility problem with the help of a heuristic. By doing this, we are able to significantly reduce the time required to solve a scheduling problem. Additionally, we have also complemented the problem to introduce different levels of fault-tolerance in the schedule.

C4 **Distributed solving of the transfer scheduling problem**: The definition of the aforementioned feasibility problem permits to use a divide and conquer approach to create a transfer schedule. In this thesis, we introduce the map-reduce paradigm and provide a tractable solution to permit the distributed solving of a scheduling problem. To the best of our knowledge, it is the first time this combination of feasibility formulation and map-reduce approach has been used to distribute the solving of a linear programming problem.

C5 **Parallel multi protocol download engine**: In this thesis we have introduced a novel parallel multi protocol download engine to speedup the transfer process. Our engine provides two main characteristics. First, it is able to support parallel connections to the same or different servers in order to retrieve difference pieces of a file concurrently. Second, it supports the use of multiple protocols at the same time. This permits to use more mirrors in a transfer independently of the underlying protocol. The download engine supports different failure scenarios providing adequate fault-tolerance to recover or maintain the transfers. Additionally, our download engine offers access to secure remote sites using basic and certificate credentials.

7.2 OPEN RESEARCH DIRECTIONS

The results obtained during the realization of this thesis open several research directions that can be explored in the future as a continuation of the presented work.

ONLINE COMMUNITIES

- **Propose a new algorithm that is not based on the modularity metric**. As shown on Chapter 4 most community detection algorithms try to maximize the modularity of a network. Our results demonstrate that the value of this metric may not provide enough information in networks that show a high connectivity degree among nodes. In this respect, a possible future research line would consider the definition of a new community detection algorithm based on non-modularity metrics.
- **Incremental community detection algorithms**. A social network is an evolving entity that may change its structure, its size or its connectivity

degree over time. In this respect, a future research line would explore different approaches to permit incremental application of the community detection algorithms.

FILE TRANSFER SCHEDULING

- **Definition of new heuristics**: The results presented in Section 5.4.2 demonstrate the importance of selecting a good heuristic for determining the Tmax value of a problem. In this respect, a future research line would explore the definition of improved heuristics to maximize the probability of assigning values of Tmax that reduce the computational complexity of solving the scenario.
- **Additional minimization objectives**: The model presented in Section 5.2.1 tries to minimize the makespan of a schedule. However, other objectives may be of interest. In particular, we identify three main objective that may be pursued in the future: 1) To reduce the number of intermediate transfers of a file in order to increase the fault-tolerance. 2) To characterize the energy consumed by the different elements of the system: servers, user destination and intermediate appliances such as routers and minimize the total energy consumed. And 3) to consider the cost of using different types of servers when using on-demand resources.
- **Effect of the hardware on the solving process**: Scalability has been proven to be one of the most important challenges presented by current state-of-the-art solvers. In this respect and based on our initial experiments presented in Section 5.4.6 a future research line would explore the benefits of using different hardware architectures with regards to the solving computational time. In particular, we are interested in studying the benefits of multicore architectures and analyze the effect of the different cache hierarchies (number, connectivity, and size).
- **Comparison between reactive and proactive scheduling**: In Section 5.2.1 we presented a model that is able to produce a schedule given the current state of the system. This model needs to be reactively executed each time a new condition changes that may affect the current schedule. By contrast, on Section 5.3.2 we presented an extension of the model that permits to introduce fault-tolerant capabilities in the schedule. In this respect, we are interested in studying the benefits of each technique when presented with changing scenarios. In particular, we want to determine which is the overhead impose by the reactivity of the first model, and compare it to the overhead of solving a more complex scenario with the proactive model.

FAST DATA TRANSFERS

- **Adaptative download fault-tolerance**: The evaluation of the Download Engine presented in Section 6.2 demonstrated the fault-tolerance capabilities of the engine when presented with different types of failures. While some failures can be detected rapidly, others involve timers in the TCP layer outside of our control. In this respect, we plan to extend the existing

fault-tolerance capabilities by reducing the delay required for recovering a download process. In order to do that, we plan to add a monitoring layer on the download engine that is able to actively register the evolutions of the downloads. This will permit to detect other type of situations such as low performing servers, or automatically decide that a connection is lost without requiring the TCP timer to expire.

7.3 THESIS RESULTS

This section presents the published results of this thesis, as well as the effort put on the dissemination of the results. The section also details a research internship and other research performed during the realization of this thesis.

7.3.1 *Publications*

The main results of this thesis have been published in international journals and conferences related with the studied topic. We enumerate the different references in the following paragraphs.

JOURNALS

- D. Higuero, J. M. Tirado, J. Carretero, F. Felix, and A. Fuente "HIDDRA: A Highly Independent Data Distribution and Retrieval Architecture for Space Observation Missions", *Journal of Astrophysics & Space Science*, vol. 321, issue 3, pp. 169 - 175, 2009.
- J. M. Tirado, D. Higuero, F. Isaila, J. Carretero, and A. Iamnitchi, "Affinity P2P: A self-organizing content-based locality-aware collaborative peer-to-peer network", *Computer Networks*, vol. 54, pp. 2056 - 2070, August 2010.

CONFERENCES

- D. Higuero, J. M. Tirado, F. Isaila, and J. Carretero, "Enhancing file transfer scheduling and server utilization in data distribution infrastructure" in *Proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* MASCOTS, August 2012, pages 431-438.

WORKSHOPS

- D. Higuero. F. Isaila, and J.Carretero. "Optimizing the design of elastic content distribution systems for data-intensive scientific communities". In *The 5th EuroSys Doctoral Workshop*, EuroSys 2011, April 2011, Salzburg, Austria.
- D. Higuero, J. M. Tirado, and J. Carretero. "Distributing Data to Scientific Communities". In *European Geosciences Union General Assembly 2011*, EGU 2011, April 2011, Vienna, Austria.

- J. M. Tirado, D. Higuero, and J. Carretero. "Hiddra: Filling the gap between the archive and the user". In *12th NASA-ESA Workshop on Product Data Exchange*, PDE 2010, June 2010.
- J. M. Tirado, D. Higuero, and J. Carretero. "High Performance Data Distribution for Scientific Community". In *European Geosciences Union General Assembly 2010*, EGU 2010, May 2010, Vienna, Austria.
- J. M. Tirado, D. Higuero, J. Carretero, F. Félix, and A. de la Fuente. "HIDDRA: Highly Independent Data Distribution and Retrieval Architecture for Earth Observation Missions". In *4th GRID & e- Collaboration Workshop - Digital Repositories*, Poster session, ESRIN, Frascati, Italy, February 2009. European Space Agency.

Regarding the dissemination of the research, we have established contact with the Genesi-DR project during the realization of this thesis. The Genesi-DR project (http://www.genesi-dr.eu/) from the Seventh Framework Program showed interest in the data transfer approach presented in this thesis as a solution to provide efficient data transfer between repositories resulting in a Memorandum of Understanding between both projects.

As the problem of distributing large quantities of data to different communities is common to many scenarios, the European Community has also recognized the original HIDDRA [55] project as an initiative for interoperability data management in the e-IRG Report on Data Management [45].

7.3.2 *Research internship*

The realization of this thesis provided the opportunity to perform a research internship in the *Kerdata* [1] team at INRIA Rennes - Bretagne Atlantique Center in France. The internship lasted three months from September, 2011 to November, 2011 and was supervised by Gabriel Antoniu.

The internship focused on analyzing and modeling the behavior of the BlobSeer [2] distributed data storage system in order to determine the optimal component configurations attending to the profile of the target application.

7.3.3 *Research grants*

This thesis have been supported by the following grants:

- Beca de Personal Investigador en Formación, PIF UC3M 02-0910, Ref. 494, Carlos III University, Funding: 48 months.
- Programa propio de investigación, Ayudas de Movilidad 2011, Carlos III University, Funds: 1.050 €.

Additionally, the produced publications have been partially supported by:

1 www.irisa.fr/kerdata/
2 blobseer.gforge.inria.fr

- *Input/Output techniques for distributed and high performance computing environments*, Spanish Ministry of Science and Innovation under grant TIN2010-16497.
- *TEALES: New scalable storage techniques for high-performance computing*, Spanish Ministry of Education and Science under grant TIN2007-63092.

7.3.4 *Related research*

Additionally, other work presenting ideas related with this thesis focusing on elastic web server infrastructures has been published:

JOURNALS

- J. M. Tirado, D. Higuero, J. Blas, F. Isaila, and J. Carretero, "CONDESA: A Framework for Controlling Data Distribution on Elastic Server Architectures," *Transactions on Parallel and Distributed Systems*, Under Review

CONFERENCES

- J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Reconciling dynamic system sizing and content locality through hierarchical workload forecasting". In *18th IEEE International Conference on Parallel and Distributed Systems* ICPADS 2012.
- J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Multi-model prediction for enhancing content locality in elastic server infrastructures", In *IEEE International Conference on High Performance Computing*, 2011.
- J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Predictive data grouping and placement for cloud-based elastic server infrastructures", In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* CCGrid 2011, pp. 285 - 294.

WORKSHOPS

- J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero. "Analyzing the Impact of Events in an Online Music Community". In *Workshop on Social Network Systems*, held at Eurosys, 2011.

BIBLIOGRAPHY

[1] M. Abdullah, M. Othman, H. Ibrahim, and S. Subramaniam. Optimal workload allocation model for scheduling divisible data grid applications. *Future Generation Computer Systems*, 26(7):971–978, 2010.

[2] V.K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z.L. Zhang. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Proceedings of the 2012 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 7–12. IEEE, 2012.

[3] V.K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.L. Zhang. Unreeling netflix: Understanding and improving multi-CDN movie delivery. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2012)*, pages 1620–1628. IEEE, 2012.

[4] Y.Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th international conference on World Wide Web*, pages 835–844. ACM, 2007.

[5] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771, 2002.

[6] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. Gridftp: Protocol extensions to ftp for the grid, April 2003. URL `http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf`.

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society, 2005.

[8] H. Arend and A. Christoph. Modularity and anti-modularity in networks with arbitrary degree distribution. *Biology Direct*, 5, 2010.

[9] J.M. Badia, J.L. Movilla, J.I. Climente, M. Castillo, M. Marqués, R. Mayo, E.S. Quintana-Ortí, and J. Planelles. A parallel solver for huge dense linear systems. *Computer Physics Communications*, 182(11):2441 – 2442, 2011. ISSN 0010-4655. doi: 10.1016/j.cpc.2011.06.010.

[10] M. Balman. *Data Transfer Scheduling with Advance Reservation and Provisioning*. PhD thesis, Louisiana State University, August 2010.

[11] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *Proceedings of the 20th Conference on Computer Communications (IEEE INFOCOM 2001)*, volume 2, pages 1028–1037. IEEE, 2001.

[12] P. Baptiste and R. Sadykov. Time-indexed formulations for scheduling chains on a single machine: An application to airborne radars. *European Journal of Operational Research*, 203(2), 2010.

[13] R. Barták. Data transfer optimization: Going beyond heuristics. In *Proceedings of the 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, volume 1, page 9, February 2010.

[14] M. Belshe and R. Peon. Spdy protocol draft, August 2012. URL `http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00`.

[15] P. Berkhin. A survey of clustering data mining techniques. *Grouping multidimensional data*, pages 25–71, 2006.

[16] A.R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 3–9. ACM, 2002.

[17] RS Bhuvaneswaran, Y. Katayama, and N. Takahashi. Dynamic co-allocation scheme for parallel data transfer in grid environment. In *Proceedings of the 1st International Conference on Semantics, Knowledge and Grid (SKG'05)*, pages 17–17. IEEE, 2005.

[18] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[19] M. Budiu, D. Delling, and R.F. Werneck. Dryadopt: branch-and-bound on distributed data-parallel execution engines. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1278–1289. IEEE, 2011.

[20] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys (CSUR)*, 34(2): 263–311, 2002.

[21] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling data transfer in content-centric networking. In *Proceedings of the 23rd International Teletraffic Congress*, pages 111–118. ITCP, 2011.

[22] M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[23] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2007.

[24] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on Networking (TON)*, 17(5):1357–1370, 2009.

[25] R.S. Chang and P.H. Chen. Complete and fragmented replica selection and retrieval in data grids. *Future Generation Computer Systems*, 23(4):536–546, 2007.

[26] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications*, 23(3): 187–200, 2000.

[27] A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[28] B. Cohen. The bittorrent protocol specification, June 2009. URL http://www.bittorrent.org/beps/bep_0003.html.

[29] European Commission. Work program 2008 for information and communication technologies. Technical report, European Commission, 2008. URL http://cordis.europa.eu/fp7/wp-2008_en.html.

[30] European Commission. Work program 2009 for information and communication technologies. Technical report, European Commission, 2009. URL http://cordis.europa.eu/fp7/wp-2009_en.html.

[31] European Commission. Work program 2013 - cooperation theme 3 - ict - information and communication technologies. Technical report, European Commission, July 2012. URL http://cordis.europa.eu/fp7/ict/docs/ict-wp2013-10-7-2013-with-cover-issn.pdf.

[32] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[33] Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K.W. Ross, and A. Ghose. Broadcast Yourself: Understanding YouTube Uploaders. In *Proceedings of the ACM SIGCOMM Internet measurement conference (IMC)*, pages 361–370, 2011.

[34] A. Dogan. A study on performance of dynamic file replication algorithms for real-time file access in data grids. *Future Generation Computer Systems*, 25(8): 829–839, 2009.

[35] A. Domenici, F. Donno, G. Pucciani, H. Stockinger, and K. Stockinger. Replica consistency in a data grid. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 534(1): 24–28, 2004.

[36] N. Du, B. Wang, and B. Wu. Community detection in complex networks. *Journal of Computer Science and Technology*, 23(4):672–683, 2008.

[37] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[38] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[39] M. Eleyat and L. Natvig. Mixed-precision parallel linear programming solver. In *Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 41–46. IEEE, 2010.

[40] P. Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(1):6, 2007.

[41] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[42] Facebook. Key facts, August 2012. URL http://newsroom.fb.com/.

[43] J. Feng and M. Humphrey. Eliminating replica selection-using multiple replicas to accelerate data transfer on grids. In *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 356–366. IEEE, 2004.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol–http/1.1, June 1999. URL http://tools.ietf.org/html/rfc2616.

[45] Data Management Task Force. e-IRG Report on Data Management. Technical report, European Strategy Forum on Research Infrastructures, 2009.

[46] P. Ford-Hutchinson. Rfc 4217: Securing ftp with tls, October 2005. URL http://tools.ietf.org/html/rfc4217.

[47] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[48] I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[49] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.

[50] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[51] J. Galbraith and O. Saarenmaa. Ssh file transfer protocol (sftp), July 2006. URL http://tools.ietf.org/html/draft-ietf-secsh-filexfer-13.

[52] U. Gargi, W. Lu, V.S. Mirrokni, and S. Yoon. Large-Scale Community Detection on YouTube for Topic Discovery and Exploration. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM 2011)*, 2011.

[53] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the IEEE 10th International Symposium on Workload Characterization (IISWC 2007)*, pages 171–180. IEEE, 2007.

[54] T.A. Henzinger, IST Austria, A.V. Singh, V. Singh, T. Wies, and D. Zufferey. Static scheduling in clouds. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing*, June 2011.

[55] D. Higuero, J.M. Tirado, J. Carretero, F. Félix, and A. de La Fuente. Hiddra: a highly independent data distribution and retrieval architecture for space observation missions. *Astrophysics and Space Science*, 321(3):169–175, 2009.

[56] P. Holub, H. Rudová, and M. Liška. Data transfer planning with tree placement for collaborative environments. *Constraints*, pages 1–34, 2011.

[57] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and evaluating large-scale cdns. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 15–29, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-334-1.

[58] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, 10(6):643–652, 2004.

[59] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster. The small world of file sharing. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1120 – 1134, July 2011.

[60] K. Jain, AV Vidhate, V. Wangikar, and S. Shah. Design of file size and type of access based replication algorithm for data grid. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pages 315–319. ACM, 2011.

[61] P. Jancura, D. Mavroeidis, and E. Marchiori. DEEN: A Simple and Fast Algorithm for Network Community Detection. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*, pages 150–163. Springer Berlin Heidelberg, 2012.

[62] B. Keller and G.U.I. Bayraksan. Scheduling jobs sharing multiple resources under uncertainty: A stochastic programming approach. *IIE Transactions*, 42(1), 2010.

[63] H. Kllapi, E. Sitaridi, M. Tsangaris, and Y. Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 international conference on Management of data*, pages 289–300. ACM, 2011.

[64] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, et al. Miplib 2010. *Mathematical Programming Computation*, pages 1–61, 2011.

[65] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. File-based replica management. *Future Generation Computer Systems*, 21(1):115–123, 2005.

[66] I.X.Y. Leung, P. Hui, P. Lio, and J. Crowcroft. Towards real-time community detection in large networks. *Physical Review E*, 79(6):066107, 2009.

[67] C.F. Liaw. An efficient tabu search approach for the two-machine preemptive open shop scheduling problem. *Computers & Operations Research*, 30(14):2081–2095, 2003.

[68] L. Liming, J.P. Navarro, E. Blau, J. Brechin, C. Catlett, M. Dahan, D. Diehl, R. Dooley, M. Dwyer, K. Ericson, et al. Teragrid's integrated information service. In *Proceedings of the 5th Grid Computing Environments Workshop*, page 8. ACM, 2009.

[69] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.

[70] Y. Liu and B. Plale. Survey of publish subscribe event systems. Technical report, Computer Science Dept, Indiana University, 2003.

[71] Z. Liu, P. Li, Y. Zheng, and M. Sun. Community detection by affinity propagation. Technical report, Technical Report, 2008.

[72] A. Loewenstern. Dht protocol, June 2008. URL http://www.bittorrent.org/beps/bep_0005.html.

[73] A. Makhorin. Modeling language gnu mathprog, December 2008. URL http://www.cs.unb.ca/~bremner/docs/glpk/gmpl.pdf.

[74] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[75] B. Meindl and M. Templ. Analysis of commercial and free and open source solvers for linear optimization problems. Technical report, Technische Universität Wien, March 2012.

[76] A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[77] A. Mislove, B. Viswanath, K.P. Gummadi, and P. Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 251–260. ACM, 2010.

[78] G. Mühl. *Large-Scale Content-Based Publish-Subscribe Systems*. PhD thesis, TU Darmstadt, November 2002. URL http://tuprints.ulb.tu-darmstadt.de/274/.

[79] M.E.J. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69:066133, Jun 2004. doi: 10.1103/PhysRevE.69.066133.

[80] M.E.J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[81] M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[82] T.V. Nguyen. *Content distribution networks over shared infrastructure: a paradigm for future content network deployment*. PhD thesis, School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, 2005.

[83] E. Nygren, R.K. Sitaraman, and J. Sun. The akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[84] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 303–314. ACM, 1998.

[85] S. Pandey and R. Buyya. Scheduling data intensive workflow applications based on multi-source parallel data retrieval in distributed computing networks. Technical report, The University of Melbourne, September 2010.

[86] M. Pathan and R. Buyya. A taxonomy of CDNs. *Content delivery networks*, pages 33–77, 2008.

[87] P.R. Pietzuch. *Hermes: A scalable event-based middleware*. PhD thesis, University of Cambridge, 2004.

[88] M.L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2012.

[89] P. Pons and M. Latapy. Computing communities in large networks using random walks. *Computer and Information Sciences-ISCIS 2005*, pages 284–293, 2005.

[90] J. Postel and J. Reynolds. File transfer protocol, October 1985. URL http://tools.ietf.org/html/rfc959.

[91] U.N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[92] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona: a high performance publish-subscribe system for the world wide web. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[93] J.M. Ramírez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J.L. González-García, and A. Hirales-Carbajal. Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. *Journal of Grid Computing*, 9(1), 2011.

[94] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high-performance data grid. *Grid Computing—GRID 2001*, pages 75–86, 2001.

[95] E. Rescorla. Rfc 2818: Http over tls, May 2000. URL http://tools.ietf.org/html/rfc2818.

[96] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*, volume 35. Elsevier Science, 2006.

[97] M. Rosvall, D. Axelsson, and C.T. Bergstrom. The map equation. *The European Physical Journal-Special Topics*, 178(1):13–23, 2009.

[98] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[99] T. Sawik. *Scheduling in Supply Chains Using Mixed Integer Programming*. Wiley Online Library, 2011.

[100] M. Smelyanskiy, V. W. Lee, D. Kim, A. D. Nguyen, and P. Dubey. Scaling performance of interior-point method on large-scale chip multiprocessor system. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, volume 22. ACM, 2007. ISBN 978-1-59593-764-3.

[101] D. Soumitra and I. Mia. Global information technology report 2008-2009. In *The World Economic Forum, Geneva, Switzerland*, 2009.

[102] R. Stewart and C. Metz. Sctp: new transport protocol for tcp/ip. *Internet Computing, IEEE*, 5(6):64 –69, nov/dec 2001. ISSN 1089-7801. doi: 10.1109/4236.968833.

[103] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[104] J. Sun, Z. Jiang, S. Gao, and W. Yang. A new optimal replica placement strategy in content distribution networks. In *Proceedings of the 2010 International Conference on Intelligent Computing and Integrated Systems (ICISS)*, pages 351–354. IEEE, 2010.

[105] D. Tam, R. Azimi, and H.A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. *Databases, Information Systems, and Peer-to-Peer Computing*, pages 138–152, 2004.

[106] W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A.P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM, 2003.

[107] J. Thompson, S. Bergqvist, M. McKeay, M. Sintorn, M. Smith, P. Kersch, R. Möller, and M. Levy. The state of the internet 4th quater 2012. Technical report, Akamai, 2012.

[108] J. M. Tirado, D. Higuero, F. Isaila, J. Carretero, and A. Iamnitchi. Affinity P2P: A self-organizing content-based locality-aware collaborative peer-to-peer network. *Computer Networks*, 54(12):2056 – 2070, 2010. ISSN 1389-1286. doi: 10.1016/j.comnet.2010.04.016.

[109] J.M. Tirado, D. Higuero, F. Isaila, and J. Carretero. Predictive data grouping and placement for cloud-based elastic server infrastructures. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 285–294. IEEE, 2011.

[110] J.M. Tirado, D. Higuero, F. Isaila, and J. Carretero. Reconciling dynamic system sizing and content locality through hierarchical workload forecasting. In *Proceedings of the 18th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2012.

[111] R. Torres, A. Finamore, J.R. Kim, M. Mellia, M.M. Munafo, and S. Rao. Dissecting video server selection strategies in the youtube cdn. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS)*, pages 248–257. IEEE, 2011.

[112] J. Touch, J. Heidemann, and K. Obraczka. Analysis of http performance. Technical Report ISI/RR-98-463, USC/Information Sciences Institute, 1998.

[113] P. Triantafillou and I. Aekaterinidis. Content-based publish-subscribe over structured p2p networks. In IET, editor, *Proceedings of the third international workshop on distributed event-based systems (DEBS)*, pages 104–109, 2004.

[114] M. Tu, P. Li, I.L. Yen, B.M. Thuraisingham, and L. Khan. Secure data objects replication in data grid. *IEEE Transactions on Dependable and Secure Computing*, 7(1):50–64, 2010.

[115] R. Tudoran, A. Costan, and G. Antoniu. Mapiterativereduce: a framework for reduction-intensive data processing on azure clouds. In *Proceedings of 3rd international workshop on MapReduce and its Applications Date*. ACM, 2012.

[116] G. Urdaneta, G. Pierre, and M. Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845, 2009.

[117] A. Vakali and G. Pallis. Content delivery networks: Status and trends. *Internet Computing, IEEE*, 7(6):68–74, 2003.

[118] S. Vazhkudai. Enabling the co-allocation of grid data transfers. In *Proceedings of the 4th International Workshop on Grid Computing*, GRID '03, pages 44–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2026-X.

[119] V.V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2004.

[120] S. Voulgaris, E. Rivière, A.M. Kermarrec, and M. Van Steen. Sub-2-Sub: Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale Collaborative Networks. Rapport de recherche RR-5772, INRIA, 2005.

[121] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Optimizing latency and throughput of application workflows on clusters. *Parallel Computing*, 37(10):694–712, 2011.

[122] D.J. Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 2003.

[123] J. Wulf, R. Zarnekow, T. Hau, and W. Brenner. Carrier activities in the CDN market-An exploratory analysis and strategic implications. In *Proceedings of the 2010 14th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 1–6. IEEE, 2010.

[124] R. Xu, D. Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[125] C.T. Yang, M.F. Yang, and W.C. Chiang. Enhancement of anticipative recursively adjusting mechanism for redundant parallel file transfer in data grids. *Journal of Network and Computer Applications*, 32(4):834–845, 2009.

[126] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 927–936. ACM, 2009.

[127] H. Yin, X. Liu, F. Qiu, N. Xia, C. Lin, H. Zhang, V. Sekar, and G. Min. Inside the bird's nest: measurements of large-scale live vod from the 2008 olympics. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 442–455. ACM, 2009.

[128] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, 26(8):1200 – 1214, 2010. ISSN 0167-739X. doi: 10.1016/j.future.2010.02.004.

[129] M. Zerola, M. Sumbera, R. Barták, and J. Lauret. Using constraint programming to plan efficient data movement on the grid. In *Proceedings of the 21st International Conference on Tools with Artificial Intelligence (ICTAI'09)*, pages 729–733. IEEE, 2009.

[130] M. Zerola, J. Lauret, R. Barták, and M.Š umbera. Building efficient data planner for peta-scale science. In *Proceedings of the 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, volume 1, page 25, February 2010.

[131] M. Zerola, J. Lauret, R. Barták, and M.Š umbera. Efficient multi-site data movement using constraint programming for data hungry science. In *Journal of Physics: Conference Series*, volume 219, page 062069. IOP Publishing, 2010.

[132] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[133] P. Zhao and C.Q. Zhang. A new clustering method and its application in social networks. *Pattern Recognition Letters*, 32(15):2109 – 2118, 2011. ISSN 0167-8655. doi: 10.1016/j.patrec.2011.06.008.

[134] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM, 2001.