Quality of Availability for Widely Distributed and Replicated Content Stores

Dissertationsschrift in englischer Sprache
genehmigt vom Fachbereich Informatik
der Technischen Universität Darmstadt von

Diplom-Informatiker Giwon On

geboren am 22.1.1965

zur Erlangung des Grades eines
Doktor-Ingenieur (Dr.-Ing.)

Darmstadt 2004
Hochschulkennziffer D17

Vorsitz:	Prof. Dr. Jose Luis Encarnacao
Referent:	Prof. Dr. Ralf Steinmetz
Korreferent:	Prof. Nicolas D. Georganas (Ph.D.)

Tag der Einreichung:	06. April 2004
Tag der Disputation:	01. Juni 2004

Abstract

As the Internet continues to experience a tremendous expansion in number of users and services today, the importance of satisfying service availability becomes recently one of the most critical factors for the success of “Internet services” such as the World Wide Web, peer-to-peer file sharing and digital audio/video streaming.

In this thesis, we take an availability-centric view on end-to-end service Quality of Service (QoS) and investigate service availability for large-scale content distribution and replication systems in wide-area internetworks such as the Internet. The main focus is on issues of providing availability guarantees, i.e., delivering target level of service availability with guarantees for all individual users of services running on top of the above systems. Specifically, the thesis aims at the development of concept and framework for technically realizing the availability-centred QoS approach, on the one hand, and of decision strategies for content replication in order to provide availability guarantee for the large-scale Internet-based services, on the other hand. In particular, we tackle the replica selection and placement problem and study the effectiveness of realistic replication strategies on the achieved service availability in the Internet.

There are several contributions that this thesis makes. The first contribution is the concept of Quality of Availability (QoA) which enables one to treat availability as a controllable and observable QoS parameter. On the basis of this concept, three refinements of the existing availability definition have been investigated (decoupled, differentiated and fine-grained) in order to enable a more quantitative specification and evaluation of the service availability. For a technical realization of the QoA concept, we have developed a QoA framework in which users can specify their service requirements in terms of QoA and the requirements are mapped into the low-level replication specification. Additionally, the QoA framework offers QoA metrics and mechanisms to control the QoA guarantee.

The second contribution is an evaluation of the replication techniques used to resolve the replica placement problem for different content distribution and replication system models. We have tackled the replica placement problem and investigated specifically the effects of the degree of replication, the granularity and location of replicas on overall service availability achieved by the selected placements. We divided the placement problem into two sub-problems, (1) the static full server replica placement in content delivery network (CDN)-like replicated systems and (2) the dynamic partial data replica placement in peer-to-peer (P2P) systems. For solving the static full server replica placement problem, we first modelled the content distribution and replication system as a stochastic graph and then developed several ranking-based heuristics and an exact state enumeration algorithm for improving and guaranteeing service availability of the system. By simulating the behaviour of the proposed algorithms, we found that the location of replicas is a relevant factor for the service availability. Even though the QoA improvement could be achieved by increasing replica numbers, replicas’ placement and their dependability affected the QoA more significantly.

In the case of the dynamic replica placement problem in P2P-based content distribution and replication systems, we took the intermittent connectivity of peers of the system explicitly into account. We modelled the system as a dynamic stochastic graph where each node (peer) goes up and down according to its assigned up-time probability. We considered different time scales for replica creation so that replicas could be created either proactively at the service initialization phase, or on-the-fly during the service running phase. We re-used the placement heuristics used for solving the static replica placement problem and modified them slightly so that they are fully decentralized, assume no global information about the system condition or network topology, and work in a cooperative way to replicate content on-the-fly. To quantitatively study the effectiveness of the used heuristics, we developed an event-driven simulation framework which captures the data access model as well as peers' dynamic behaviour. The simulation results indicated that the cooperative placement heuristics offer, in general, better QoA than non-cooperative local placement approach, while all of them induce approximately the same amount of replacement cost in terms of the number of replacements.

The third contribution is a new approach for determining replica placements in large-scale content distribution and replication systems, which always provides an availability guarantee for all service requesting entities in a content distribution and replication system. For the first time, we suggested an admission control-based replica placement scheme that solves the static replica placement problem in a polynomial time, with respect to the size of the input system. Through a simulative study, we have experimentally evaluated the admission control-based algorithm with random and power-law graphs of different sizes. Our results show that the placement achieved by the admission control-based algorithm always provides a QoA guarantee, and that the upper bound of the replication degree (i.e., replica numbers) required for delivering QoA guarantees is about 27% and 37% of the total nodes in the random and power-law graph, respectively. As far as we know this is the first time that the admission control concept has been integrated with a replica placement scheme that solves the problem in a polynomial time while offering service availability guarantees.

Zusammenfassung

Mit dem rasanten Wachstum des Internet, hinsichtlich der Anzahl von Diensten und Anwendern, dehnen sich “Internet-Dienste” wie World Wide Web, Peer-to-Peer Datenaustausch, digitale Audio/Video-Übertragung großflächig aus. Eine der wichtigen Anforderungen, die die Benutzer dieser Internet-Dienste häufig erwarten, liegt darin, dass sich die Dienste auf einer akzeptierbaren Qualitätsebene bewegen sollen. Die Qualität eines Dienstes kann anhand des Durchsatzes, der Verfügbarkeit, der Sicherheit, usw. spezifiziert werden.

Dienstgüte (Quality of Service, QoS) ist der Schlüsselansatz dafür, Ressourcenzusicherung und Dienstgarantien für viele vernetzte Multimedia- und andere Dienste sicherzustellen. Bisher jedoch konzentrierte sich die Aufmerksamkeit im QoS-Bereich hauptsächlich auf die Leistung, wie Leistungsoptimierung und -garantien. Während die QoS-Unterstützung bezüglich der Übertragungscharakteristiken für viele Internet-Dienste (insbesondere für die Soft-Echtzeit-Dienste) weiterhin wichtig ist, nimmt eine völlig andersartige Benutzeranforderung an Bedeutung zu: “wie verfügbar” ein bestimmter Dienst und dessen Daten sind. Im Rahmen von QoS ist das Thema der Verfügbarkeit noch nicht so detailliert untersucht worden.

In der vorliegenden Arbeit wird die Darstellung über QoS auf die Verfügbarkeit zentriert. Dabei betrachtet die Arbeit den Problemkreis der Ende-zu-Ende-Dienstverfügbarkeit für die Internet-Dienste, insbesondere für weitverteilte und replizierte Content-Stores. Die Arbeit stellt zuerst ein Konzept (genannt Quality of Availability, QoA) und ein dementsprechendes Bezugssystem vor, die zusammen die Verfügbarkeit als einen neuen, steuerbaren und beobachtbaren QoS-Parameter betrachten und somit die Spezifizierung und die Evaluierung der Verfügbarkeitsanforderungen in unterschiedlichen Dienstklassen unterstützen. Dieses Bezugssystem umfasst ein Modell für Spezifikation und Verwaltung von Ressourcen, das hinsichtlich der Dienstverfügbarkeit die Zuordnung der von den Endnutzern definierten Dienstanforderungen und die Mechanismen zur garantierten Verfügbarkeit von Ressourcen-Allokationen ermöglicht.

Auf der Basis des QoA-Konzepts und der dabei verfeinerten Notationen von Verfügbarkeit wurde speziell das Problem der Replikaplatzierung in weitverteilten und replizierten Content-Stores detailliert betrachtet. Der Fokus liegt dabei auf den Replikationsmechanismen und -strategien, die sowohl die Verbesserung der Verfügbarkeitsgüten als auch die Bereitstellung der Verfügbarkeitsgarantien für alle individuellen Benutzer der Dienste unterstützen und ermöglichen. Es wurde untersucht, wie die Anzahl, Granularität und Lokation der Replikate die Dienstverfügbarkeit beeinflussen, die durch ausgewählte Replikaplatzierungen erreicht werden kann. Abhängig von der Replikaerzeugungszeit und der Granularität der zu replizierenden Contents wurde das Problem der Replikaplatzierung in zwei Problemdomänen geteilt: die statische Platzierung von Content-Server-Replikaten in Content-Delivery-Network (CDN) Systemen und die dynamische Platzierung von Daten-Replikaten in Peer-to-Peer (P2P) Systemen.

Zur Lösung des statischen Platzierungsproblems der Serverreplikate wurden die Content-Stores als ein parametrisierter, stochastischer Graph modelliert. Als Platzierungsalgorithmen wurden eine Reihe von Heuristiken und eine exakte, zustandaufzählungs-basierte Methode entwickelt, die sich jeweils für die Verbesserung und die Gewährleistung der Dienstverfügbarkeit der Systeme eignen. Durch eine Simulationsstudie und eine quantitative Evaluierung der erreichten Replikationsgüten wurde gezeigt, dass die Lokation von Replikaten und deren Verlässlichkeit ein relevanter Faktor für die Verfügbarkeitgüte ist. Weiterhin wurde gezeigt, dass die Heuristiken im Gegensatz zur exakten Methode im Allgemeinen keine Garantie hinsichtlich der Dienstverfügbarkeit abliefern.

Im Fall des Problems der dynamischen Replikaplatzierung in P2P-basierten Content-Stores wurde die zeitweilige Konnektivität von Peers ausführlich berücksichtigt. Die P2P-Systeme wurden dann als ein dynamischer, stochastischer Graph modelliert, wobei jeder Knoten und jede Kante gemäß der zugeordneten Ausfallswahrscheinlichkeit ausfallen bzw. erreichbar werden. Für die Erzeugung von Replikaten wurden zwei unterschiedliche Zeitphasen eingefügt, so dass Replikate entweder proaktiv bei der Initialisierung oder ‘on-the-fly’ während der Laufzeit der Dienste erzeugt werden können. Die zur Lösung dieses dynamischen Platzierungsproblems entwickelten Heuristiken sind rangordnungs-basiert und unterscheiden sich grundsätzlich dadurch, ob sie bei der Entscheidung der Platzierung miteinander kooperativ arbeiten oder nicht. Zur quantitativen Evaluierung wurde ein ereignisbasiertes Simulationssystem entwickelt, das neben einer Einstellungsmöglichkeit replikations-relevanter Parameter die Verwendung unterschiedlicher Netzwerktopologien ermöglicht. Das Simulationsergebnis beleuchtet die Wichtigkeit der kooperativen Platzierung in P2P-Systemen: die kooperativen Heuristiken haben hinsichtlich der Dienstverfügbarkeit eine höhere Güte als das typische nicht-kooperative Platzierungsverfahren, während die durchschnittlichen Replikationskosten für alle benutzten Platzierungsverfahren annähernd gleich waren.

Als ein weiteres Problem der Replikaplatzierung wurde ein Lokationsproblem untersucht, bei welchem die Ressourcen beschränkt sind. Dabei soll die als Lösung ausgewählte Platzierung von Replikaten die angeforderte Verfügbarkeitgüte für alle Benutzer immer gewährleisten. Der entwickelte Lösungsansatz integriert das Konzept der Zugangskontrolle (Admission Control) und die Replikaplatzierung derart, dass das Problem in polynomieller Laufzeit gelöst werden kann und Garantien bezüglich der Dienstverfügbarkeit gemacht werden können. Im Rahmen einer Simulationsstudie mit unterschiedlichen Netzwerktopologien und Graphengrößen wurde der Algorithmus bezüglich der Effizienz sowie der Umsetzbarkeit experimentell evaluiert. Die Arbeit leistet somit einen wichtigen Beitrag, um Garantien bezüglich der Verfügbarkeit von Internet-Diensten und deren Daten in weitverteilten Netzen wie z.B. dem Internet zu ermöglichen.

Table of Contents

List of Figures

List of Tables

Alphabetical List of Abbreviations

AC	Admission Control
AS	Autonomous System
API	Application Programming Interface
AFS	Andrew File System
CDRS	Content Distribution and Replication System
COM	Combined Highly Available First and TransitNode Heuristic
CORBA	Common Object Request Broker Architecture
CDN	Content Delivery Network
CPU	Central Processing Unit
DiffServ	Differentiated Services
DNS	Domain Name System
DRP	Dynamic Replica Placement
FTP	File Transfer Protocol
GUI	Graphical User Interface
HA	Highly Available First Heuristic
HAP	Highest Available Path
IETF	Internet Engineering Task Force
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IntServ	Integrated Services
IP	Internet Protocol
ISP	Internet Service Provider
ITU-T	International Telecommunications Union - Telecommunications Standardization Sector
LAN	Local Area Network
LC-FTP	Loss Collection File Transport Protocol
LC-RTP	Loss Collection Real-time Transport Protocol
LFU	Least Frequently Used

LRU	Least Recently Used
MAN	Metropolitan Area Network
MFU	Most Frequently Used
MPEG	Motion Picture Experts Group
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
NoD	News-on-Demand
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
PQ	Priority Queue
QoA	Quality of Availability
QoS	Quality of Service
RAID	Redundant Array of Inexpensive Disks
RPC	Remote Procedure Call
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
SRP	Static Replica Placement
SRP-G	Static Replica Placement for Guaranteeing Availability
SRP-M	Static Replica Placement for Increasing Availability
TCP	Transmission Control Protocol
TR	TransitNode Heuristic
UDP	User Datagram Protocol
UP	Highly Up First Heuristic
VoD	Video-on-Demand
VPN	Virtual Private Network
WAN	Wide Area Network
WWW	World Wide Web

Trademarks

Linux is a trademark of Linus Torvalds.

SuSE is a registered trademark of SuSE Linux AG.

Redhat is a registered trademark of RedHat.

Solaris is a trademark of Sun Microsystems, Inc.

Microsoft, Windows and Windows 2000 Active Directory are trademarks or registered trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

IBM's DB2 is a registered trademark of International Business Machines Corporation.

AFS is a registered trademark used by the Transarc Corporation to identify the commercial packaging of the Andrew File System.

Other company, product and service names may be trademarks or service marks of others.

Chapter 1 - Introduction

1.1 Motivation

Today, the Internet continues to experience a tremendous expansion in number of users and services. Conventional “Internet services” such as remote login, file transfer and electronic mail continue to be widely used. Additionally, “Internet services” such as the World Wide Web, peer-to-peer file sharing, digital audio/video streaming and on-line transactions (e.g., banking, shopping) expand in popularity. The latter services can be characterized depending on whether they are time-dependent (also known as ‘soft real-time’) and whether their core contribution enables high degrees of data sharing between participating users.

One of the important demands that users of these services commonly expect is that the services should deliver acceptable levels of quality. The service quality can be specified, for example, in terms of performance, availability, security, etc. This users demand further leads to the requirement for the underlying (often distributed and replicated) systems that they should be able to manage their resources (e.g., storage, access link bandwidth) in order to provide “service guarantees” to the users.

Quality of Service (QoS) is a key approach to provide resource assurance and service guarantees for many networked (multimedia) services [Sch01a, Wol96, Wan01]. The idea of QoS is particularly well established in the transmission domain of data networking [HLP91, DBB+93, AGH94, NS95, SW97]. The main research goal of QoS in networked systems is to improve the service quality by addressing the issues of resource management [ZDE+93, Wol96, OLKC97, OLKC98, GRH02, CGS03], service differentiation [NS96, ACH98, BCS94, BBD+98, LALT02, DDTT03] and performance optimization [BGM+97, DG00, Sch01a, KSC02, GJMT03].

There have been many significant research results, technology advances and solutions in QoS in the last 20 years. However, their application to commercial products in a wide scale has not been as successful by comparison to their attention in the research arena. One critical reason may be that much of the attention of QoS has been (and is) focused on the performance issues, e.g., performance optimization and guarantee, raised by the particular soft real-time services such as video streaming and tele-conferencing. Thus, the QoS metrics are typically performance-oriented and capture mainly time-dependent parameters of data transmission in the network layer. For instance, the typical QoS parameters are throughput, delay (bound) and packet loss rate [Wan01, Wol96, SN04].

While for the soft real-time services the control of the transmission characteristics is certainly important, it may not be the most pressing need with regard to QoS requirements of the application users [Hen99, Sch01b, SEQ01]. As an example, we consider a high performance video-on-demand server that can stream simultaneously to thousands of clients with a quality of 100ms maximum delay. However, the QoS guarantees of this server are not very useful as guarantees, if the server cannot be made *available* under changing conditions of the underlying system and of its associated service environment. Indeed, there are other quite different user

requirements: the user's expected QoS includes also "how available" a certain service and its data are.

In the context of QoS, the availability issue has not yet been studied in great depth. In this thesis, we take an availability-centric view on QoS and consider the issue of the end-to-end service availability for the Internet services, specifically for widely distributed and replicated content services.

Availability is often defined as a ratio of successful queries to the total number of queries (see [CDK01 and TvS02] for an overview). Based on this availability definition, we define *service availability* as follows:

> *A service is said to be available if it functions as expected,*
> *and its data are reachable during the service access time.*

There are several approaches for increasing availability of distributed systems. One approach is improving the reliability of the system so that the system can run error-free during the total service time. Many researchers have turned their attention to architectures that improve overall availability, for example in the context of distributed file servers [SKK+90, Cam98, LN00], distributed database servers [Cor98, IBM99], clustered web servers [FGC+97], clustered email servers [SBL99] and software RAID systems [TDMV96]. All these studies mainly focus on the availability of server systems. However, given the pervasive use of the Internet to access remote services, even 100% server availability will most certainly neither deliver high availability to end users nor provide any availability guarantee. For example, [DCGN03] shows that network failures prevent client access to a service between 1.5 ~ 2% of the total service time.

Replication is a widely-known, well-proven technique for increasing availability of Internet services in the face of server failures or network partitions [WPS+00, DMP+02, GDN+03]. For example, a popular web site can be either completely or partially replicated from an original web server to multiple servers, which are often geographically spread across the Internet [Rab98, Dav01, PvS01]. The web contents are still available to users even in the presence of any failure of the original server or connection link to that server. However, replication is not a panacea [GHOS96, RS02]. Important decision issues to be resolved are the target location and granularity of replicas, the degree of replication, the placement of replicas, and the request redirection mechanism required to achieve a target level of service availability.

There have been many efforts to resolve these replication-related problems. However, they either applied to small centralized networks, e.g., Intranet [TA92, GRR+98, RRP99, KM02], or focused mainly on the request redirection problem [BAZ+97, LCC+02, BCG+02, Kan02, CS02]. Thus, there is comparatively little understanding of the impact on service availability of different schemes for selecting and placing replicas. Furthermore, the main goals of all those existing replica selection and placement schemes are either increasing availability or reducing replication cost, but not achieving a target level of service availability with guarantees.

1.2 Goals and Methods

In this thesis, we take an availability-centric view on end-to-end service QoS and investigate service availability for large-scale content distribution and replication systems in wide-area internetworks such as the Internet. The main focus is on issues of providing availability guar-

antees, i.e., delivering target level of service availability with guarantees for all individual users of services running on top of the above systems. Specifically, the thesis aims at the development of concept and framework for technically realizing the availability-centred QoS approach, on the one hand, and of decision strategies for content replication in order to provide availability guarantee for the large-scale Internet-based services, on the other hand. In particular, we tackle the replica selection and placement problem and study the effectiveness of realistic replication strategies on the achieved service availability in the Internet.

One goal of this thesis is to identify the fundamental problems in specifying and evaluating quantitatively different levels of service availability for Internet-based services. A further goal is to develop concept and framework which enable one to resolve these problems, in the context of the QoS.

An additional goal is to enable Internet-based services to tune their systems and further service availability as their workload changes and as network reliability and network condition change. For this purpose, we study the effects of the degree replication, the granularity and location of replicas, and the reliability of the underlying system and network on overall service availability. In particular, we quantify the 'goodness' of achieved replica placements between two extremes of replica granularity (full and partial) and replication time scale (static and dynamic) with different network size and topology.

Besides the extensive placement studies, a further goal of this thesis is to develop fast and efficient placement algorithms which provide replica placements that always guarantee the service availability for all service requesting entities in a large-scale Internet-based services.

With regard to the methods employed to achieve these goals, we will utilize simulative experiments. The simulation method enables us to model the details of individual characteristics of the Internet-based service systems in a reasonable depth whereas analytical approaches are often intractable for explicitly capturing such aspects. We will build a module-based scalable simulation framework which is capable of experimenting with different replication strategies, several networks of different size and topology, service and data access models, failure models and replica granularity. Additional advantages of such simulative experiments are the usefulness and efficiency for rapidly collecting statistics. The statistics are used to evaluate, compare and quantify the effectiveness of, the used replication strategies on the achieved availability.

1.3 Outline

Chapter 2 presents an overview of techniques for content distribution and replication system, including how these technologies have evolved on the Internet in the past few years. In order to provide an insight into content distribution and replication systems, the most relevant content distribution architectures are presented, and major replication issues such as the granularity and location of replicas, as well as the time scale are discussed.

In Chapter 3, the concept of *Quality of Availability* (QoA) is proposed; it assumes availability to be a controllable, observable QoS parameter. It includes some refinements of availability definition, which are necessary for specifying differentially and evaluating quantitatively the availability requirements and the achieved availability, respectively. For a technical implementation of the QoA concept, a QoA framework is developed and proposed.

Chapter 4 tackles the replica placement problem and studies the effects of number and location of replicas with respect to the achieved service availability. In the problem definition, it is supposed that a complete copy of the original contents is treated as a replica (full replication model), and that the placement of replicas is determined prior to starting service. The problem is divided into two main issues: finding a 'good' placement for a fixed number of replicas and determining the number and location of replicas for providing availability guarantees for all service requesting entities. Several placement algorithms are proposed and the 'goodness' of their placements are evaluated through simulative experiments.

Chapter 5 examines dynamic data replica placement from the point of view of peer-to-peer file sharing systems. Specifically, the chapter investigates dynamic data replication, where the goal is to choose dynamically the number and location of replicas for a particular content, while taking intermittent connectivity of peers explicitly into account. In this chapter, the replica unit is either any particular content or a subset of the contents. Furthermore, the chapter considers different times of replica creation, i.e., proactive at the service initialization phase and on-the-fly during the service running time. The placement algorithms proposed in Chapter 4 are slightly modified such that they work in a fully decentralized, cooperative way to replicate content on-the-fly. A set of simulation results that are collected through an event-driven simulation study show the efficiency of the used placement algorithms under different simulation parameter settings.

Chapter 6 describes an admission control approach as a means for solving the replica placement problem, where the goal is providing a service availability guarantee for all service requesting entities in a capacitated content distribution and replication system. On the base of this approach, a new placement algorithm is developed, of which the computational complexity grows polynomially with respect to the size of the input system. The chapter shows its feasibility and practicality through a simulative study.

Chapter 7 concludes our major findings and contributions. Furthermore, an outlook to future research aspects is given.

Chapter 2 - Techniques for Content Distribution and Replication

This chapter presents an overview of techniques for content distribution and replication, which are applied to modern large-scale Internet-based distributed applications, in order to provide better service availability or performance. The main focus of this chapter, as well as this thesis, is on models and strategies for content distribution and replication. The chapter starts with defining terminology used throughout this thesis. It then gives an overview of content distribution approaches, ranging from the centralized client-server to the fully decentralized peer-to-peer model. Finally, major replication issues such as the granularity and the location of replicas, as well as the time scale, i.e., when to replicate are discussed.

2.1 Terminology

In order to ensure a well-defined use of terminology throughout the thesis, we start with defining terms and terminology related to our work of content distribution and replication throughout this thesis. According to [MT04] we will use the term *content* to denote any kind of audiovisual, visual, sound, textual data, or software package in any type and format such as MP3 music file, MPEG videos, ASCII text, etc. We use the term *content store* to denote a repository of contents where contents are stored and managed. We will use the term *content provider* to denote an entity (an organization or an individual) who provides content for others to access, while the term *users* will be used to denote the individuals who access this content.

We will use the term *host* to denote a computer which is available to users, when accessing the content. A host which permanently provides services is denoted as a *server* whereas the host requesting the service is denoted as a *client*. However, in a decentalized peer-to-peer (P2P) system in which any peer in the system can be a client and/or a server, every host is a *peer*. Furthermore, all users are also content providers.

A content store that can provide a single logical view to users may be physically distributed, i.e., split into smaller parts and subsequently placed across multiple interconnected machines either in a local-area network (LAN) or a wide-area network (WAN). Conventionally, the content (usually denoted as *data*) could be distributed and shared by means of a distributed shared memory, a distributed (shared) database, or a distributed file system.

In comparison to these forms of sharing, contents on the Internet are often widely shared by means of the Web: users who want to provide contents (usually web documents) upload them to a web site or multiple web sites so that other users who want to retrieve the contents download them from the web site. The content can be accessed/downloaded multiple times from several users, according to the factor of its importance or popularity. As a result, multiple copies of a content may exist on the Internet, being spread geographically.

In this thesis, an identical copy (i.e., error-free) of a content is called a *replica*. A replica can be created, accessed for reading or updating, and deleted within a content store, upon access request from content providers or users. A system which manages the distribution and replication of a content store is called a *content distribution and replication system* (CDRS). A general architecture of a CDRS is shown in Figure 1. Components of the CDRS and the relationships among them are described as follows:

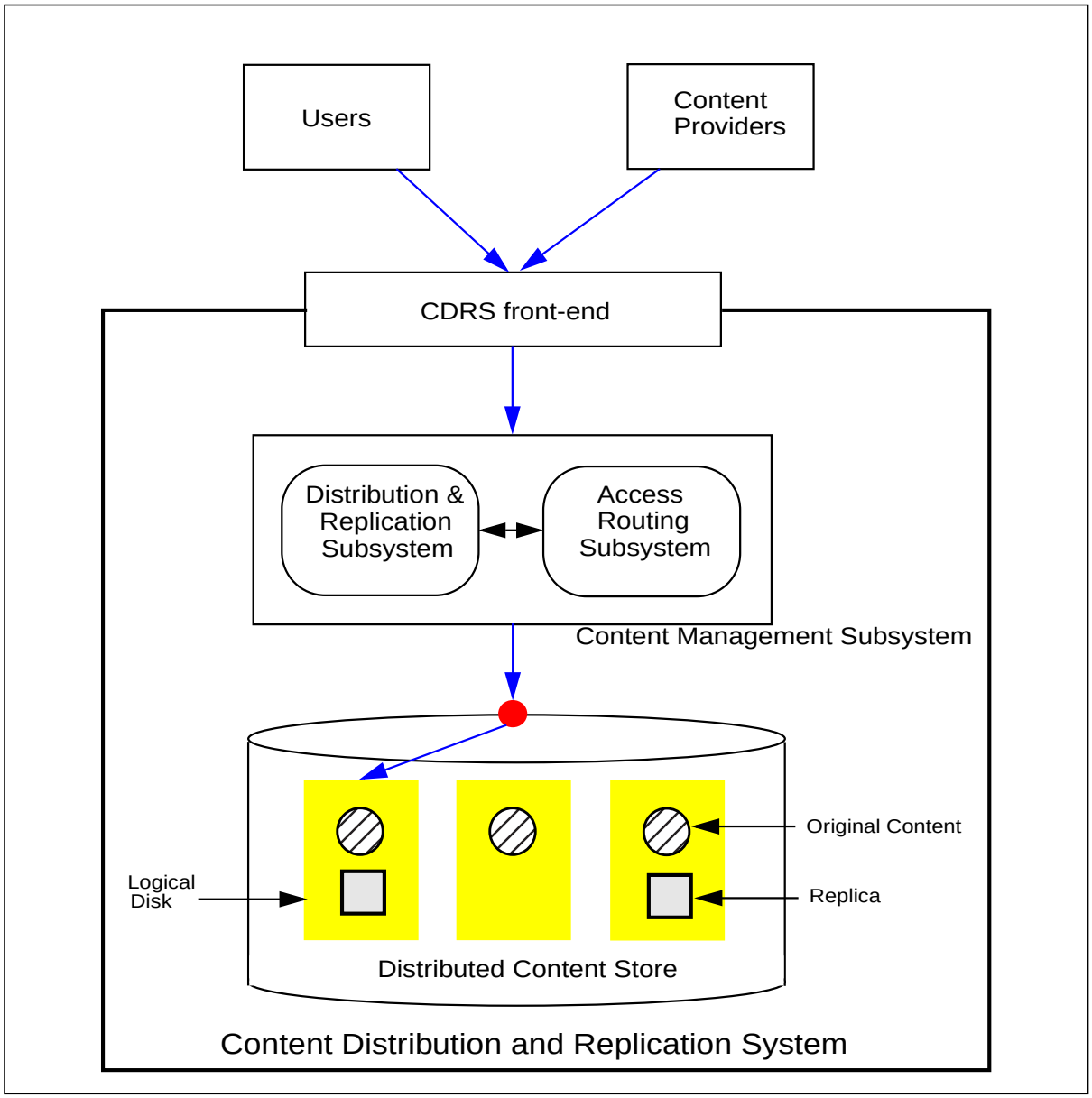


Figure 1: A general architecture of CDRS.

The content provider publishes content that is to be distributed and replicated into the CDRS. The distribution and replication subsystem moves the content to the distributed content store. Additionally, it interacts with the access routing subsystem through feedback to assist in the replica server (logical disk in Figure 1) selection process for users request. A user requests contents from what it perceives to be the origin. The access request is actually directed to a

suitable replica server in the distributed content store by the access routing subsystem. The requested content is delivered to the user by the selected replica server.

In this thesis, we concentrate on techniques for efficient distribution and replication of content to increase and guarantee service availability of such a content store, in the context of the Internet.

2.2 Content Distribution and Replication Approaches

There are many ways in which contents of a content store can be physically distributed and replicated across multiple hosts. The most generic distinction can be made based on the distribution paradigm, roughly between two types: *client-server* and *peer-to-peer* paradigm. In the client-server paradigm, clients and servers are different. Typically, servers are responsible for storing and distributing contents so that the contents are permanently available to the clients, while the clients access the contents to read or write. However, the peer-to-peer paradigm assumes no dedicated servers which host clients: therefore, every peer (host) can be considered both a client and a server.

In this section, we give an overview of commonly used architectures for content distribution and replication. We then discuss their core features and limitations in supporting replication for increasing service availability.

2.2.1 Client-Server Model

The client-server architecture is often cited when distributed systems are discussed and most widely employed for building distributed systems [TvS02, CDK01]. In this architecture, the roles of server and client are distinct. For example, in the context of content services, servers are typically responsible for storing and distributing contents so that the contents are available to the clients for access.

In a distributed content store that is built based on this conventional client-server model the content could be distributed and shared by means of a distributed shared memory [IS99, CDK01], a distributed database [Cor98, IBM99], or a distributed file system [SKK+90, Cam98, LN00].

To increase service availability of the system based on this model, replication of servers or contents is often deployed. There are two ways in which replication can take place. The first form is using multiple servers which interact as necessary to provide a content service to a client. Figure 2 shows this approach. The CDRS consists of several separated servers residing in a LAN. The content servers may partition the set of contents on which the service is based and distribute them between themselves. Moreover, they may maintain replicas of the contents on a subset of the contents. The content and replica manager that also acts as the front-end of the CDRS may keep metadata about which parts of the original content as well as their replicas are located on which content server. Based on this information the manager routes the incoming request to a suitable content server.

The downside of this approach is that the CDRS can become unavailable, due to a network partition between this LAN and the Internet.

The second form of the client-server model based CDRS, which is a widely used solution for content replication in the Web is making an entire copy of the contents available at a differ-

ent server residing on another LAN. This approach is called *mirroring* (or *mirror* for short) [TvS02, JJK+01].

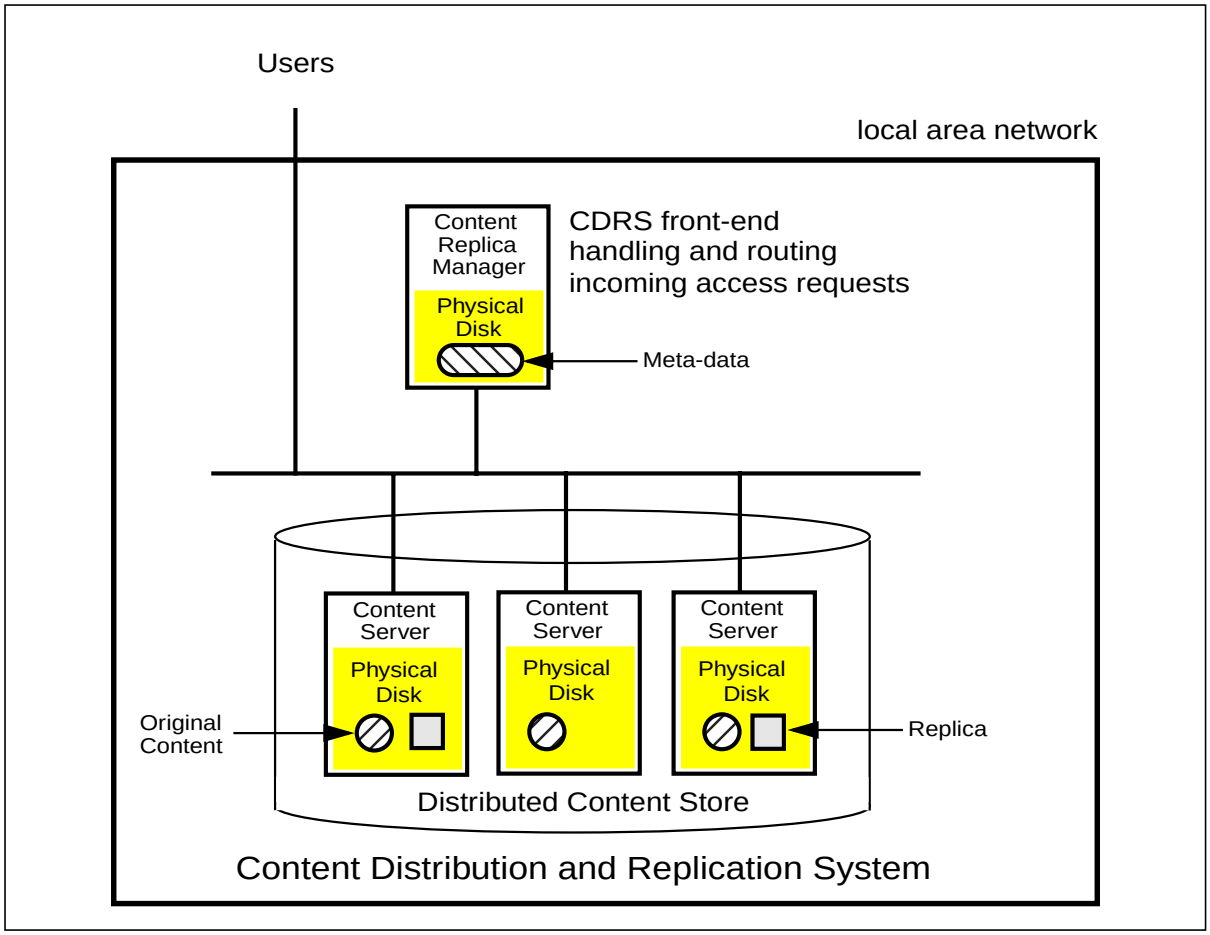


Figure 2: A general architecture of CDRS consisting of multiple servers in a LAN.

In this approach a complete web site is copied completely from one origin server to a number of mirrors, which are often geographically spread across the Internet. The content on the mirror is identical to that on the origin server. Thus users willing to access the web site's content simply choose one of the mirrors from a list offered to them and get the content they want. Since each mirror serves only a portion of the total requests, users can be served faster. Furthermore, users can always get the content as long as there is at least one server available, although the origin web server may be down. One of the critical inefficiencies of the mirroring approach is to handle multiple mirrors in order to ensure that the content on the mirrors is always up-to-date.

2.2.2 Proxy Cache Model

In general, a cache is a store of recently accessed contents that is closer than the original content themselves. When a new content is received at a client it is added to the cache store of the client, replacing some existing contents if necessary. When a content is needed by a user the caching service running on the client first checks the client's cache and supplies the content from there if an up-to-date replica is available. If not, an up-to-date replica is fetched. Caches

may be collocated with each client or they may be located in a proxy server that can be shared by several clients. Figure 3 illustrates a general architecture of the proxy cache where the proxy servers may form a hierarchy.

Caches are used extensively in practice such as for web and video-on-demand (VoD) services [MCH+01, SQU04, INK03, KOM02, BRL01, Gri00, RA99, IRD02]. For example, web browsers maintain a cache of recently visited web pages and other web resources in the client's file system. Using a special HyperText Transfer Protocol (HTTP) [FGM+99], web browsers make requests to check with the original server that the cached pages are up to date before displaying them. Web proxy servers (e.g., Local, Regional or Root proxy server in Figure 3) provide a shared cache of web resources for clients at a site or across several sites. This creates a large user community and because popular content tends to be accessed by many users, the cache on the proxy server is able to satisfy many requests and reduce the download times for the users, reduce bandwidth usage for the organization of the LAN, and reduce load on the origin content server.

Since the cache is of a limited size, we can only store a subset of the contents seen by the cache. When the cache is full and users want to store a new content, it is necessary to replace one or more contents from the cache. The replacement policy of the cache determines which contents are removed. There has been a considerable amount of research in cache replacement policies, e.g., [CFKL95, Wan99, KRR02, RS02] for web and [Gri00, Kan02, PB02, BPLS02, MO02, Zin03] for video streaming services. In practice, the most widely used replacement policy is the least-recently-used (LRU) replacement policy which is also widely used in virtual memory systems in modern operating systems [Pat96]. LRU is also the default policy in the freely available Squid caching proxy [SQU04]. Even though other replacement policies can in some cases achieve higher hit-rates in the cache, LRU is generally considered to perform sufficiently well.

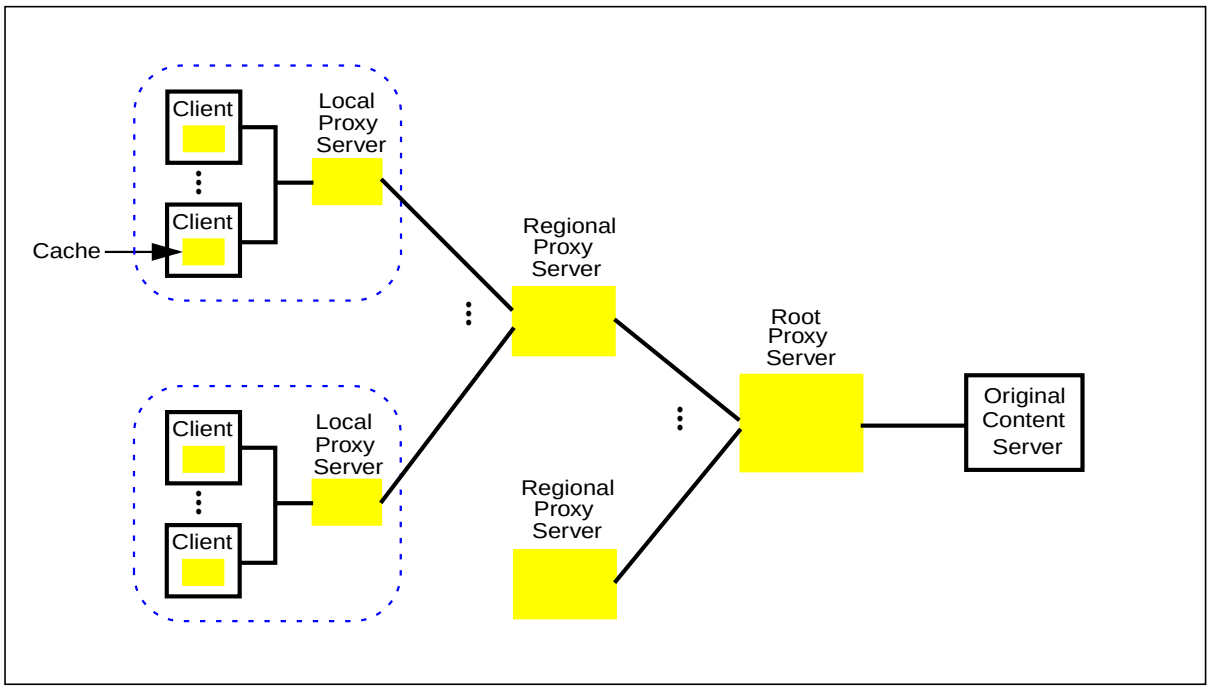


Figure 3: A hierarchical cache proxy server model.

However, installing caching proxies, whether a simple local proxy server or a caching hierarchy, has some disadvantages. The main problem is that the content provider has no control over the content once is has been retrieved from the origin server and placed into the caches. The content provider has no simple way of ensuring that if the content at the origin server changes, then all users would get the updated content. There are some simple mechanisms to inform caches about the content, such as the `Expires`-header or cache control mechanisms of HTTP/1.1, however these have not been widely deployed.

In order to avoid the problem of caches delivering stale, or out-of-date content, some content providers have resorted to marking their content as non-cachable so that the caching proxies would not store it. This problem was the primary motivation behind the development and success of content delivery networks.

2.2.3 Content Delivery Networks

Content delivery networks or CDNs are a way to improve Internet service quality. A CDN distributes and replicates the content from the origin to the replica servers scattered over the Internet and serves requests from each replica server close to where the request originates. This fact implies that a CDN can achieve short access delays and consume less network bandwidth. In addition, CDNs offer compelling benefits to content providers, including popular Web sites [DMP+02]. This is because a CDN can serve multiple content providers, and the shared resources offer economies of scale and allow the CDN to dynamically adjust content placement, request routing and capacity provisioning to respond to demand and network conditions [JCDK01, KKM02, WPP02, Kan02]. Moreover, the fact that many contents are not cachable but replicable makes CDN indispensable. This is because these types of contents include dynamic contents/objects with read-only access and personalized contents/objects (e.g., “cookied” requests).

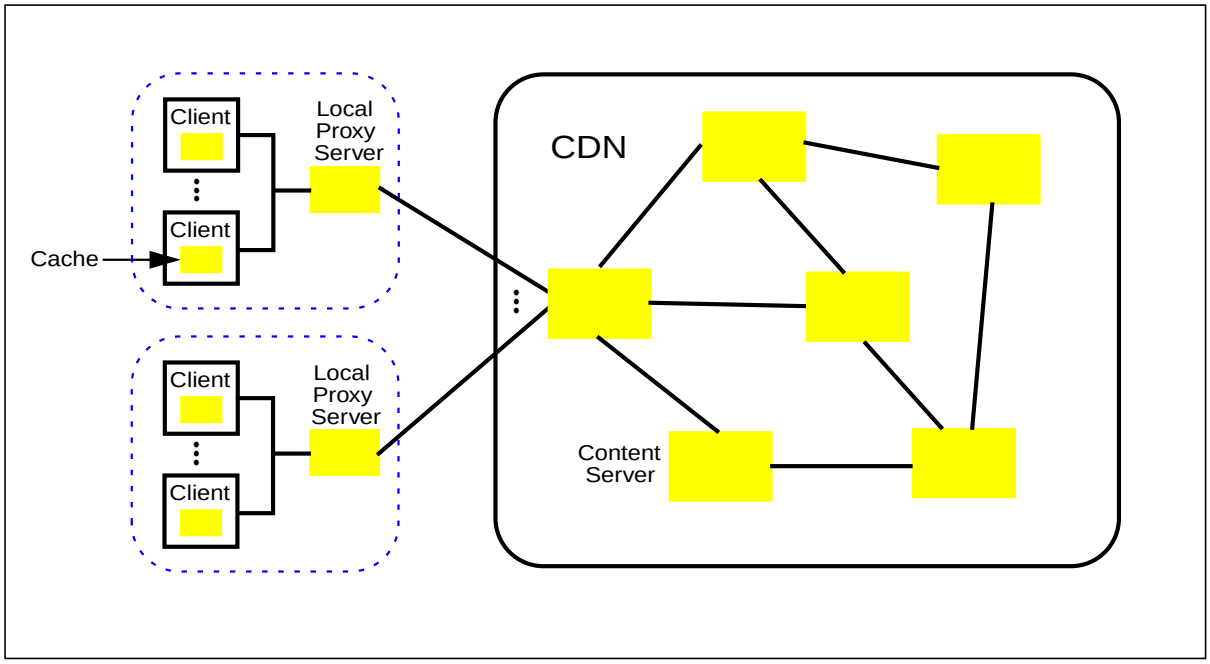


Figure 4: A general architecture of a content delivery network.

The general architecture of a CDN system is shown in Figure 4. Each of a CDN's content servers usually resides, for example, in the same "VPN" of the ISP [Hec04]. To distribute content to replica servers, a CDN establishes and maintains a distribution tree on an overlay network over the existing Internet infrastructure and disseminates content from the origin server to the replica servers via the tree or overlay. Akamai Technologies [Aka04] and Mirror Image [Mir04] use this Internet-based content distribution approach.

For content replication a CDN should decide where on the Internet the content servers should be placed, and how many replicas each content has and on which content server the replicas should be placed. In general, replica servers should be placed in such a way that they are closer to the clients, thereby reducing latency and bandwidth consumption. Also, content replicas should be placed to even the load of the replica servers trying to balance the load among replica servers [AR98, FBZA98, CKK02] in a CDN.

CDNs typically need mechanisms for routing clients' requests, i.e., selecting a suitable replica server that holds the copy of the requested content and direct the incoming requests to that server. Proximity between the client and the selected replica server and the replica server load are the two major criteria used to choose a proper replica server. In order to determine server load, techniques such as *server push* and *client probe* are often used. In the first technique, the replica servers propagate the load information to some agents, while in the latter approach the agents periodically probe the status of the servers of interest. There is a trade-off between the frequencies of probing for accurate measurement and the traffic incurred by probing.

Techniques that have been used to guide clients to use a particular server among a set of replica servers are such as DNS indirection [ST00, Kan02], anycasting [BAZ+97] and peer-to-peer routing (P2P) [ZKJ01, SMK+01, RD01a, RFH+01]. The basic distinction between the mechanisms is whether they support transparency for clients.

2.2.4 Peer-to-Peer Network

In a P2P network, there is no dedicated server, but instead all of the peers/hosts play similar roles, e.g., interacting cooperatively as peers to perform a distributed activity or computation without any distinction between clients and servers. Figure 5 shows a distributed P2P network where the control of content management (i.e., task for distribution and replication) is fully decentralized.

What sets P2P networks apart from the traditional forms of content distribution, caching and CDN's, is that in a P2P network every node/peer is both a client and a server. However, studies have shown that in reality, most of the content in a P2P network is served by a small minority of users and a large number of users do not offer any files [SGG02, SGD+03]. Regardless of this, P2P networks have become extremely popular for sharing files between users.

The first P2P network was Napster [Nap00] which allowed users to share MP3-files with each other. The main application for P2P networks has been file sharing, in which users make some files available on their computers and other users can download these files. In order for users to be able to find out which users are offering which content, the network needs some kind of a lookup service which maps content names into the machines serving these files.

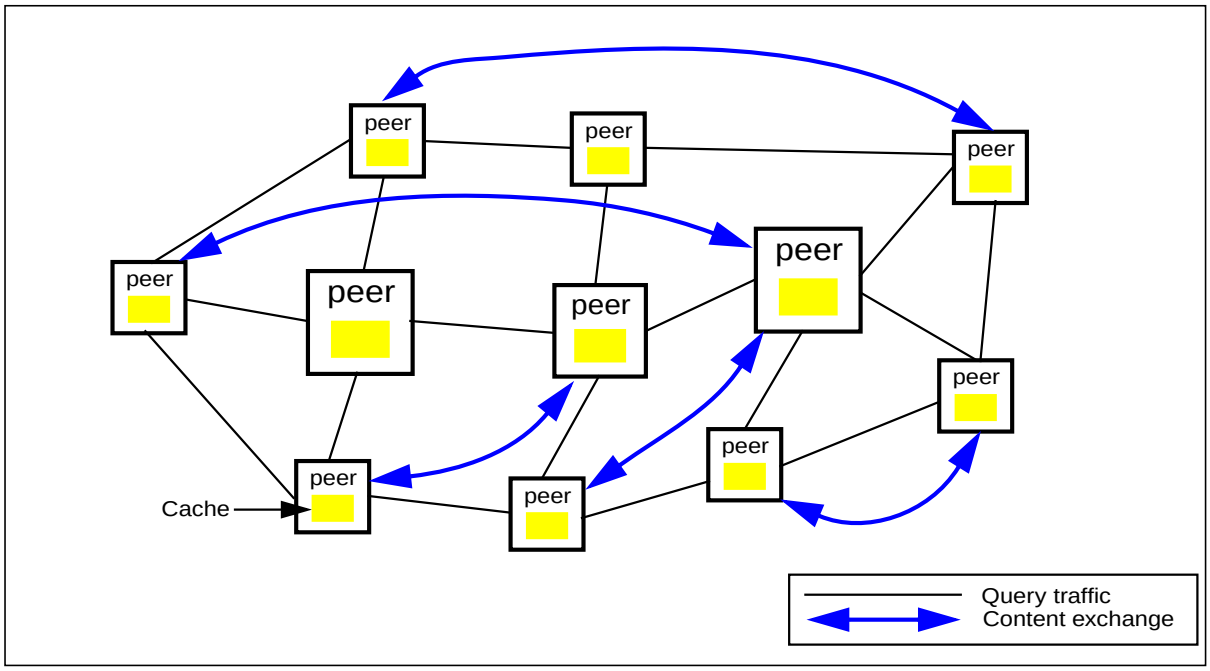


Figure 5: A decentralized peer-to-peer system model.

Napster had showed the advantages of the P2P content distribution model. Several new systems and protocols have been developed, for example Gnutella [Gnu04], Freenet [Cla99, CSWH00], FastTrack [Fas02], MojoNation [Moj02], and KaZaA [Kaz04]. There has also been considerable interest in the research community shown by the large number of P2P projects, such as CAN [RFH+01], Chord [SMK+01], Pastry [RD01a], Tapestry [ZKJ01], OceanStore [KBC+00], Farsite [ABC+02], Ivy [MMGC02] and Pond [REG+03].

2.3 Replica Placement Issues

Replication is an important scaling technique applied widely in many distributed systems and applications to enhance their services by improving performance and scalability of the systems [TvS02, CDK01]. However, replication also intends to increase service availability offered by the system in the presence of system or network failures and to decrease retrieval costs by local access if possible.

In a distributed content store, content replication involves taking a particular content (or a set of contents), making a complete copy of it, and subsequently spreading the copy across the distributed system. We will focus particularly on the decision issues of what to replicate, where and when to place replicas, and how to control the service quality, in particular the quality of availability achieved by the used replication strategies.

In the following sections, we present and classify the different replica placement properties and discuss their core features.

2.3.1 Server-Initiated vs. Client-Initiated Replication

In this section, we discuss two placement approaches which differ from each other due to the decision maker for replication, i.e., which entity (server or client) decides to replicate and manages the replicas. We distinguish between *server-initiated* and *client-initiated* replica placement. Depending on this decision maker the goals and mechanisms for supporting replication often vary strongly. The server-initiated model is more for increasing availability of service/data and for achieving more benefits for a content provider. While the client-initiated model is more for increasing performance in terms of reduced access time for clients.

Server-initiated Replication. The most important feature of server-initiated placement is that the servers of the distributed system control the replication. This is in turn ensures that the server can easily, in comparison to the client-initiated approach, collect service-related information such as the access count of any particular content. This information is helpful in adjusting the chosen replica placement to the network traffic and popularity pattern of the contents. Some well-known examples of the server-initiated replication are mirroring, replicated file server and VoD server.

Client-initiated Replication. In this approach replicas can be created, when clients access the content. After a successful access the content may be copied into local storage (e.g., cache) of the client. When a subsequent request is about accessing the same content, it is hosted by the local storage of the requesting client, and not from the server. The advantage of this approach is that the total access time can be reduced and further the network bandwidth consumption can be reduced. Thus, the performance of the total system is improved.

However, the content could be stale after a certain time period and keeping the content stored locally up-to-date is cost-intensive.

2.3.2 Granularity of Replication

Another important issue related to the replica placement is the *granularity* of replication, which determines the unit as well as the candidate content of replication. We distinguish between *full*, *partial* and *block-level* replication.

Full Replication. In the full replication the whole contents are the target of replicas. Multiple server based distributed content store, mirroring, and CDN use this approach for content replication. When applied to CDNs, the CDN takes control of the DNS mapping of the content provider's server, say `www.cnn.com`. When a client wants to request a content from this server, it first has to do a DNS lookup on `www.cnn.com` to get the server's IP address. The information in the DNS system for the domain `www.cnn.com` points to a nameserver in the CDN's network. When this nameserver receives the client's DNS lookup request, it determines which content server is the best located to handle this request and it returns the IP address of that content server as the IP address of `www.cnn.com`. When the client receives the DNS reply, it will attempt to connect to the content server. Because the content server is closer to the client than the origin server, the client will receive the requested content much faster.

The problem of this approach is that each content server must be able to handle all requests for the content provider's origin server. This implies that either each content server fully repli-

cates the contents of all content providers with which the CDN has passed an agreement, or that the content server acts as a surrogate proxy [DMP+02].

The benefit of this full replication model is that all clients requests are always sent to the content servers.

Partial Replication. In the partial replication only a subset of the total contents are replicated within the system, either to all servers (peers) or to some of the system. Thus, various intermediate content groupings are possible, these decide the granularity of replicas. In general, coarser granularity may increase the overhead for transferring contents between servers (peers) as well as for storing them, while finer granularity imposes higher overhead for collecting and maintaining access statistics as well as for placement decisions.

An additional issue of the partial replication is the decision about the target of replicas. In some cases of dynamically generated pages, a group of files, executables, and other environmental state must be migrated together, because they are used together for page generation. The cost of transferring such bundled contents may be high and should be taken into account in a replica placement algorithm. In addition to manual bundling of contents, some contents could conceivably be bundled together based on similarity of access patterns or dependency between the contents. The policies for and the feasibility of doing this is studied in [Kue97].

An additional issue of partial replication is the degree of replication (or replication ratio), i.e., how many replicas should be created for any particular content. Without going into the details of the placement solutions here[1], one can intuitively argue that the quality of service offered by the system, in terms of the achieved service availability or performance, increases with the number of replicas. However, it is shown that increasing the number of replicas beyond a certain number does not significantly reduce server load nor client download time on Internet-like settings, see [JJK+01]. This is because of the cost of keeping replicas consistent and the increased bandwidth and storage requirements of the system.

Regarding the replication degree, we can also consider a '*replication-per-content*' approach where the degree of replication is different between all replicated contents - each content may have a different number of replicas. The reasons for this approach are the different levels of popularity and importance of the contents, as well as their different characteristics. The downside of the replication-per-content model is the considerable complexity, because the model considers that different hosts hold different amounts of replicas, which will be exposed to different workloads.

Block-Level Replication. In block-level replication which is basically the same as parity schemes such as RAID [SN04], each content is split in equally-sized blocks which may be the same block unit supported by the underlying system (e.g., file system). The block-level replication is typically used in VoD systems where the size of video data is 'relatively' large [Gri00]. Although this approach does not provide the robustness necessary to survive the high rate of failures expected in the wide area, it reduces significantly the amount of bandwidth and storage required for supporting replication.

In order to provide redundancy without the overhead of strict replication the block-level replication can be used in conjunction with an erasure coding technique [BMSV02, HKRZ02].

1. We will present the placement solutions in detail in Chapter 4, 5 and 6.

An erasure code divides a content into *L* fragments and recodes them into *M* fragments, where *M* > *L*. The rate of encoding *r* with *r = L / M* increases the storage cost by a factor of *1/r*. The key property of erasure codes is that the original content can be reconstructed from any *L* fragments. For example, using an *r = 1/4* encoding on a block divides the block into L = 16 fragments and encodes the original *L* fragments into *M = 64* fragments; increasing the storage cost by a factor of four.

2.3.3 Deciding Where to Place

Another major issue of placing replicas is the location of replicas. This is because the 'goodness' of placement is typically affected by facts such as how reliable and fast the servers and the underlying network paths are, and moreover how available the system's resources are to serve the users' content access requests. Furthermore, this placement decision is often strongly affected by the requirements and characteristics of target applications, performance and service availability.

Depending on these replication requirements, replicas can be placed either near to content servers which host the clients, or near to clients which request the content, or between servers and clients.

Near-to-Provider Replication. The most important/appealing advantage of this placement approach is ensuring that the server (or the content provider) has good control over the replication (quality). For instance, if the server needs to update the contents, the total cost incurred due to the update process, in particular to the use of network bandwidth for transferring the updated data, may be lower than that of transferring the updated contents to a longer distance.

Near-to-Consumer Replication. In comparison to the near-to-provider replication, replicas can be placed near to the clients (or the content consumers) specifically to reduce the access time. Additionally, the clients can store the content that they accessed/downloaded into their local storage so that the content can be read faster when accessed again. Disadvantages of this placement approach are the higher cost for updating the local replicas and the total resource consumption, especially when there are many replicas on the clients' local storage.

2.3.4 Static vs. Dynamic Replication

One additional critical placement problem to be resolved is deciding when replicas should be created. We distinguish between *static* and *dynamic* replica placement.

Static Placement. If one can know, for example, when the server may go down, which content is very popular or important, or which content will be accessed soon, then it may be straightforward to decide the granularity and target of replicas, as well as their location. The target contents are then replicated prior to their access. Thus, the static placement approach often assumes that there is global knowledge about the service requirements and system condition available to users.

In static placement, once the replicas are created and placed, they are usually not replaced or updated during the service time, but instead typically a system administrator is involved in updating the replicas' placement, often manually, after a 'relatively' long service time, e.g., a

month, a couple of months or one year. Thus, the static placement approach is typically applied to permanent replication, where the replica update frequency is low.

One critical inefficiency of the static placement is that the replication system cannot take the dynamics of the system into account, e.g., changes in access traffic, network condition, or so-called 'hot-spots' which can build unpredictably due to any 'big' news.

Dynamic Placement. In a dynamic placement model new replicas can be created and eventually replace an existing one on-the-fly during service time. This approach is particularly useful for CDRSs such as P2P systems, where the availability of the underlying system components and their resources change very frequently.

An important issue of dynamic placement is deciding when to re-evaluate the current placement and when to perform replica replacement. Doing it frequently wastes computing resources whereas doing it too rarely would decrease overall system performance and availability. The simplest scheme is to take a fixed time interval for this adaptation to the placement, such as once a minute. However, this approach does not allow CDRSs to react quickly to sudden changes in access patterns. It is more efficient to adapt as soon as such patterns change.

To detect these changes, each host may monitor a number of variables such as frequency of requests and average response time. Significant variation in one of these variables indicates a change that would warrant replacing the current strategy. [AR98, Gri00] presents a number of techniques for computing such variation.

2.4 Summary

In this chapter, we presented an overview of techniques for content distribution and replication systems, including how these technologies have evolved on the Internet in the past few years. In order to provide an insight into content distribution and replication systems, the most relevant content distribution architectures are presented. Starting off with the conventional client-server model, we first discussed the features and limitations of the mirroring and multiple server based architecture. The second approach was client-side proxy caching, which was initially implemented with caching proxy servers, installed locally at LAN or MAN, later on these caches were used to create caching hierarchies. However, the caching approach did not allow the content provider any control over how the content would be cached. Content delivery network (CDN) emerged to remedy this problem and they have become the de-facto content distribution and replication method for most large commercial web content. We then presented a new content distribution paradigm, namely decentralized peer-to-peer model which differs from the client-server model in that each peer in the system is both a client and a server.

We also have discussed some major issues of content distribution and replication systems. We have concentrated particularly on the decision issues of what to replicate, where and when to place replicas, and who decides the replication. Regarding the granularity of replicas, which determines the unit as well as the target of replica, we distinguished between full, partial, and block-level replication. Finally, we discussed static and dynamic replica placement approaches. In Chapter 4, 5 and 6, we will tackle the replica placement problem with different settings of the replica granularity (full and partial), time scale (static and dynamic) and content distribution and replication system architecture (CDN and P2P).

Chapter 3 - Quality of Availability

The importance of satisfying service availability is becoming one of the most critical factors for the success of Internet services and applications. This chapter investigates concept, mechanisms, and technical requirements for satisfying service availability in content distribution and replication systems on the Internet. We take an availability-centric view on QoS and focus especially on the issues of providing availability guarantees and satisfying different targets of availability requirements for different users and content providers. In this chapter, we propose a new concept, *quality of availability* (QoA) that enables us to treat availability as a new controllable, observable QoS parameter. The QoA concept supports multiple availability levels to cater for the different requirements of different users and applications. We suggest some refinements of the availability definition. These are necessary for specifying differentially the availability requirements as well as evaluating quantitatively the achieved service availability. For a technical realization of the QoA concept, we develop a QoA framework which comprises key components such as QoA mapping, admission control, and replica management, for provisioning and providing the target services with QoA support. We also define QoA metrics and parameters that will be used in quantitatively evaluating the QoA fulfillment throughout this thesis.

3.1 Motivation

Most research efforts in the areas of highly available distributed services and fault-tolerant systems have their focus on achieving 99.99% or 99.999% (“four-9’s” or “five-9’s”) server availability [TDMV96, SBL99, RD01b, ABK+01, RIF02]. However, there is a demand for service differentiation from service consumers and providers due to the cost and competitive nature of the marketplace. This means that the mechanisms used for increasing availability should support different levels of services and service availability. In fact, the need for service differentiation can be observed on different Internet services and applications, such as news-on-demand or video-on-demand over the Internet [JAC98, CS00, PKvST00, FC02], as well as different users requirements [Hen99]. These requirements depend on the service type they demand, on the service time when they access, on the peripherals they have, and on the service price they pay.

From the service system provider’s point of view, not all system components need to offer the same redundancy (i.e., the same availability level). The availability level required for individual system components depends on how reliable they should be and whether they are critical for offering the service. For instance, in developing replication mechanisms for increasing availability of services and their data in a distributed multimedia system, medianode [OSS01], we analyzed the characteristics of multimedia contents and their meta-data. We then identified

that not all service operations and not all data access functions require the same availability level of the “five-9’s”. Table 1 shows an overview of the data types with their characteristics, that are handled by medianode [OSS01]. For example, the multimedia resources (in the fifth-row) which are the ‘raw’ presentation contents require a higher level of availability than that of the system resources (in the sixth row) which are used as ‘hint’-like information for a better resource management (for details on the analysis of these data and the replication system for medianode see Appendix A). Unavailable presentation materials (multimedia resources) can prevent users (e.g., teachers and students) from lectures, while the unavailability of the system resources can lead to a degradation of the service quality.

target data	availability requirement	consistency requirement	persistency	update frequency	data size	QoS playback
presentation description	high	middle (high)	yes	low	small/ middle	not required
organizational data	high	high	yes	low	small	not required
file/data description	high	middle	yes	middle	small	not required
multimedia resources	high	middle	yes	middle	large	required
system resources	middle (low)	middle	no	high	small	not required
user session/token	high	high	no	high	small	not required

Table 1: Data categories and their characteristics in medianode.

This observation motivates us to investigate concept and mechanisms, which support content distribution and replication systems for satisfying target levels of service availability for Internet services and applications. Although the concept and mechanisms of service differentiation have been studied in many QoS-related work [SPG97, BBD+98, Wan01, AG01, LALT02], they are mainly performance-oriented. Their application to differentiating service availability has seldom been addressed.

3.2 Outline

The rest of the chapter is structured as follows. In Section 3.3 we present the concept, quality of availability (QoA) and discuss the benefits of using the QoA concept. Section 3.4 defines service availability for largely interconnected content distribution and replication systems (CDRS) and presents our three refinements of the availability definition, *decoupled*, *differentiated*, and *fine-grained* availability. Section 3.5 presents the QoA framework and describes the process of turning QoA-based service requirements into a QoA realization. It also provides the semantics of QoA service specifications with which CDRSs and users can communicate the requirements and expectations of a service commitment. Section 3.6 defines a number of QoA

metrics that are used in quantitatively analysing and evaluating the reached QoA in the next three chapters of this thesis. Section 3.7 discusses related research in the areas of QoS and availability with the focus on supporting QoS in availability enhanced services to enable differentiated availability classes. Section 3.8 concludes this chapter.

3.3 The Concept of Quality of Availability

Figure 6 illustrates the basic ideas for the quality of availability (QoA) concept. Regarding QoS, we take an availability-centric view on QoS. Indeed, we move the focus of the objective function for the resource and performance optimization problems of the QoS field from satisfying transmission-dependent characteristics such as minimizing transmission delay, jitter, or data loss to satisfying the availability requirements such as minimizing failure time of service systems and their components and to maximizing the total service uptime.

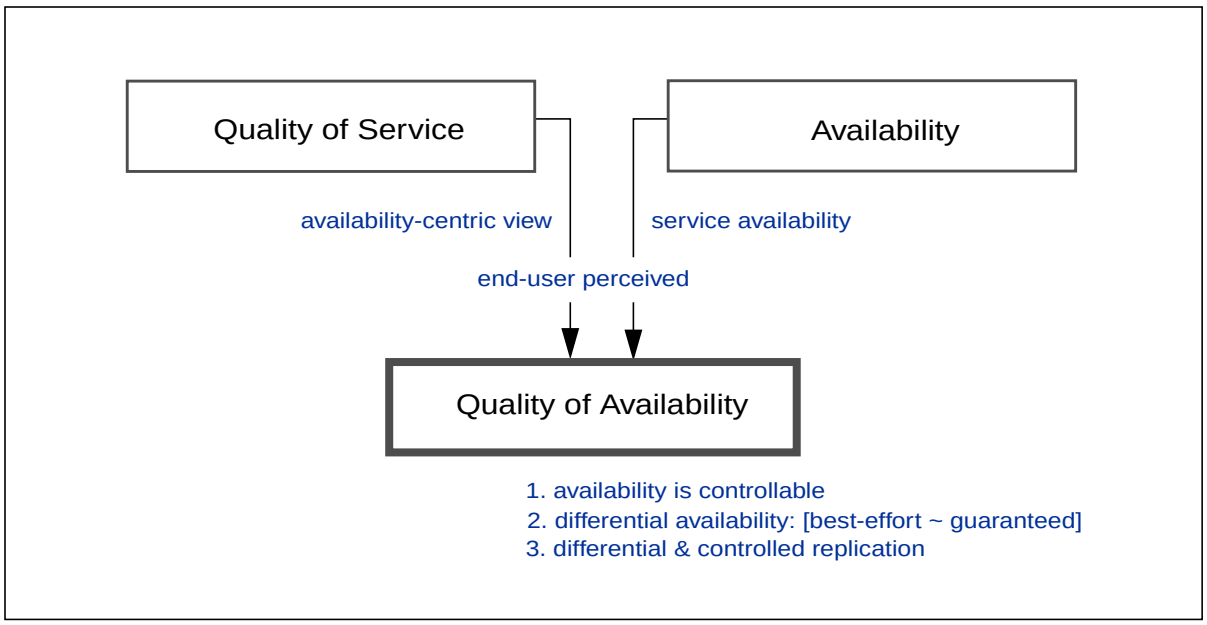


Figure 6: An Illustration of the quality of availability concept.

For the availability support we concentrate on *service availability* and try to meet the target levels of users' required availability. This is in contrast to existing availability studies which investigated mainly how to increase the system availability by either creating highly reliable systems or increasing the systems' redundancy. This is due to the fact that the user perceived availability is typically not only the reflection of the minimized downtime of the system, but instead it depends on the actual service uptime which should be determined by identifying the availability of individual service system components. Furthermore, we apply the QoS concept to content distribution and replication systems, and treat different replication techniques used for increasing service availability as different QoS service classes within a unified CDRS framework.

By combining the two concepts - service availability and QoS, we develop a new concept which is termed "quality of availability (QoA)" enabling the following features:

- availability can be treated as a new controllable, observable QoS parameter.
- availability can be specified with multiple service classes, e.g., best-effort or guaranteed availability class.
- the target and degree of replication may vary depending on the required availability classes.

We define QoA for content distribution and replication systems, based on a QoS definition in [Sch01a, SN04], as:

> *"well-defined and controllable availability (behavior) of a service according to the quantitatively measurable parameters, such as data availability, resource availability, and system (i.e., node and connection link) availability"*.

Furthermore, multiple QoA service classes can be specified in various forms such as the cases in network QoS [Sch01a, SPG97, BBD+98]:

- *deterministic* - This is an absolute guarantee for a specific service availability requirement. An example form of the deterministic guarantee is: a service (or its data item) is reachable all the time with an availability guarantee of 100 percent (%).
- *stochastic (*or *probabilistic)* - A service with stochastic guarantees is treated better than best-effort services (e.g., higher service uptime, higher replication degree, more storage capacity). Yet, this is not a hard guarantee. An example form of the a stochastic guarantee is as follows: the probability that a service availability is guaranteed to be at least, e.g., 90% of the whole service access requests should be higher than 99%.
- *best-effort* - The best-effort service availability is basic availability with no guarantees. This is typically the case where CDRS may have no concern on controlling the QoA, e.g., there is no explicit support for a service to meet its target level of service availability. The CDRS may only try to maximize the service availability for the service.

The goal of this work on QoA is then to satisfy service availability as QoA. Thus, given a set of different levels of availability requirements and a network topology with or without a finite number of possible replica locations, we are then interested in how many replicas are needed, where they should be placed, whether their placement on the given topology satisfies the individually required availability QoS and how they affect the overall service availability quality.

In the following sections, we refine availability definitions to enable the specification and evaluation of different availability levels required by users and achieved by services, respectively, and present the QoA framework that is developed for a technical realization of the QoA concept.

3.4 Availability Refinement

This section provides the definition of availability used throughout this chapter. Moreover, we investigate some refinements on the availability definition. These refinements enable us to specify multiple levels of the application-specific availability requirements and to evaluate quantitatively the reached service availability.

3.4.1 Availability Definition

In the literature [TvS02], the term “availability” is often used when referring to whether a system functions as expected and its data is reachable. The traditional definitions of availability are typically based on either how reliable the underlying system is or whether the system has any built-in features of failure[1] detection and recovery, or whether the system has any redundancy for its individual components. Thus, the availability can be defined in terms of the existence of a failure management facility or redundancy support.

Throughout this thesis, we use the two following basic availability definitions for declaring system or data availability:

- **Reliability[2]-based.** For a reliable system which has failure management facilities, such as failure detection and recovery functionality, its system availability is defined as the percentage of time during which the system is available to the total service time. Thus, when the mean time to failure (MTTF[3]) and the mean time to repair (MTTR[4]) for a system are known, its system availability is calculated as:

$$Availability = \frac{MTTF}{MTTF + MTTR} \quad (1)$$

  The availability of such a system can be increased by creating system components that are either very reliable (very high MTTF) or able to recover from failure very rapidly (very low MTTR).

- **Redundancy[5]-based.** The availability of a distributed system that may comprise of multiple interconnected servers is strongly affected by server failure and network partitions. The data availability in such a system can be increased by replicating the original data on two or more servers. This approach, replication, enables users to access the data or its replica at any of a number of working servers. If each of the servers has an independent failure probability P_f, then the availability of the data stored at the whole system can be calculated as:

$$Availability = 1 - (P_f)^n \quad (2)$$

Even though these definitions can be applied to a content distribution and replication, there is a limitation due to the fact that they cannot explicitly capture the availability of individual system components or the reachability of any data required by the system. In particular, when these individual system components that affect the quality of supplying service availability have different failure probabilities. For example, an availability value of 99% does not indicate whether it is due to the failure of a disk or system node, or network partition.

---

1. A failure is a reflection of an incorrect or unacceptable result with respect to a specification or unexpected behavior perceived by its users. The cause of a failure is said to be a fault which can be identified or detected either by the system or by its users [For01].
2. Reliability is a measure of the continuous delivery of a service in the absence of failure, while availability allows for service failure [For01].
3. This represents the time interval in which the system can provide service without failure.
4. This represents the interval in time it takes to resume service after a failure has been experienced.
5. Redundancy, in our context, involves provisioning of more resources to provide a service than would otherwise be needed - the extra resources are used when a primary resource fails.

As a consequence, we need to refine these basic availability definitions to capture the availability of all the individual system components. In the following subsections, we propose three availability refinements: *decoupled, fine-grained,* and *differentiated* availability.

3.4.2 Decoupled Availability: Demand vs. Supply Availability

One critical challenge with using the basic availability definitions introduced in the previous subsection is to quantify the 'goodness' of the supplied availability with respect to the required one. For example, with an availability level of 99.9% supplied by the service system, we do not know whether it is the same availability level that the user perceives and, if there is a gap between the supplied and required availability level how large the gap is. Being able to identify this gap can accelerate the process for resource reallocation for satisfying the required availability level.

The other reason for this refinement is that the user perceived availability is often different from the availability supplied by the system which mainly concentrates on increasing its reliability. Furthermore, the user perceived availability is often affected not only by the system's downtime, but also by the length and frequency of each failure.

Motivated by these two factors, we distinguish between availability classes which a CDRS supplies from the availability classes which users (or content providers) request and perceive. The potential benefits of this refinement are:

- It enables one to check the gap between the required and supplied availability levels. Based on the result, the system can be configured to close this gap for the user.
- It enables one to specify their own availability requirements, while the system can be controlled to meet many or all targets of the required availability levels.
- It enables one to check how much the service system maximizes availability (checking optimization ratio), as well as whether it satisfies the required availability class (checking QoA guarantee).
- Users may specify their own availability requirements in a similar manner to the QoS requirement specification, e.g., best-effort (maximizing service availability) or guaranteed (99% minimum service availability always).

3.4.3 Fine-Grained Availability

The basic availability definitions introduced in the previous section capture either the total system uptime which takes failure and repair times or the redundancy degree into account. However, the definitions do not explicitly address the failures of individual system components, e.g., connection link failure or node's resource failure. Thus, they are very limited for applying them to quantifying achieved availability for a CDRS comprising a set of interconnected system nodes. So, we refine the service availability definition as follows. For calculating the availability of each component, such as data, node, and link, we use the basic availability definitions (Equation (1) and (2)):

$$Avail_{Service} = Avail_{Data} \times Avail_{System} \tag{3}$$

$$Avail_{System} = Avail_{Node} \times Avail_{Link} \tag{4}$$

$$Avail_{Node} = Avail_{NodeDynamics} \times Avail_{NodeIntrinsics} \quad (5)$$

Thus, the service availability for a wide-area distributed system can be defined as:

$$Avail_{Service} = Avail_{Data} \times [(Avail_{NodeDynamics} \times Avail_{NodeIntrinsics}) \times Avail_{Link}] \quad (6)$$

This fine-grained availability definition captures the following features:

- a service is available when both its data and the system on which the service is running are available;
- data is available when it is reachable at access time throughout the whole system;
- a system is available, when both its (computing) nodes and connection links are available;
- a link is available when it does not fail and there are enough resources which can be allocated for transmitting the requested data for the demanding application;
- a node is available when it is up, i.e. not disconnected from the network, and its resources can be allocated for processing the service request. Memory, CPU cycle, and storage space are examples of such resources.

Keeping these basic definitions in mind, we distinguish between three levels of available systems:

- *basically available*. At this level, a system delivers correct functionality as long as no failures occur, but it neither offers any redundancy for its components and data, nor failure detection and recovery mechanisms;
- *highly available*. In addition to the feature of the basically available level, a system in the highly available level provides a certain level of redundancy and eventually the mechanisms for fault-tolerance support;
- *continuously available*. A system in the continuously available level is capable of degrading gracefully to preserve as much critical functionality as possible in the face of server failures and network partitions. A continuously available system needs to be able to switch compatible services in an established connection and substitute acceptable alternatives. It must also be able to dynamically adapt to the threats in its environment to reallocate essential processing to the most robust resources.

3.4.4 Differentiated Availability

A CDRS may frequently host multiple applications, users and content providers. In such an environment the service availability requirements can be differentiated as follows:

- Different users, applications and content providers require different availability levels. For example, not all applications require the highest availability level of ‘five nines’, but instead an appropriate level which satisfies its application-specific requirements. A similar phenomenon can be observed within a single application in which individual users demand different levels of availability due to the users interest or the amount of available resources of their access devices.
- Different services and contents have different importance levels. In a single user’s point of view, the importance of a service or a content is strongly related to its type (e.g., on-line

banking or web browsing) and the purpose of access (e.g., payment transaction, trend survey or project result upload etc.).

- Availability levels are affected by the time of day. For example, let us imagine that a user tries to download a document file that is very important for preparing a presentation that is scheduled at 9 am. The required availability level of this document for the user prior to the presentation time may be very much higher than after the presentation time, e.g., 6 pm.

Figure 7 shows an overview of the three refinements that we propose. 'Demand Availability' can be easily mapped to a single service usage scenario in which the different availability classes (for example, A0, A1 and A2) are affected by the time of day (9 am, 3 pm, and 6 pm). Instead of assuming the different time of day we can also take different applications where the availability levels may be different among applications.

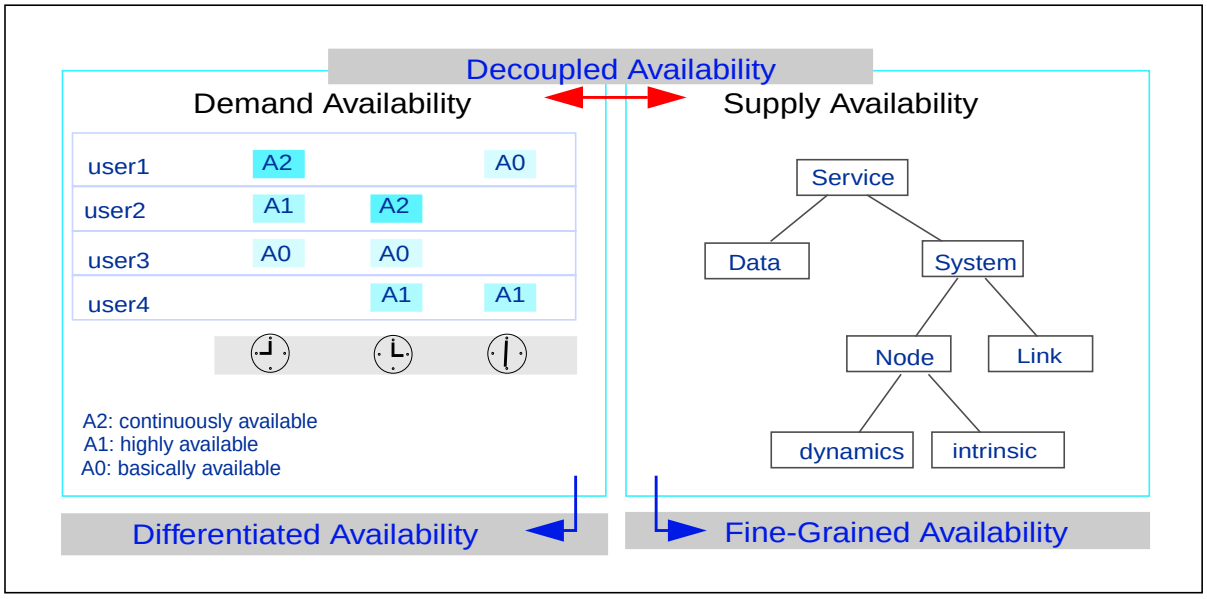


Figure 7: An illustration of the three refinements of availability definition.

3.5 QoA Framework

This section presents the QoA framework, its architecture and key mechanisms. This QoA architecture is able to support, in one integrated framework, widely distributed and replicated services in different quality levels with availability guarantees.

3.5.1 An Architectural Overview

The architecture of the QoA framework introduces the following components for QoA implementation:

- service specification module
- QoA mapping module
- admission control module
- replica management module

The QoA framework binds these components and defines the interfaces between them. Figure 8 illustrates the process of turning QoA-based service requirements into QoA realization by mapping and managing system resources, i.e., determining the number and place of replicas.

In the service specification module, clients or content providers can specify their level of service availability requirement. Associated with the service are (a) its QoA requirements, (b) the service profile regarding data served, such as data size, or even some a-priori information about the data access pattern. The service specification can be generated by using the QoA requirement specification semantics (presented in Section 3.6) and be forwarded to the QoA mapping module.

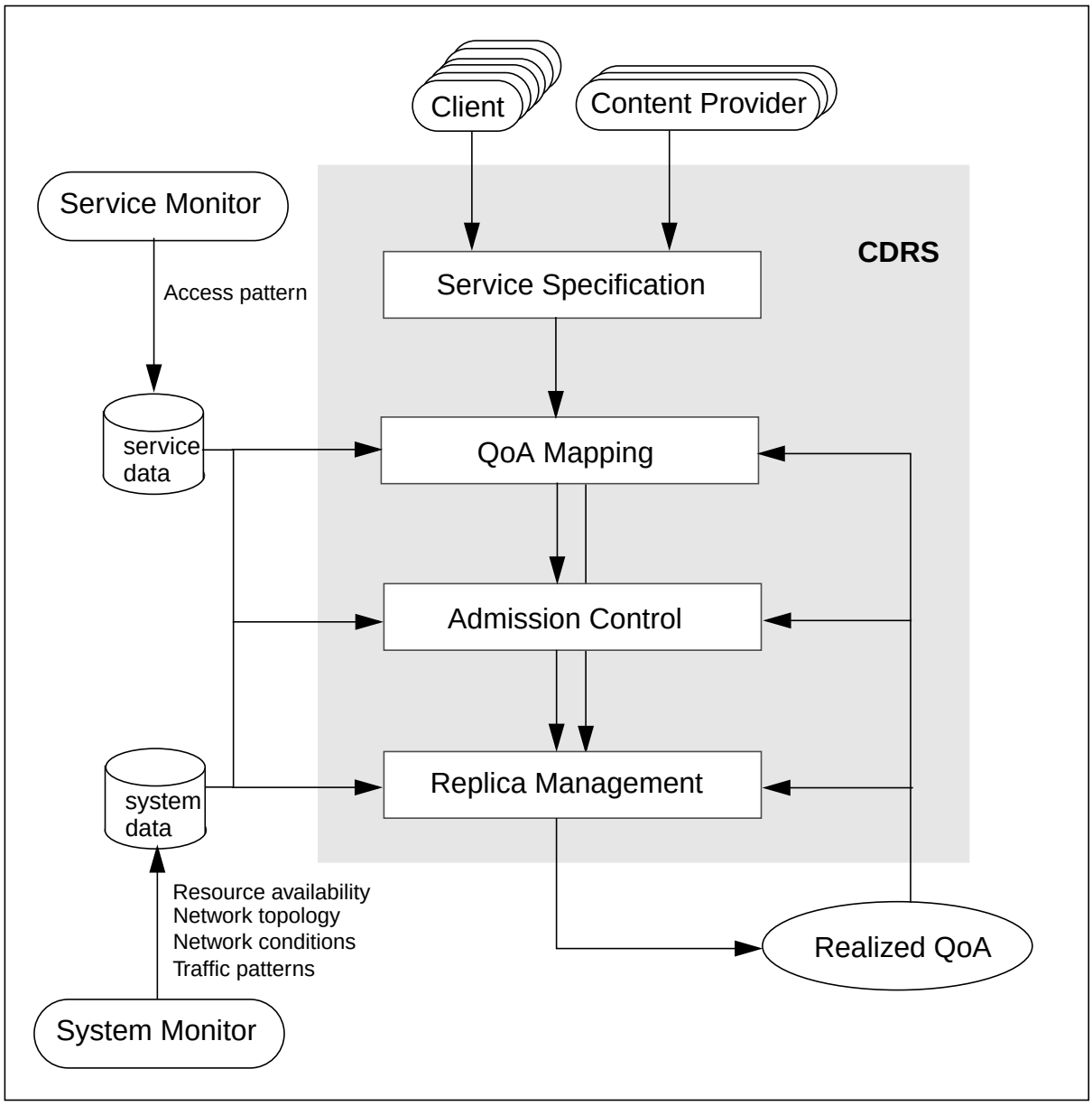


Figure 8: Architectural overview of the QoA framework.

In the QoA mapping module, a CDRS maps the QoA requirements into an optimal set of specific resource requirements in terms of the target, number and location of the replicas to meet the target service availability level at minimum resource cost and consumption. Addition-

ally, the replication policy (e.g., pull or push, static only or dynamic) can be determined. For performing the mapping task, the QoA mapping module uses, as the admission control and replica management modules, information available to it regarding network topology, availability of the system resources, (e.g., storage and access link capacity), which are traced by service and system monitors and maintained in a data repository.

The admission control module focuses mainly on guaranteeing QoA. After mapping the high-level service requirements into low-level replication (resource) requirements, such as a target placement with selected replica nodes, the individual replica nodes will have to be consulted to check whether they are able to admit the service request. If one or more of the replica nodes reject the request, then an alternative QoA mapping, i.e., an alternative placement must be computed, and the process repeated. If all the replica nodes accept the request, then the service can be established. The admission control module is necessary for meeting the target level of the required service class with a QoA guarantee. This also means that the admission control step can also be bypassed for the 'best-effort' QoA level which needs no QoA guarantees.

The replica management module is used either to maximize local or global resource utilization, or to maintain service commitments when faced with changes in the system conditions (e.g., node failure or network partitions). The replica management consists of a set of submodules; these are modules for replica list management, consistency maintenance, and replica replacement.

3.5.2 QoA Specification

In this section, we establish the semantics of the QoA service specification so that content providers, clients, and CDRS providers can communicate their requirements and expectations of a service commitment. As is the case for the QoS concept, users (clients or content providers) make a QoA specification, while the system (e.g., CDRS system) manages the QoA.

The QoA specification at service-level consists of three elements: *service type declaration*, *availability requirements* and *service profile*. As described in Section 3.3, there are three service types: best-effort, stochastic and deterministic; while the first one gives no guarantee, the other two types offer a guarantee. The availability requirements can be expressed along one or more of the following dimensions[6]:

- service availability
- data access latency
- acceptable data inconsistency

The service availability can be expressed either directly (e.g., 99%) as a service uptime or as a combination of separated data redundancy and system availability. The data access latency is one of the typical performance-constrained QoS parameters. By taking the latency into account, the QoA framework can offer a dual availability and performance-enhanced QoS for a CDRS and a service running on top of the CDRS. This makes the QoA model more realistic, because service consumers (clients) usually expect to receive a service within an acceptable response time, when they require the service, and when it is assumed that the service is avail-

6. Although the work in this thesis is focused on the service availability, the QoA concept can be extended to a form that also supports performance (QoS) and consistency issues.

able. The service profile declares the amount of resources (e.g., storage capacity, access link capacity), the time and duration of the reservation, and the distribution of data accesses, if known. Table 2 provides some example QoA specifications for illustrative purposes. Contractors which specify the QoA requirement are either content provider or client.

Service type	Availability requirement	Service profile	Contractor
Deterministic	99.9% minimum service uptime, 1% maximum data inconsistency	10 GB storage capacity, 1 month, popularity-oriented	content provider
Stochastic	Probability[service uptime < 99%] < ε, Probability[inconsistency > 5%] < ε	10 GB storage capacity, popularity-oriented	content provider
Best-effort	maximizing service uptime, minimizing worst-case hop count	10 GB storage capacity, 3 months	content provider
Deterministic	99.9% minimum service uptime, maximum 3 hops	1 Mbps link bandwidth, 1 month (Oct.2003)	client
Best-effort	maximizing service uptime	1 Mbps link bandwidth	client

Table 2: Examples of high-level QoA specification

3.5.3 Resource Specification and QoA Mapping

The QoA mapping module, hereafter called *QoA mapper*, performs a resource mapping function, where the high-level service specification described using the availability class specification and a service profile is mapped into low-level resource requirements. Once the users (e.g., a content provider) have forwarded a service specification to the QoA mapper, it determines the target, number and location of replicas to meet target levels of the required service availability for the users. In the mapping process, the QoA mapper may use, if available, information about the network topology, data access patterns and resources' (e.g., storage, link bandwidth) capacity. Finally, the QoA mapper finds a placement which either maximizes the service availability or satisfies the users' service requirement with a QoA guarantee, depending on the QoA class required by the users. Figure 9 illustrates the individual steps of the process of mapping user-specified availability requirements into the replication-level resource requirements and selecting a placement within the QoA framework in detail. The data in Figure 9, such as the service class, availability requirement, service profile, etc. are only an example.

The processing steps are described as follows:

1. A user forwards a QoA specification (i.e., a required service availability level and a service profile) to the QoA mapper.
2. Information about system and service usage can be used by the QoA mapper, if available.
3. The target of replicas is determined. For instance, a partial replication policy is chosen, where only the top 10 popular contents will be replicated.
4. It is determined how many replicas should be created for the selected contents.

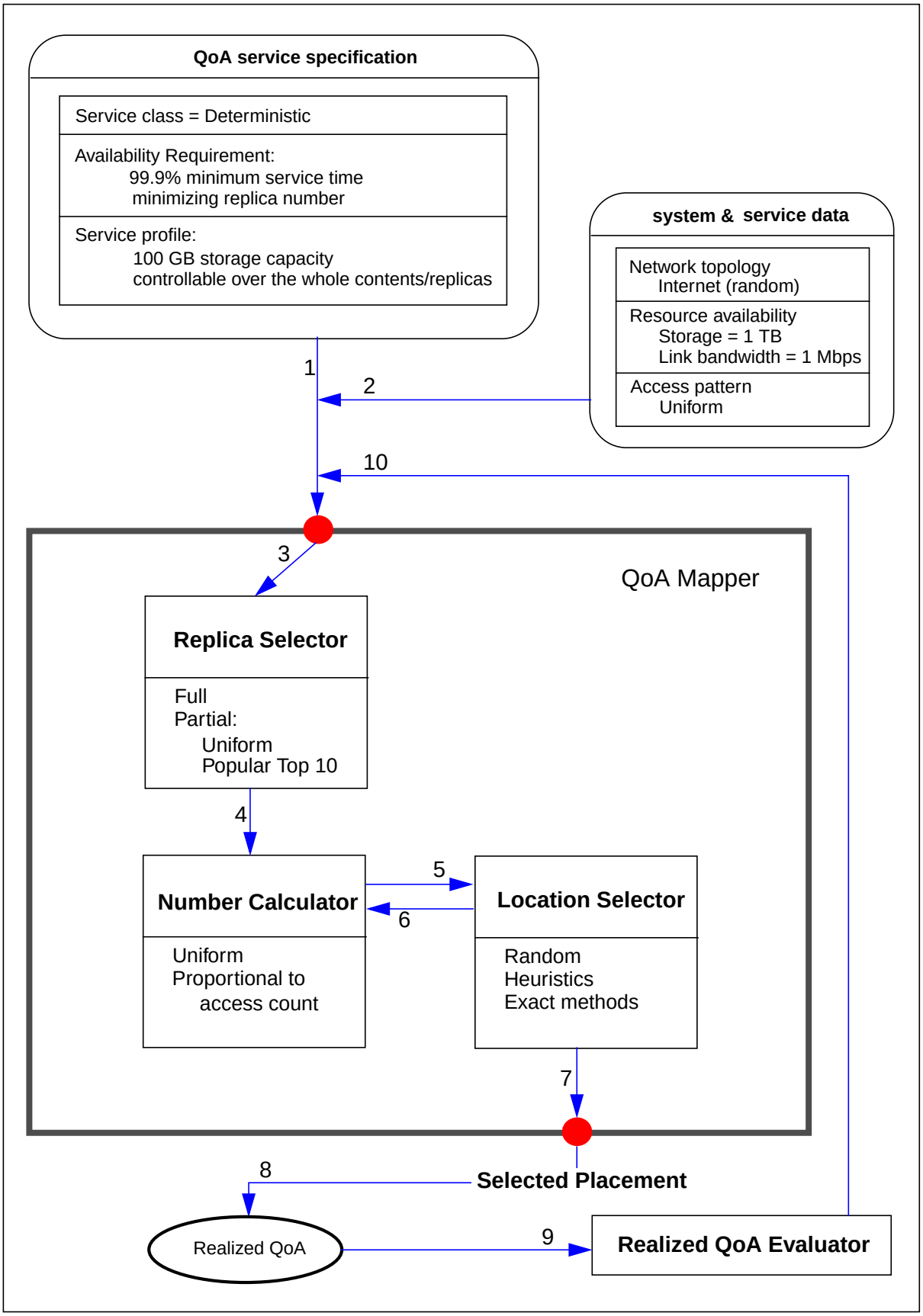


Figure 9: An illustration of the QoA mapping process.

5. The QoA mapper determines the location of replicas, i.e., servers on which the replicas will be placed.
6. According to the QoA values achieved at the placement the process of calculating replica numbers can be repeated.
7. The final placement is selected.
8. Based on the placement, the CDRS provider supplies service availability to the user.
9. 'Goodness' of the placement is evaluated. It can be identified either by the user or by the CDRS provider whether the achieved QoA is at least at the level the user required.
10. The evaluation result is forwarded to the QoA mapper which uses this information to improve the replica placement.

3.6 QoA Metrics and Parameters

The 'goodness' achieved by a content distribution and replication system, i.e., whether the QoA requirement specified by client or content provider is fulfilled can be identified by evaluating the reached QoA.

Parameter	Notation	Definition	Example value
Related to satisfiedQoA	satisfiedQoA $QoA_{sat}(v)$	for each demanding node[a], how much the availability is fulfilled at the selected placement	0.95, 1.05[b]
		the ratio of supplied availability to demanding availability for node v with $\forall v \in V \backslash R$	
	minSatQoA QoA_{min}	minimum of all satisfiedQoA for all demanding nodes at the selected placement	0.9
		$\min \{ QoA_{sat}(v) : \forall v \in V \backslash R \}$	
	avgSatQoA QoA_{avg}	average of all satisfiedQoA for all demanding nodes at the selected placement	0.95
		$1/n(\sum QoA_{sat}(v))$, $v \in V \backslash R$ where $n = (\|V\| - \|R\|)$	
Related to guaranteedQoA	guaranteedQoA $QoA_{gua}(v)$	for each demanding node v, fulfilled (1) or not (0)	1 or 0
		availability guarantee: $A_R(v) = 1$, if $QoA_{sat}(v) \geq 1$	
	avgGuaQoA QoA_{avgGua}	percentage of fulfilling nodes	0.9
		the ratio of $\|V_{sat}\|$ *to* $\|V\|$, where $\|V_{sat}\|$ = set of nodes with $QoA_{sat}(v) \geq 1$	

Table 3: Notion and definition of the QoA metrics

a. demanding nodes are all the service nodes which are not selected as replica nodes.
b. a QoA value that is higher than 1.0 (100%) is only due to the definition. We keep this definition for the purpose of load balancing between replica nodes; for example, the value 1.05 indicates that the system uses too much resources (number of replicas) for serving the demanding node *v*.

The QoA metrics that will be used in evaluating the QoA fulfillment, throughout this thesis, are as follows:

- *satisfiedQoA* - This metric indicates for each demanding entity (i.e., client or content provider) how much the availability requirement has been fulfilled by the selected replication strategy. For example, the required and satisfied availability values are 95% and 94%, respectively. Then, the *satisfiedQoA* is 0.99.
- *minSatQoA, avgSatQoA* - These metrics are the minimum and the average value of the *satisfiedQoA* for all demanding entities with the selected placement *R*, respectively.
- *guaranteedQoA* - This metric is a form of 'binary' QoA, i.e., the value is either 1 or 0. For a given service demanding entity, if the *satisfiedQoA* is greater (or at least equal to) than the required one, then the *guaranteedQoA* is 1 else 0.
- *avgGuaQoA* - This metric is the average value of the *guaranteedQoA*.

Table 3 shows the notion and definitions of these QoA metrics. *V* is a node set of a graph G(V,E) which models a content distribution and replication system, *R* is a replica node set (i.e., a selected placement *R*) with $R \subseteq V$. |*V*| and |*R*| are the cardinality of the node sets V and R, respectively.

3.7 Related Work

Our work on QoA concept and framework is based on the key ideas of (a) an availability-centric view on QoS and (b) satisfying different levels of service availability required by individual users in content distribution and replication systems. Even though there are many research efforts in the area of QoS and availability, the problem of satisfying different levels of service availability requirements has so far been seldom mentioned.

There is some work that deals with techniques for building QoS-enhanced distributed service systems. AQuA presented in [CSR+98, Ren01] is a framework for providing adaptive fault tolerance to distributed CORBA [Gro98] applications. In AQuA, fault tolerance is achieved by replication of distributed objects. While AQuA includes different types of replication schemes, both in active and passive replication models, which also ensure strong data consistency between replicas, it does not provide any mechanisms for service differentiation and availability guarantees.

In [Chu99] stor-serv introduces the concept of QoS to the network storage domain to support distributed network storage services with multiple service classes. It stor-serv treats caching and replication as different QoS classes, e.g., caching as a best-effort service and replication as a guaranteed service. Even though the approach of stor-serv is similar to our QoA approach, the goals and metrics are different: while our QoA work focuses on satisfying different targets of application-specific service availability requirements, stor-serv focuses on providing performance guarantees. Furthermore, stor-serv does not explicitly address the service availability requirements and, thus, takes only data availability as an additional performance parameter into account.

[Vah02] presents issues on automatically provisioning large-scale distributed network resources to meet target levels of performance, availability and data quality. In a subsequent

work [BKR+02], the author proposes an overlay peer utility model (Opus) which is based on the Active Networks [Wet99] concept and uses an economic model to determine per-application priority levels and constructs a service overlay for each application. Opus details the mechanism of constructing cost efficient, reliable overlays among competing applications. However, it does not detail the service specification semantics nor the metrics needed to quantify reached availability or performance.

Regarding the system availability of widely distributed systems, there are some measurement studies. [DCGN03] deals with the availability of service delivery across wide area networks. It develops a network unavailability model for coping especially with connectivity failures. Based on a set of HTTP service traces, the authors analyzed the average availability, duration of unavailability and failure location. [SGG02] studies characteristics of host availability in the Napster and Gnutella peer-to-peer file sharing systems. By actively probing TCP/IP addresses, it identifies a set of heterogeneity in both systems, e.g., only 20% of the hosts in each system have an IP-level system uptime of 93% or more. A related study is [BMSV02], which deals with host availability of peer-to-peer storage systems. It empirically characterizes the availability of a large peer-to-peer system and measures host turnover in the system. The measurement results from [SGG02, BMSV02, DCGN03] can be used to build a realistic CDRS model which will be used as a base in our service availability study.

3.8 Summary

In this chapter we have investigated a concept and mechanisms, which support content distribution and replication systems for satisfying different levels of service requirements with availability guarantees.

We have developed a novel concept *quality of availability* that enables one to treat availability as a controllable, observable QoS parameter. After a definition of QoA and its service types, we have refined, to enable a more quantitative specification and evaluation of the service availability, the basic availability definition in the forms, *decoupled*, *differentiated*, and *fine-grained*. For a technical realization of the QoA concept, we have also developed a QoA framework which comprises of the four modules, i.e., service specification, QoA mapping, admission control, and replica management. We have shown how users can specify their service requirements in terms of QoA, how the requirements are mapped into the low-level replication specification, and how the QoA guarantee can be controlled. We have defined both the QoA service specification semantic and the QoA metrics and given an example of high-level QoA requirement specification.

Chapter 4 - Static Replica Placement

Through the QoA framework presented in the previous chapter, content providers or clients can specify their application-specific availability requirement which is mapped, by the QoA mapper, into the replication-level resource requirements. Given a QoA specification and a network topology, it is then determined which data should be replicated, how many replicas for a particular data unit should be created, and where the replicas should be placed. We refer to this mapping problem as *replica placement problem*. The replica placement problem can be further divided into two domains depending on the replica creation time and target of replicas: static replica placement and dynamic data[1] replica placement. Static replica placement considers the issue of placing replica servers on the given network, while the dynamic data replica placement deals with the topic of on which replica server and how many replicas to dynamically place for a particular data. In this chapter, we tackle the static replica placement problem specifically and study the effects of number and location of replicas on the achieved service availability. The dynamic data replica placement problem will be studied intensively in Chapter 5. Within the static replica placement problem, we take the two main issues into account: (1) finding a "good" placement for a fixed number of replica servers and (2) determining the number and location of replica for satisfying all QoA requirements. For each of these static replica placement issues, we review existing related work for algorithms ranging from heuristic to exact methods. We also devise new algorithms and evaluate their achieved QoA. Based on a simulation study, we find that the location of replica servers is a more relevant factor than their number for satisfying the QoA requirements by different users, and that heuristic methods, in general, cannot give any guarantee for their achieved QoA, even though they are very efficient for large size graphs.

4.1 Motivation

Important decision issues for the QoA mapper, as well as for the replication management system in general, are:

- *what to replicate?* (data replica selection),
- *how many to replicate for a particular data unit?* (number of replica selection)
- *where to place the replica?* (replica server selection)

1. As mentioned in Chapter 2, the terms data, content, object and file are used interchangeably, when discussing the replica placement problem throughout the work in this thesis.

In this thesis, these decision issues are referred as replica placement problem. In this chapter, we tackle this placement problem, specifically the static case (see Section 2.3), and study the effects of number and location of replicas on the achieved service availability. More specifically, we consider the following scenario. A popular content provider (via Web site) aims to improve its overall service availability and performance by moving its content to some hosting services, i.e., content distribution network (CDN) companies such as Akamai [Aka04], Digital Island [Dig01], Mirror Image [Mir04], Cidera [Cid01], etc. Thus, the content provider makes an agreement with a CDN company which operates content servers that are typically placed near to the users. Assuming full replication, the whole contents of the content provider are completely replicated[2] to each of the CDN's content servers. Within the static replica placement problem domain and under this assumption, we concentrate specifically on the server replica placement problem. Indeed, the static replica placement problem is to choose k replica servers among N total content servers (k > N) so that some objective function is optimized under a given network topology and service access pattern. In our study on the static replica placement problem, there are two objective functions:

- Improving QoA - it is about finding k replica servers and assigning m clients (or service demanding nodes) to them so that (a) the minimum service availability for a client which is allocated to any one replica server is maximum, or (b) the average service availability supplied from the replica servers to all clients is maximum.
- Guaranteeing QoA - in contrast to the case of improving QoA, the goal is to satisfy the service availability requirement for all the demanding nodes with a QoA guarantee.

Concerning guaranteeing QoA, we perform two different simulative experiments in this chapter. The first one is to check the placement selected for improving QoA and whether it also meets all the target levels of the availability requirements for all users. If this is the case, then the placement can further be optimized so that the number of replica servers is minimized. The second experiment is selecting exactly the optimum, i.e., the minimum number and location of replica servers, while offering the QoA guarantee.

We model a content distribution and replication system (CDRS) as a stochastic graph [LM97] in which each node (content server) and link (connection link between two content servers) are parameterized, statistically independently of each other, with known failure statistics.

We develop several placement algorithms for both of the objective functions. To improve QoA, we take ranking-based heuristics [Pea84, KM02] which calculate the supplied availability for all content server nodes and select the nodes with higher (or highest) availability values as replica nodes on which the replicas are placed. To guarantee QoA, we develop an exact method called *state enumeration* which enumerates all possible placements without skipping any solution.

We evaluate the 'goodness' of the various placement algorithms by simulating their behavior on Internet-like topologies. Based upon our results, we conclude that (1) the location of replicas is a relevant factor for the service availability of a CDRS, (2) even simple heuristics

2. Replicas created in this way are called complete replicas. Mirror servers (or simply mirrors) are a typical example for complete replicas.

can achieve reasonably high service availability, but, they cannot give any guarantee for their achieved availability, and (3) the *state enumeration* algorithm guarantees the service availability with its placement results, but the run-time complexity is very high.

4.2 Outline

The rest of this chapter is organized as follows. In Section 4.3, we discuss related work. We describe the system model and basic notations used throughout this chapter in Section 4.4. Then in Section 4.5 and Section 4.6, we describe graph theoretic formulations of the static replica placement problem and present a number of placement algorithms for both improving and guaranteeing QoA. In Section 4.7, we describe our simulation methodology. The placement results are evaluated and discussed in Section 4.8, and Section 4.9 concludes this chapter.

4.3 Related Work

There has been much work on Web and storage server performance, ranging from Web workload characterization [PF95, BCF+99, PQ00, BW01, SWCK02], closest server selection [FAZ99, ST00, VTF01, WPP02], to developing techniques to enhance those servers' performance and scalability [MDZ99, Bre01, JCDK01, MO02]. Caching and replication are the primary techniques used for enhancing the Web and distributed storage services. Previous work has studied many aspects of caching and replication, such as cache replacement [Wan99, CI97, PB03], content distribution [Gri00, BRL01, BDK+02], inter-replica or inter-proxy-cache communication [WC97, BCD+01, BCG+02, Mau02], cooperative caching [DWAP94, FCAB00, DKK+01, Zin03], caching infrastructure [RA99, PvS01, CFK+01, MCH+01, IRD02] and update management [GS96, Nin01, NKS+02, CO02]. However, less attention has been given to the placement of the content server replicas in wide-area internetworks.

[LGID99] investigated the proxy placement problem with the assumption that the underlying network topologies are trees, and modeled it as a dynamic programming problem. However, the Internet topology is not a tree. Furthermore, the paper does not evaluate how well their dynamic programming algorithm based on tree topologies works for Internet-like topologies. In contrast to that, [QPV01] studied the server replica placement problem in content distribution networks (CDNs) with both synthetic and real network topologies, as well as Web server traces. The authors formulate the server replica placement problem as a minimum k-median graph theoretic problem and propose a number of heuristics that minimize the cost for clients to access contents replicated on the servers. The algorithms use workload information, such as client latency and request rates, to make informed placement decisions. Through a simulation study, the paper shows that a greedy algorithm for Web server replica placement can provide CDNs with performance that is close to optimal.

Regarding placement algorithms, the two well-known graph theoretic algorithms are min k-center [Bar92, Bar96] and k-HST [Vaz01]. [JJJ+00] uses them to determine the number and placement of instrumentation boxes for the purpose of network measurement. The authors use nearest mirror selection as a motivating problem, the three mirrors they consider are manually placed on arbitrarily selected locations. For a larger network size, [JJK+01] investigate the mirror placement problem on the Internet under a realistic setting where the number of mirrors is

still small, but generally larger than three, and the placement is restricted to a given set of hosts. They show that increasing the number of mirror sites under the constraint is effective in reducing client download time and reducing server load only for a surprisingly small range of values regardless of the mirror placement algorithm.

None of these papers addresses the issue of replica placement for satisfying or guaranteeing service availability.

4.4 Definition of the Static Replica Placement Problem

This section provides the basic assumptions, the system model and the notations used throughout this chapter.

4.4.1 Basic Assumptions

The target content distribution and replication system considered in this chapter is a content repository (e.g., CDN or Web mirrors) consisting of a set of content servers (hereafter called *nodes*) interconnected with each other in the Internet. As network topology for the content repository, we use simulated topologies which closely resemble realistic Internet topologies. For this purpose, we make use of a number of (power-law) random graph generator as used by [TGJ+01, RHKS02, HPSS03] and, additionally, empirical Internet measurement data sets on different topologies from the National Laboratory for Applied Network Research (NLANR) [NLA02].

Further abstractions of our target system and assumptions are as follows:

- All nodes have the same service and storage capacity. Thus they can hold the same amount of content.
- While the levels of the service availability requirement are different among the users, all nodes receive the same workload, i.e., have an identical rate of content access requests from the users.
- The failure probability values between nodes, as well as between links are different, and independent of each other.
- The content access pattern is identical at each node.
- All the contents are accessed in ‘read-only’ mode by clients.
- Update of the contents is done only by the content providers and the update frequency is very low in comparison to the read access.
- If a node is selected as a replica node, then it is a complete replica (i.e., full replication).
- System and service information such as the network topology and condition, resource availability, access pattern and failure statistics are globally available.
- Each user (client) uses a single replica node, when accessing a content, while multiple clients can use the same replica node. In other words, a client gets all the contents from only the same replica node.

While the abstractions of the system and the assumptions do not capture all aspects of real systems, we are confident that they do capture the essential features needed to understand the relevant factors for satisfying service availability and the qualitative differences between the placement algorithms that we develop.

4.4.2 System Model

We model a content repository as a graph, $G(V,E)$, where V is the set of nodes and $E \subseteq V \times V$ the set of connection links. This graph is *static* if the members and the cardinality of V and E do not change otherwise it is *dynamic*. The graph is said to be *stochastic* when each node and link are parameterized, statistically independently of each other, with known failure or availability probabilities. The static replica placement problem can be distinguished into constrained and unconstrained replica placement cases, according to whether they consider any constraints (e.g., geographical distance or minimum resource capacity). In the constrained replica placement problem, there are a set of service demanding nodes D and a set of service supply nodes S, so that $D \cup S = V$ and $D \cap S = \{\}$, and the replica set R can only be built from the nodes of the set S, $R \subset S$. In the unconstrained replica placement problem, every node can be either a service demanding node or a service supply node, i.e., there is no difference between D and S, and $R \subset V$. Concerning the static replica placement problem, the following sections (Section 4.5 and Section 4.6) introduce the two main placement issues that are explored in this thesis. For both issues, we model the static replica placement problem as a *static, stochastic* and *unconstrained* graph. The overall goal of these two static replica placement cases is then as follows: given a finite number of content servers, we are interested in whether their placement on the Internet affects the overall service availability. More specifically, we are interested in whether larger number of replicas provide, as one can intuitively argue, a further increase in service availability, and further how their placement effects the achieved QoA.

4.4.3 Notations

In the description of the static replica placement problem and the placement algorithms throughout this work, we use the notations presented in Table 4.

Notation	Description
G (V, E)	a graph with V, the set of nodes, and $E \subset V \times V$, the set of links
v	a node in G with $v \in V$
e	a link in G with $e \in E$
D	a set of service demanding nodes (i.e., clients) and a subset of nodes of G, $D \subset V$
S	a set of service supply nodes (i.e., servers) and a subset of nodes of G, $S \subset V$, $D \cap S = \{\}$
R	a set of replica nodes, a subset of nodes of G, $R \subseteq S \wedge R \subset V$
|V|, |D|, |S|, |R|	cardinality of the node sets V, D, S, R, respectively
T(R)	topological placement of R, i.e., the location of R's elements
p_v	availability probability of a node $v \in V$ for service supply

Table 4: The notations.

Notation	Description
q_v	failure probability of a node $v \in V$ for service supply, q_v=1- p_v
$p_{v_{req}}$	required service availability of a node $v \in V$ for service demand
p_e	availability probability of a link $e \in E$ for service supply
q_e	failure probability of a link e for service supply, q_e=1- p_e
x_v	a boolean variable for the state of a node $v \in V$: one (1) if v functions else zero (0)
x_e	a boolean variable for the state of a link $e \in E$: one (1) if e functions else zero (0)
a(G)	service availability of the graph G
g_i	a state of the graph G
$G(g_i)$	partial graph of G associated with g_i
$A_v(R)$	achieved QoA for a node $v \in V$ with R: one (1) if satisfied else zero (0)
$\bar{A}(R)$	achieved QoA for all nodes of G with R: one (1) if satisfied else zero (0)

Table 4: The notations.

4.5 Finding a Good Placement for Improving QoA

This section describes the first static replica placement problem, hereafter called *SRP-M*, which consists of finding a 'good' placement so as to maximize the service availability perceived by clients (demanding nodes). After a formal definition of the SRP-M, we present a number of algorithms for tackling the SRP-M problem.

4.5.1 The Problem Description

The static replica placement problem is an optimization problem [PS82]. In an instance of the static replica placement, i.e., SRP-M, we are given an integer $k > 0$, a (parameterized) stochastic graph G (V,E) where a failure probability value is assigned to each node $v \in V$ as well as each link $e \in E$. An additional parameter $p_{v_{req}}$ which is the demanding service availability value assigned to each node *v*. We can take

F = {all possible placements with $|F| = k$ } and

$QoA_{sat}(f)$ = average satisfied QoA for all nodes of G with a placement f .

The SRP-M problem is to find a placement instance $r \in F$ for which

$$QoA_{sat}(r) \geq QoA_{sat}(h) \quad \forall (h \in F) \tag{7}$$

According to the QoA feature of different service classes, such as deterministic, stochastic and best-effort classes, several degrees of firmness of the satisfied QoA can be defined:

- average satisfied QoA: $QoA_{avg} > \tau_{avg}$
- minimum (worst case) satisfied QoA: $QoA_{min} > \tau_{min}$
- average and minimum satisfied QoA: $QoA_{avg} > \tau_{avg}$ and $QoA_{min} > \tau_{min}$

Thus, the optimization conditions of the SRP-M problem are:

$$QoA_{min}(r) \geq QoA_{min}(h) \quad \forall (h \in F) \tag{8}$$

for minimum satisfied QoA and

$$QoA_{sat}(r) \geq QoA_{sat}(h) \quad \text{and} \quad QoA_{min}(r) \geq QoA_{min}(h) \quad \forall (h \in F) \tag{9}$$

for average and minimum satisfied QoA.

The constraints considered in our SRP-M work are:

- The number-of-replica constraint that limits the number of replicas placed;
- The storage and bandwidth capacity constraints that place an upper bound on the storage capacity of the node and the bandwidth capacity of the link;
- Maximum number of replica for a particular content at a server node;

4.5.2 The Algorithms

The static replica placement problem (also SRP-M) belongs in general to the class of discrete location problems [GJ79, PS82]. Many similar location problems are introduced and algorithms are proposed to solve the problems in this category [MR95, TGJ+01, WNB97, CW99, BSS00, GMM01, JMS02, ST02]. The heuristics, popularly known as rules of thumb [Pea84], such as *Greedy, TransitNode, Vertex substitution*, etc. are applied to many location problems and have shown their efficiency [MH97, JJK+01, KRS00, MLH00]. In this work, we take some basic ranking-based heuristic algorithms [Vaz01, KM02] as follows. Yet, different variants of these heuristics and any such improvement can be used with slight modifications to enhance the efficiency and performance of our basic heuristics:

- *Random*. By using a random generator, we pick a node *v* with uniform probability, but without considering the node's supplying availability value, and put it into the replica set. If the node already exists in the replica set, we pick a new node, until the given number reaches *k*.
- *HighlyAvailableFirst (HA)*. For each node *v*, we calculate *v*'s actual supplying availability value by taking the availability values of all adjacent edges of the node into account. The nodes are then sorted in decreasing order of their actual availability values, and we finally put the best *k* nodes into the replica set.
- *TransitNode (TR)*. The basic principle of the *TransitNode* heuristic is that nodes with the highest (in/out) degrees, i.e., the number of connection links to adjacent nodes, can potentially reach more nodes with smaller latency (or hop count). So we place replicas on nodes of V in descending order of (in/out) degree. This is due to the observation that nodes in the core of the Internet that act as transit points will have the highest (in/out) degrees [JJJ+00, RGE02].
- *HighlyAvailableFirst+TransitNode (COM)*. This method is a combination of the *HF* and *TR* algorithms. Figure 10 details the *COM* algorithm.

Algorithm *COM*

Input: a parameterized graph *G (V,E)*
a number *k >0*;
Output: a replica node set R
Initialization: *R* := {},
candidate node set *candSet* := {}

1. **forall** node v of V
2. **forall** adjacent edges of v,
3. calculate max/min/mean values for supply availability value
4. **forall** nodes v of V
5. build a *candSet* with nodes of which their availability value and degree are greater than the average values, respectively.
6. sort the nodes in *candSet* in decreasing order of the supply availability value
7. for nodes which have the same availability values,
8. sort these nodes in decreasing order of the (in/out) degree
9. **while** (|*R*| <= *k*)
10. put the best *k* nodes from *candSet* into *R*
11. **return** *R*

Figure 10: Pseudo-code of the *COM* algorithm

We applied the *COM* algorithm to an example stochastic graph which represents a model of a overlay network topology[3]. (Figure 11). In this example graph G(V,E) with |V|=10 and |E|=20, nodes and links are parameterized with randomly generated probability values, e.g., the demand and system availability values of nodes are chosen between 90% and 99%, and the failure probability values of links are between 1% and 10%. For a given number $k = 1$, the candidate replica nodes are {4}, {9}, {8}, {3} and {0}. By considering only the supply availability value, the *COM* algorithm selects {4} as the replica set.

4.6 Finding the Optimum with a QoA Guarantee

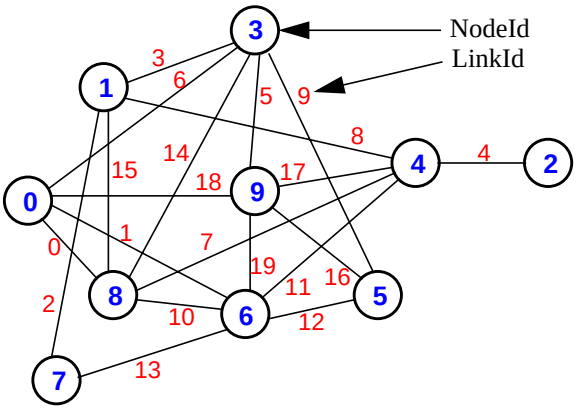
This section describes the second static replica placement problem, hereafter called *SRP-G*. We divide the SRP-G problem into two sub-problems. The first is to check whether the placement selected for improving QoA also meets all the target levels of the availability requirements for all users. The second is selecting exactly the optimum, i.e., the minimum number and location of replica servers, while offering the QoA guarantee for all demanding nodes.

4.6.1 Checking Reached QoA

Given a stochastic graph *G(V,E)* and a replica set *R* with its topological placement T(R). The objective of this problem is to check for all demanding nodes whether the reached availability satisfies the required QoA for them, i.e., whether $\bar{A}(R)$ is 1 or 0. In comparison to the SRP-M problem, this problem requires a solution which exactly tests whether the result is 1 or 0. Similar works are introduced in the literature [LM97], which are devoted to the problem of *net-*

3. In this network, each node is logically connected (for example, via a TCP connection) to a small subset of the other nodes, i.e., to the node's neighbors, to form an overlay network [RHKS02].

work reliability. The methods that provide an exact reliability are called exact methods, in contrast to the heuristic methods which provide an approximate result.



Node parameter assignment: the demand and node availability values for all nodes are independent and decoupled. The values are in percentage (%).

NodeId	0	1	2	3	4	5	6	7	8	9
Demand availability (%)	94	92	91	92	93	94	93	92	95	95
Node availability (%)	96	95	91	96	98	94	95	92	96	97

Link parameter assignment: to each edge a failure probability value is assigned.

LinkId	0	1	2	3	4	5	6	7	8	9
Failure Probability (%)	3	9	1	6	3	1	2	4	4	2

LinkId	10	11	12	13	14	15	16	17	18	19
Failure Probability (%)	6	3	8	1	3	7	5	2	4	8

Figure 11: An example stochastic graph G(V,E) with |V| = 10 and |E| = 20.

Enumerating all possibilities without skipping any solution case requires exact methods for solving this problem. From some exact methods which are proposed in the literature, we adopted the state enumeration method [LM97] and modified it for our problem. In the state enumeration method, the state of each node and each edge are enumerated: the state value is either *1* when it functions or *0* when it fails. Indeed, there are $2^{|V|+|E|}$ states for a graph *G = (V,E)*, i.e., $2^{|V|+|E|}$ partial graphs for G. We then check the QoA for all partial graphs with the replica set given as input. Figure 12 details the *StateEnumeration* algorithm.

In order to show how the *StateEnumeration* algorithm works, we applied it to an example stochastic graph G(V,E) with |V|=5 and |E|=5, placement R={2}, |R|=1, as shown in Figure 13. As the test node, we take the node 0 which has the availability requirement value 97%. Figure 13 (right) presents the state matrix, the availability and the sum of availability probability value for each partial graph. In this example, we only enumerate the state of nodes to reduce

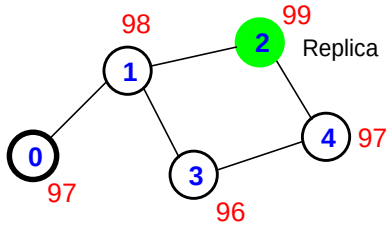
the run-time complexity, i.e., from $2^{|V|+|E|}$ to $2^{|V|}$. The first column means number of the partial graphs to be tested. Instead of considering all partial graph cases of $2^{|V|+|E|}$, we can only get half of them by skipping the cases in which the node 1 is 'not available', i.e., (the state value of the node 1 is 0), because the node state 'zero' of node 1 causes no further possible connection for the test node 0 to build any path to the replica node 2.

Algorithm StateEnumeration

Input: a parameterized graph G (V,E)
a placement (i.e., replica set R);
Output: the QoA: either 1 (satisfied) or 0 (not satisfied)
Initialize variables: state matrix

1. **forall** nodes v of V
2. build a state matrix for all partial graphs of G
3. **forall** states
4. check the availability of the state $a\langle G\rangle$, i.e., whether there is at least one path
5. for the states for which the checked availability value is 1,
6. compute availability probability $pa\langle g_i\rangle$ for each state
7. sum all the availability probability values of satisfied states
8. **if** p_v <= $A_v\langle G\rangle$ **then** $A_v\langle R\rangle$ =1
9. **forall** nodes v,
10. **if** $A_v\langle R\rangle$ =1
11. **then** $\bar{A}(R)$ = 1, i.e., the QoA of G with R is satisfied
12. **else** $\bar{A}(R)$ = 0
13. **return** $\bar{A}(R)$

Figure 12: Description of *StateEnumeration* algorithm



m	StateMatrix	$a\langle G\rangle$	$pa\langle g_i\rangle$
1	1 1 1 1	1	0,98 * 0,99 * 0,96 * 0,97
2	1 1 1 0	1	0,98 * 0,99 * 0,96 * 0,03
3	1 1 0 1	1	0,98 * 0,99 * 0,04 * 0,97
4	1 1 0 0	1	0,98 * 0,99 * 0,04 * 0,03
5	1 0 1 1	0	/
6	1 0 1 0	0	/
7	1 0 0 1	0	/
8	1 0 0 0	0	/

Figure 13: An example graph G=(V,E), |V|=5, |E|=5: only the node states are enumerated where the nodeId 1's state is one (1)

After building a state matrix for the nodes 1, 2, 3 and 4 at the second column, we check whether there is any path between the node 0 (test node) and the node 2 (replica node) at the third column. According to the result of this check, we calculate the availability probability values $pa\langle g_i \rangle$ for each partial graph of which the availability check value is 1: as shown in the fourth column, we calculate the $pa\langle g_i \rangle$ just for the first 4 partial graphs (m = 1, 2, 3 and 4). The summation of the availability probability values of the four satisfied states is: $A_v\langle G \rangle = pa\langle g_1 \rangle + pa\langle g_2 \rangle + pa\langle g_3 \rangle + pa\langle g_4 \rangle = 0{,}97135624$, and this availability value is greater than the availability value required by the node 0. Indeed, the QoA for the node 0 is: $A_v\langle R \rangle = 1$. The QoA for the rest test nodes can be calculated by repeating this aforementioned process of the *StateEnumeration* algorithm. In the next section, we show extensively the effectiveness of this algorithm on the reached availability through a simulation study.

4.6.2 Finding the Optimum

This section describes the second SRP-G problem. As input, only a stochastic graph *G(V,E)* is given. It has to be determined how many replicas must be deployed and where these replicas should be placed to guarantee the required QoA.

Satisfying a certain, required QoA value with a guarantee means that we always have to offer a replica set which fulfils the given QoA requirements. Heuristics are no proper approaches for solving this problem, because they do not always give a solution with QoA guarantee. To solve this problem, we generally can use the exact methods, one possible case may be the enumeration method which is described in the previous section. For solving this problem, we need to call the *StateEnumeration* algorithm $2^{|V|}$ times, i.e., for all the possible replica solution sets. The run-time complexity of this algorithm is then $O(2^{2^{|V|}+|E|})$.

4.7 Simulation

In this section, we describe the methodology and parameter settings for our simulation study. It is actually impossible to take all real characteristics of an Internet-based content distribution and replication system into account. Thus, the goal of our simulation study is not to resolve small quantitative disparities between different placement algorithms, but instead to reveal fundamental qualitative differences between them and, as a consequence, to understand the effects of the reached different placements on the service availability.

4.7.1 Simulation Methodology

We built an experimental environment to perform a simulation study for the two static replica placement problems, SRP-M and SRP-G. Our goal in conducting an availability evaluation is to study the effect of changing the number ($|R|$) and location ($T(R)$) of replica servers on the reached QoA. A further goal is to determine exactly the replica server set R for given QoA requirements.

For our availability evaluation, we conducted simulations on a number of Internet-like network topology [ZCB96, ZCD97, RHKS02, LBCM03, HPSS03, SFFF03]. By using LEDA graphic library [LED02] and TIERS topology generator [TIE01] several random topologies in different sizes can be generated at run time. To most closely resemble the actual Internet topol-

ogy, we used empirical data sets on different network topologies from NLANR [NLA02]. These data sets describe the Autonomous System topology on a different day. Based on the data sets, we also associated in/out-degree of the Autonomous System with each node of the randomly generated graphs.

However, since we had no access to any real data concerning the availability or the failure parameters, we simply assigned a random availability/failure value to each node and each edge in the topology. For example, we took the values from 50, 70, and 90 to 99% as nodes' availability value and from 10 or 5 to 1% as links' failure probability value. These availability/failure assignments are probably quite misleading, since the true Internet failure values are not random; at the very least, they usually obey the triangle inequality.

We decoupled the availability values between the demanding and supply nodes, i.e., all nodes have two availability parameters assigned: one value as the demanding availability parameter and the other as the supplying. Thus, when a node is a demanding node (client), then its demanding availability value is used, while for a supplying node the actual supplying availability value is, for example, calculated by multiplying the availability values of its own and the average availability value of its adjacent edges. At the phase of building replica sets, each node is evaluated according to its calculated supplying availability value. Node degree information of the test graphs used in our simulation study are shown in Table 5.

random graph	graph size (V, E)	average degree	maximum degree	average node failure probability
G1	(100, 200)	4.7	10	0.045700
G2	(1000, 3000)	6.4	15	0.047200
G3	(10000, 50000)	10.1	26	0.045946

Table 5: Test graphs

As a replication model, we assume *static* and *full replication* in which the whole data items of an origin server system are replicated to other nodes located within the same network prior to starting service. The simulation program is written in C/C++ and runs on Linux and Sun Solaris machines.

4.8 Evaluation of Reached QoA

In this section, we present an analysis of the service availability of both static server replica placements (SRP-M and SRP-G). We evaluated the reached QoA of our heuristics and the exact enumeration method using topologies of different sizes. We ran each basic heuristic and the exact state enumeration method on each topology using different value ranges for the availability and failure probability parameters of nodes and edges. The demanding and initial supplying availability values of the nodes and the failure probability values of the edges are assigned randomly, from a uniform distribution where we varied the parameter values. Statistics for the SRP-M are collected from 20 simulation runs for each placement algorithm, which ensures reasonably small standard deviation in our results. To evaluate the QoA offered by our

heuristics and the state enumeration algorithm, we used the QoA metrics defined in Section 3.6.

4.8.1 Effects of |R| and T(R) on Reached QoA

We first evaluate the reached QoA by our simple heuristics. The baseline for our experiment is a random (*Random*) placement which is obtained by randomly selecting *k* nodes from *V*. We then compare the reached QoA of each heuristic to this baseline and present the relative QoA improvement obtained with each heuristic.

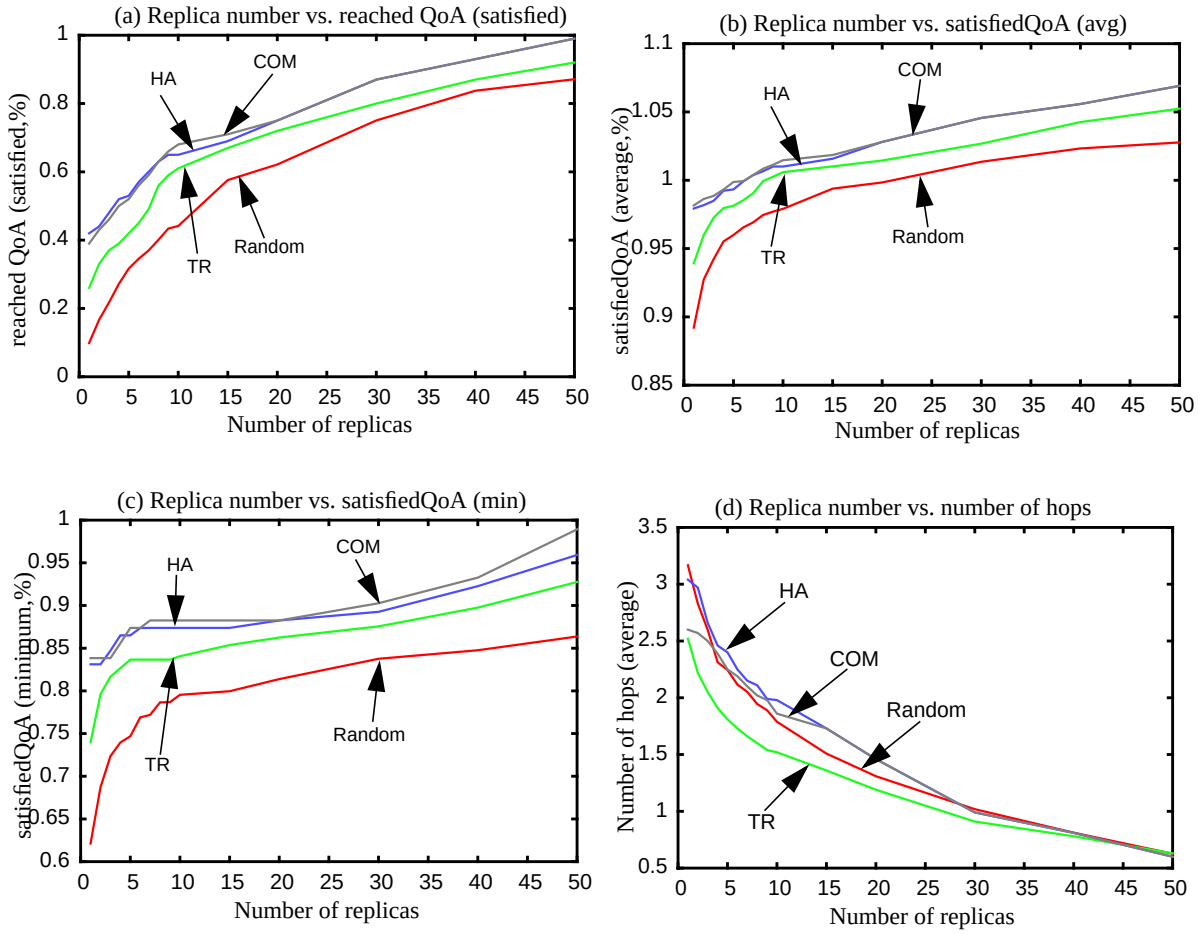


Figure 14: Comparison of QoA values reached by our heuristics with the graph *G1*.

The first experiment examines how the number of replicas affects the reached QoA. For this purpose, we fixed the nodes' supply availability as well as links' failure probability. We increased the number of replicas, from 1 to 50 replicas for the graph *G1* (|V| = 100, |E| = 200). We assumed that there is no constraint on the topological location of the replicas, i.e., *G1* is a unconstrained graph and *S* = *V*, and that replicas may be placed at any node *v* in *G1*.

Figure 14 shows the results from this experiment with *G1*. We plot the number of replicas on the x-axis and the reached QoA on the y-axis. In each graph, we plot different curves for different heuristics. Figure 14, (a), (b) and (c) show the percentage of satisfied QoA with a guarantee (the reached QoA is greater than 1.0), in average and minimum, as the replica number increases. We can see from the figures that our heuristics *HA*, *TR* and *COM*, though simple,

reach higher QoA in comparison to the baseline *Random* placement. For instance, with the number of replicas between 5 and 10, the improvement of satisfied QoA rate with a guarantee is about 20% (Figure 14 (a)). Even though this improvement may not seem much, it is important to note that the number of replicas is really a relevant factor for improving QoA: the larger the replica number is, the better is the reached QoA.

4.8.2 Effects of Varying Availability Requirement Value Ranges on QoA

In the second experiment, we examined the effects of different ranges of the required availability values on the reached QoA. The range values have been varied from 50~99% to 70~99% and 90-99% for the same graph *G1*, where the values in each range are distributed uniformly. As a placement algorithm, the *HA* has been used. From Figure 15 (a) and (d), we can see that the improvement rate of the satisfied QoA value with a guarantee and the average number of hops do not depend on the range size of the availability requirement values. However, it is noticeable, as Figure 15 (b) and (c) show, that the percentage gap between the average and minimum satisfied QoA values was bigger the larger the range size.

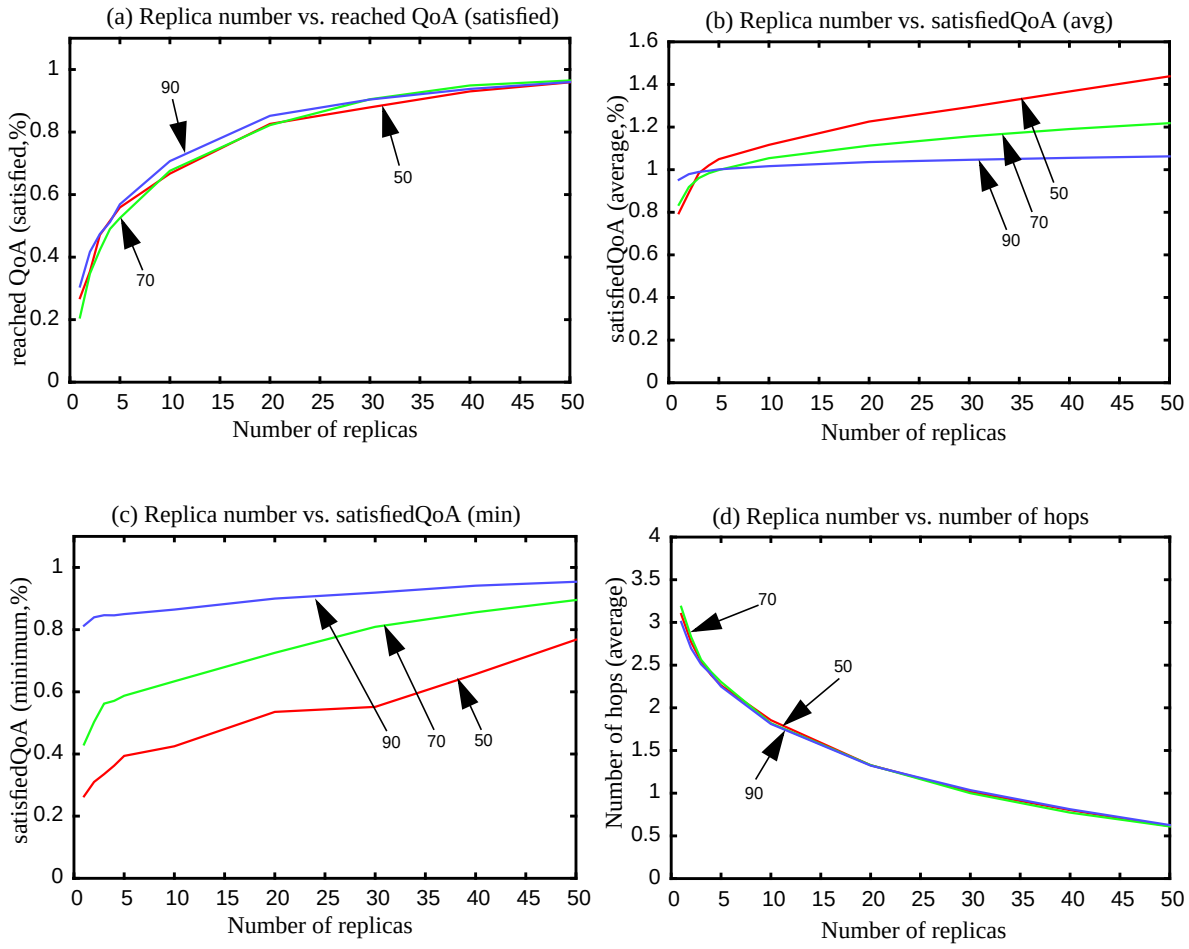


Figure 15: Comparison of reached QoA values with varying values of demand availability.

4.8.3 Effects of Varying Graph Size on QoA

In the third experiment, we took different sizes for the test graphs to study whether the improvement rate can vary, and whether it depends on the graph size. We fixed the node's supply availability, link's failure probability and the placement algorithm (*HA*). As test graphs, G1, G2 and G3 graphs are used (see Table 5). We increased the replication ratio[4], from 0.01 to 0.5.

In Figure 16, we plot the replication ratio on the x-axis and the reached QoA on the y-axis. In each graph, we plot different curves for different graphs. From Figure 16, we can see that, even with the same placement algorithm (*HA*), the improvement rate is different between the graphs: the larger the graph size is, the higher the improvement of reached QoA rate with the same replication ratios. For instance, with a replication ratio 0.1 the satisfied QoA values for G1, G2 and G3 are 68%, 85% and 92%, respectively, as Figure 16 (a) shows.

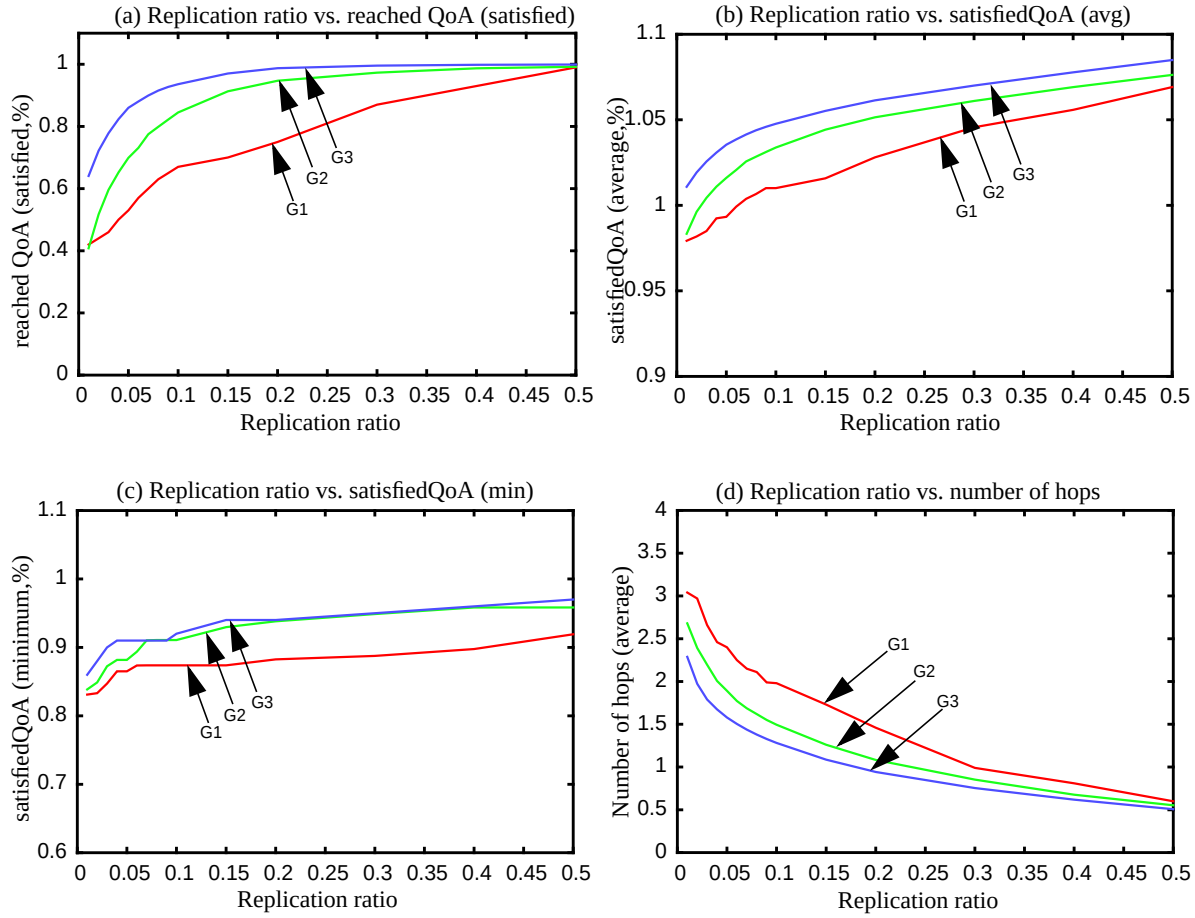


Figure 16: Comparison of reached QoA values with different size of test graphs.

4.8.4 Exact Evaluation of Reached QoA by Heuristics

In this section, we evaluate the QoA reached by our heuristics in an exact form and check whether the reached QoA can really satisfy the required QoA for all demanding nodes. Addi-

4. Replication ratio means the ratio of nodes that have the replica to the total nodes. For example, a replication ratio 0.5 means that replicas are placed on the 50% of total nodes.

tionally, we test also how many replica nodes the heuristics need to give a QoA guarantee. For this purpose we ran our *StateEnumeration* algorithm with replica sets produced by our heuristics *HA* and *COM* as input. Due to the exponentially growing run-time complexity and the memory requirements with growing graph sizes, we limited our experiments for the *StateEnumeration* to a small graph *G0* with |V| = 20 and |E| = 30. The demand availability values are between 90 and 99% and the link failure probability value is 0%.

Table 6 shows the detailed test result from *COM*. For the calculation of the average and minimum satisfied QoA, we excluded the QoA values for replica nodes. Even though the *COM* algorithm could reach an average satisfied QoA (1.0118) greater than 1.0 with one replica node, it could not offer the QoA guarantee: for this small graph, 10 replicas were needed to satisfy all the QoA requirements with a guarantee.

replica numbers	Replica locations, *T(R)*	satisfied QoA (avg)	satisfied QoA (min)	guaranteed QoA (avg) by exact test
1	8	1.0118	0.9100	0.80
2	8,10	1.0194	0.9100	0.90
3	8,10,12	1.0226	0.9193	0.90
4	8,10,12,11	1.0355	0.9496	0.95
5	8,10,12,11,13	1.0399	0.9496	0.95
6	8,10,12,11,13,0	1.0487	0.9591	0.95
7	8,10,12,11,13,0,16	1.0556	0.9900	1.00
8	8,10,12,11,13,0,16,1	1.0577	0.9900	1.00
9	8,10,12,11,13,0,16,1,2	1.0610	0.9900	1.00
10	8,10,12,11,13,0,16,1,2,5	1.0711	1.0000	1.00

Table 6: A detailed result for *COM* with a graph G(20,30)

4.8.5 Finding the Optimum - |*R*| and *T(R)*

In the last experiment, we considered the case of finding the optimum, i.e., the minimal number of replicas and their geographical placement which satisfies the service availability with guarantee. We re-used the *StateEnumeration* algorithm and the test graph *G0* with the same values for availability and failure probability parameters. We started the routine with a replica degree of 1, i.e., $k=|R| = 1$, and selected each node as replica node. We then incremented the replica degree, until we reached the guaranteed QoA = 1.0 (a QoA with guarantee). Figure 17 plots the reached QoA that *StateEnumeration* algorithm calculated exactly with each instance for the given *k*. The y-axis is the average guaranteed QoA in percentage, while the x-axis is the number of instances[5] of the placement with a fixed replica degree *k*. For k=1, the gap between the best (80%) and worst QoA (10%) values is 70%.

5. The number of instances for a placement is: $\binom{n}{k}$ where *n* is the total node number and *k* is the replica number.

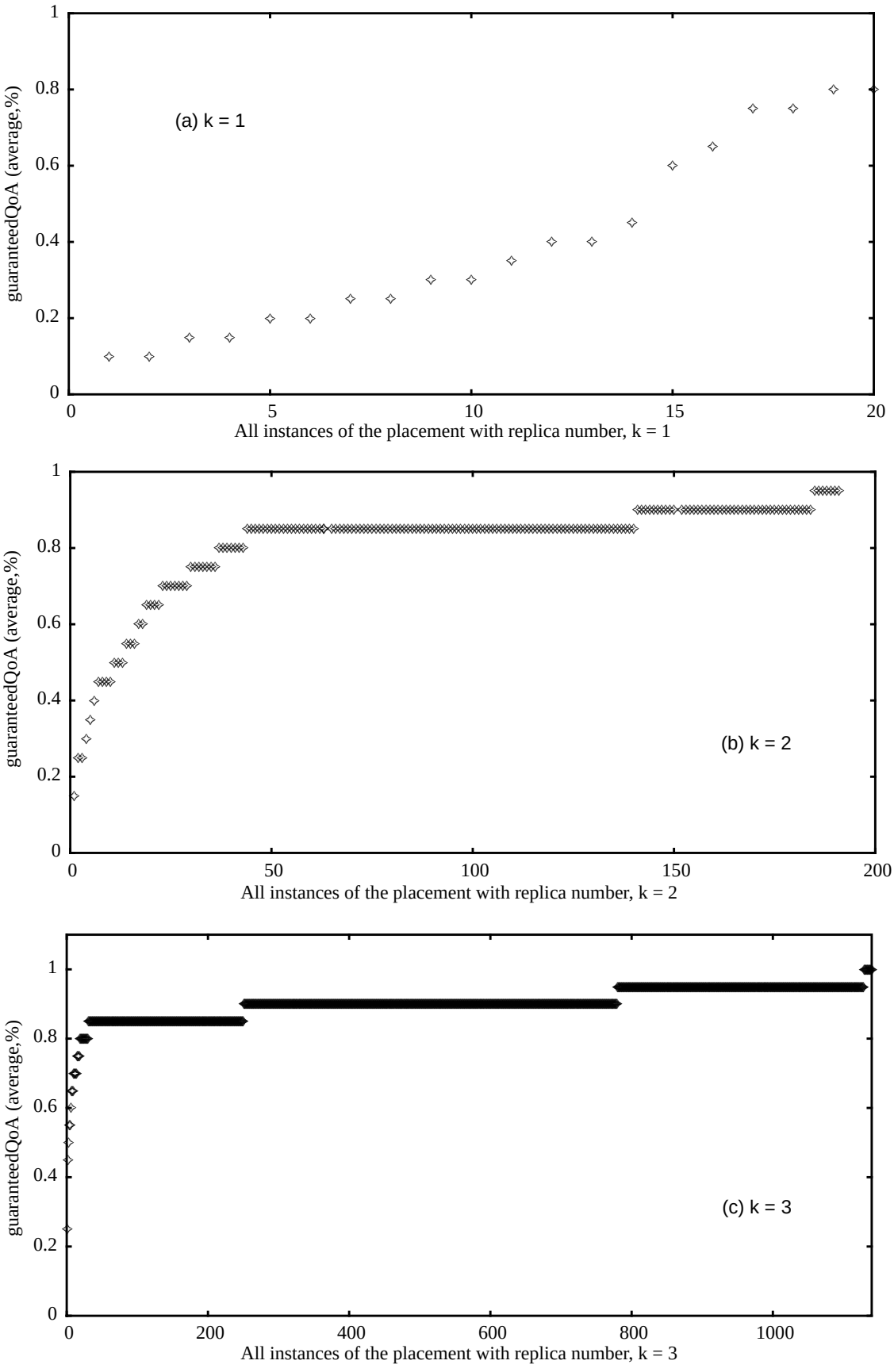


Figure 17: Effects of different replica locations on reached QoA: checked exactly by using the *StateEnumeration* algorithm.

For k=2, the QoA gap is bigger: with k=2, the best QoA value is 95%, while the worst one is 15%. Thus, the gap is 80%. Moreover, from the results of (a) and (b), we can see that some placement instances with k=1 can offer even higher QoA than those with k=2. For example, for a reached QoA value of 60%, while 4 placement instances with k=1 satisfy the target QoA value, there are 20 placement instances with k=2, which reached a QoA value less than 60%. A similar effect can be observed for k=3. As Figure 17-(c) shows, the gap between the best (100%) and worst QoA (25%) values is 75%.

From the results of this experiment, we can conclude that the location of replicas is a more relevant factor than the number of replicas for satisfying the service availability. The results further indicate the importance of investigating good placement algorithms, which often offer a better placement than the *Random*.

4.9 Conclusion

In this chapter, we investigated the static replica placement and studied the effects of number and location of replicas on the achieved service availability. We divided the static replica placement problem into two sub-problems: (a) finding a "good" placement for a fixed number of replica servers and (b) determining the number and location of replica servers for satisfying all demanding QoA requirements.

We have modeled content distribution and replication systems as a stochastic graph in which each node and link are parameterized, statistically independently of each other, with known failure statistics. We have developed several placement algorithms for both the two objective functions and evaluated the 'goodness' of the proposed algorithms by simulating their behavior on Internet-like network topologies.

Through simulations and modeling studies, we have learned that:

- The location of replicas is a relevant factor for the service availability. Even though the QoA improvement could be achieved by increasing replica numbers, replicas' placement and their dependability affected the QoA far more significantly.
- Using a heuristic method is more efficient than the exact method, at least in terms of the runtime complexity, to find a good placement for large graphs. However, the replica ratio of their placement results is in most cases higher than those of the exact method. Furthermore, the heuristics give no guarantee for the QoA.
- In comparison to the heuristic method, the exact method guarantees the QoA with its placement results, although the runtime complexity is very high: $O(|V_R| \cdot 2^{|V|+|E|})$ and $O(2^{|V|} \cdot |V_R| \cdot 2^{|V|+|E|})$ for the availability checking and the guaranteed QoA problems, respectively.

Chapter 5 - Dynamic Replica Placement in Peer-to-Peer Systems

This chapter examines dynamic content replica placement from the point of view of peer-to-peer file sharing systems. In the context of this thesis, peer-to-peer systems are characterized as being decentralized, self-organizing distributed systems, in which most communication is symmetric. In a peer-to-peer system, there is no peer which is permanently serving other peers. Instead, each peer is a host that intermittently connects to the system to download/upload contents and donates a fraction of its disk storage and access bandwidth to the system. Furthermore, the peer holds a portion of contents that may be accessed by other peers. We argue that the service availability of such a peer-to-peer system can be significantly enhanced if the system carefully replicates its contents. We investigate specifically dynamic content replication for availability, where our goal is choosing dynamically the number and location of replicas to satisfy the service availability requirement for all individual peers, while taking intermittent connectivity of peers explicitly into account. We provide a number of fully decentralized, ranking-based heuristics which work in a cooperative way to replicate contents on-the-fly. Through an event-driven simulation study, we analyse the 'goodness' of the placements provided by the used heuristics and further identify the relevant factors for improving service availability in the domain of data replica placement. Our proposed replication and simulation model can be used for further study on the dual availability and performance QoS for dynamically changing, large-scale peer-to-peer systems, as well as on the replica placement for QoA guarantees.

5.1 Motivation

The rapid popularization of Internet-based peer-to-peer (P2P) applications such as Napster [Nap02], Gnutella [Gnu04], FastTrack [Fas02], and KaZaA [Kaz04] has inspired the research and development of technologies for P2P services and systems. While much of the attention has been focused on the issues of providing scalability, copyright solutions or routing mechanisms within P2P networks, the availability issue has so far seldom been mentioned, and there is no work known to us which tries to satisfy and guarantee the availability requirements for all individual peers.

In this chapter, we present a study of QoS-controlled dynamic content replication where our goal is to choose dynamically the number and location of replicas for a particular content. This is to satisfy the QoA requirements for all individual peers, while taking intermittent connectivity of peers explicitly into account. In particular, the main focus of our work is building a model and devising mechanisms to study the problem of how to satisfy *different* availability

requirements for distributed and replicated multimedia services in wide-area P2P systems, and to evaluate the achieved QoA. Some selected characteristics of P2P systems, which motivate our work on the dynamic replica placement in this chapter are:

- Peers go up and down independently from each other. They are connected to a P2P network for a while and become disconnected after doing some service-related operations, e.g., downloading contents.
- Peers are symmetric in terms of supplying and demanding services or content. This means that there is no peer which is permanently serving other peers, and vice versa.
- Peers demand and supply *different levels* of service availability. Whether a peer has launched the P2P system's program and whether the peer still has enough storage capacity or access link bandwidth, affects strongly the supplying availability of the peer.

We model the P2P system as a dynamic stochastic graph. In this graph, all node and edge elements are parameterized, statistically independent of each other, with known availability and up-time probabilities. An availability requirement value is additionally assigned to each node so that the target replica placement problem is to find a replica set with which the availability requirements for all peers are satisfied. We also introduce the notion of *time epoch*, i.e., a unit time, and use a peer-up distribution (e.g., binomial distribution) to model the intermittent connectivity of the peers. Within this model, the number of nodes and edges of the stochastic graph changes between the time slots according to the used peer-up distribution whereas the graph is static within each time epoch.

As a replication model, we consider *partial* replication for our dynamic replica placement study, rather than the *full* replication used in the previous chapter where we tackled the static replica placement problem. Our partial replication model considers both the fraction of replicated contents and the replication ratio to determine the number of replicas for each individual content. Additionally, our model supports different times of replica creation. For instance, replicas are created either proactively by the peers which publish original contents or on-the-fly by the peers which access the content or its replicas.

The placement algorithms considered in this chapter are fully decentralized, ranking-based heuristics which work in a cooperative way to replicate contents on-the-fly. Thus, the heuristics do not assume any global information about the system condition or topology and work only with partial information which each peer can collect by exchanging its local information with its neighbors.

To quantitatively study the effectiveness of the proposed placement algorithms, we develop an event-driven simulation model which captures the data access model as well as peers' dynamic behavior, e.g., going up and down, etc.

Through the simulation study, we learn that the cooperative placement heuristics offer in general better QoA than a non-cooperative placement approach. Furthermore, the placements offered by the *UP+HA* heuristic which takes both uptime probability and supplying availability of the (neighbor) peers into account are always the best. Regarding replication distribution, the *Proportional* scheme offers higher satisfied QoA than the *Uniform* scheme for a Zipf-like access query model. This means that using popularity information affects clearly the placements and it increases the service availability of the system. However, when the replication dis-

tribution is *Uniform*, we can see that the query distribution was irrelevant for the achieved QoA.

5.2 Outline

The rest of this chapter is organized as follows. Section 5.3 discusses related work. In Section 5.4, we describe the P2P system and content distribution models which are used for our work on the dynamic replica placement throughout this chapter. Section 5.5 formulates the dynamic replica placement problem. It also details the target replication model and presents the placement heuristics used for solving the dynamic replica placement problem. In Section 5.6, we present the simulation methodology and parameter settings. The simulation results are analyzed in Section 5.7 and finally Section 5.8 concludes this chapter.

5.3 Related Work

Replication in P2P systems related works that have recently been published are [Kan02, KRT02, LCC+02, CS02]. Their goals are somewhat different from that of our work; maximizing hit probability of access requests for the contents in P2P community, minimizing content searching (look-up) time, minimizing the number of hops visited to find the requested content, minimizing replication cost, distributing peer (server) load, etc.

Kangasharju et al. [KRT02] studied the problem of optimally replicating objects in P2P communities. The goal of their work is to replicate content in order to maximize hit probability. They especially tackled the replica replacement problem where they proposed *LRU* (least recently used) and *MFU* (most frequently used) based local placement schemes to dynamically replicate new contents in a P2P community. As we will show (in Figure 4), maximizing hit probability does not satisfy the required QoA and, furthermore the two different goals lead to different results.

Lv et al. [LCC+02] and Cohen and Shenker [CS02] have recently addressed replication strategies in unstructured P2P networks. The goal of their work is to replicate in order to reduce random search times.

Yu and Vahdat [YV02] have recently addressed the costs and limits of replication for availability. The goal of their work is to solve the minimal replication cost problem for a given constraint on replication cost. The authors defined the replication cost as the sum of the cost of replica creation, replica tear down and replica usage. Our work differs in that our goal is to replicate content in order to satisfy different levels of QoA values required by individual users. Furthermore, their work does not take P2P system specific features such as changing peers' state into account.

There are many research efforts such as Chord [SMK+01], Tapestry [ZKJ01] and Pastry [RD01a] related to supporting lookup services. They detail the mechanisms for supporting the services that they offer such as indexing, lookup, insert, search, update, and delete. While some of them support fault tolerance by replicating the mapping information, i.e., the key/value binding information on multiple peers, they do not give any availability guarantee for values, e.g., files or multimedia contents, other than that of 'best-effort' availability support. Furthermore, it is not clear under which criterion the number and location of replicas are determined.

5.4 The Model

The next two subsections describe the P2P system and content distribution models which are used for our placement study throughout this chapter. The important issues that arise in the accurate modeling of a P2P content distribution system are intensively discussed.

5.4.1 Peer-to-Peer System Model

Basic Features

In the context of this thesis, P2P systems are characterized as being decentralized, self-organizing distributed systems, in which most communication is symmetric. Based on the arts of service discovery and (content) downloading mechanisms, the P2P systems can further be classified between structured and unstructured ones [LCC+02, MS03].

In this thesis, we mainly focus on decentralized and unstructured P2P architectures in which there is neither a centralized directory nor any precise control over the network topology or content placement. Some selected characteristics of this kind of P2P systems are as follows.

P2P Network topology

Each peer has a certain number of neighbors and the set of inter-neighbor connections forms a P2P overlay network. In the rest of this chapter, when we refer to the network, it is the P2P overlay network and not the underlying Internet. We assume that each peer has at least one neighbor, and that the P2P network graph does not change during the simulation of our placement algorithms. We use two network topologies in our simulation study:

- Normal Random Graph (Random): we generate two random graphs with 1000 and 10000 nodes by using Leda graph library [LED02].
- Power-Law Random Graph ([MLMB01]): this is a 1000-node random graph. The node degrees follow a power-law distribution: when ranked from the most connected to the least connected, the *i*-th most connected node has c/i^{α} neighbors, where *c* is a constant. Once the node degrees are chosen, the nodes are connected randomly (see [LCC+02, SFFF03]). Many real-life P2P networks have topologies that are power-law random graphs [FFF99, PSF+01, RHKS02, HPSS03].

Uptime

Peers go up and down independently from each other. They are connected to a P2P network for a while and become disconnected after doing some service-related operations, e.g., downloading or uploading contents. At any given time, a given peer may be up or down; it may be down because the peer's device is physically disconnected from the network. A peer is also not available when the peer service application is not launched. Each peer is assigned an up-time probability, that is, the fraction of time that the peer is up. It is assumed that the up probabilities of the peers are known and independent of each other. For simplicity, we assume that the peers have different up probabilities and consider three peer-up probabilities: 0.3, 0.5 and 0.9 on average. Additionally, based on the measurement study of [SGG02], we suppose that the up probability for the various peers follows the power-law distribution.

Storage

Each peer has private storage and shared storage. Only content in the shared storage can be accessed by the other peers of the system. The portion of the shared storage capacity can be different between the peers as the peers may be heterogeneous: they are either a powerful workstation, a personal computer, a laptop, or an Internet-connected PDA. We suppose that the content in a peer's shared storage is not lost during the peer's disconnection time. Thus, when the peer comes back up, all of the content in its shared storage is available again for sharing. This is generally the case in P2P file sharing systems such as Gnutella and KaZaA. In this case, we do not address the consistency issue, because we assume a read-only access.

Service Availability

Peers supply *different levels* of service availability. Whether a peer has launched the P2P system's program and whether the peer has still enough storage capacity or access link bandwidth, strongly affect the supplying availability of the peer. To determine the service availability supplied ($sa_{i_{sup}}$) by a peer, we only take the availability of the storage and access link bandwidth into account:

$$sa_{i_{sup}} = \left(w_1 \cdot \frac{s_{i_{free}}}{s_{i_{total}}} \right) + \left(w_2 \cdot \frac{b_{i_{free}}}{b_{i_{total}}} \right) \text{ with} \tag{10}$$

$s_{i_{free}}$: free storage capacity of the peer *i*,
$s_{i_{total}}$: total storage capacity of the peer *i*,
$b_{i_{free}}$: free access link bandwidth of the peer *i*,
$b_{i_{total}}$: total access link bandwidth of the peer *i*, and

$$(0 \leq w_1, w_2) \wedge (w_1 + w_2 = 1) \tag{11}$$

where w_1 and w_2 are constant values.

However the availability level, that peers demand at service access time, differs between peers. Some peers may expect extremely high available access, while other peers may be happy with '*best-effort*' availability level. The demanding availability level can be either given directly from the application as a QoA specification as mentioned in Section 3.4 or a replication-level availability specification mapped from the higher-level one.

Modeling P2P System as a Stochastic Graph

We model the P2P system as a dynamic stochastic graph G(V,E). In this graph, the nodes go up and down depending on their assigned uptime probability and issue content access events with a certain level of availability requirements. We assign QoA values to every node of the graph, where the required QoA value and the supplying QoA value are decoupled for each node. The required QoA value is assigned at the graph creation time, while the supplying QoA value is calculated by checking the node's own availability probability value and its link degree (#of adjacent links).

The scope of dynamics that we capture in this work are peers' state (e.g., either up or down) which causes the change of the number of total peers being up, their connectivity and their available storage capacity. Concerning a peer's state and availability of contents located on the

peer, we can assume that the contents on the peer are unavailable, when the peer goes down. In our P2P model which will be used in the simulation study, we treat the up/down probability of each peer as (a) given as *a prior* knowledge or (b) unknown.

5.4.2 The Content Distribution Model

This section describes how we model the type and size of content each peer chooses to share. In [SGG02, QSS02, SGD+03], it is observed that P2P networks are usually heterogeneous in terms of type and size of contents shared. For example, the P2P file sharing applications such as FastTrack and KaZaA provide peers sharing of MP3 music files and video clip files typically in the size of 3-4 Mbytes and 10-100 Mbytes, respectively. Furthermore, it is also observed in KaZaA and [SGG02, SGD+03] that only a small portion of contents is widely shared among peers. For example, in Gnutella, approximately 1% of the total files are shared intensively between 0.1% of the total peers. Regarding the popularity of individual contents, many P2P-based content distribution systems such as Napster and Gnutella, as well as the Web [QPV01, PQ00], exhibit Zipf-like distributions [BCF+99]. This reflects the fact that some popular contents are very widely replicated and carried, while most contents are held by far fewer peers.

For our dynamic replica placement study in this thesis, we model a content distribution model which reflects the real-world P2P networks described above.

We assume that there are *m* contents of different sizes, e.g., 3, 15, and 90 Mbytes, which are shared among *n* peers. Let b_j denote the size of the *j*th content and q_j be the relative popularity, in terms of the number of access queries issued for it, of the *j*th content. We normalize the query popularity values as:

$$\sum_{j=1}^{m} q_j = 1 \tag{12}$$

In this formulation, we investigate the following two query distributions:

- Uniform: all contents are accessed with equal popularity probability.

$$\text{Uniform: } q_j = 1/m \tag{13}$$

- Zipf-like: contents are accessed with different popularity which follows a Zipf-like distribution.

$$\text{Zipf-like: } q_j \propto 1/j^{\alpha} \quad \text{where } \alpha \text{ is a coefficient.} \tag{14}$$

5.5 Dynamic Replica Placement

Having specified parameters of the target P2P system and content distribution model, we now formulate the dynamic replica placement problem in this section. As a replication model, we consider *partial* replication for our placement study, rather than the *full* replication supposed in the static replica placement problem where the replica target was the whole contents (complete replica). Thus the goal of the static replica placement study was to select replica servers to place the complete replicas.

The main focus of our dynamic replica placement study is on determining both the number of replicas and their placement so that the service availability offered to all peers is improved. In particular, we consider the following two aspects in the partial replication model:

- Replica creation time (*when to replicate?*)
- Individual location of replicas (*on which peers are replicas placed?*)

We suppose that replicas are created either proactively by the peers which publish original contents or on-the-fly by the peers which access the content or its replicas. Thus, we divide the replica creation time into two phases: proactively at the service initialization phase and on-the-fly during the service running phase.

In addition to the replica creation time, another important decision issue on the replica placement is the location of replicas. As observed in the static replica placement study (Chapter 4), the replica location is a more relevant factor than the number of replicas for service availability. The following two sections discuss some relevant issues related to the placement decision and describe placement algorithms used for solving the placement problem in each of the two replica creation phases.

5.5.1 Proactive Placement

In the proactive placement, the location of replicas is usually determined prior to the beginning of service. Additionally, the location does not change during service time, although the popularity of individual contents may vary or new contents may be created and published during the service time. We concentrate on the proactive placement in this section, while we tackle both proactive and on-the-fly placements in our dynamic replica placement study in this chapter.

Our proactive replication model considers both the fraction of replicated contents (i.e., *how many contents are replicated?*) and the replication ratio (i.e., *on how many peers should contents be replicated?*). The core of our partial replication model is a mapping function $r: \{1,\ldots,m\} \rightarrow \{1,\ldots,n\}$ so that the value of $r(j)$ is the number of replicas of content *j* (hereafter we denote r_j as the number of replicas of content *j*).

For each of the *m* contents, the number of replicas can be defined individually and may be different. Deciding how many replicas for a particular content are required depends on the given service availability requirements specified by individual peers. Moreover, it also depends on the underlying P2P system's condition and the access model used. For the proactive placement model, we consider the following two replication distributions which are widely used to model replication distribution in many other replica placement studies [LCC+02, KRT02, KKM02, CAMN02]:

- Uniform: all contents are replicated on the same number of peers, hence the contents have the same replication ratio.

$$\text{Uniform: } r_j = R/m \tag{15}$$

  where R is the total number of replicas stored in the P2P system.

$$R = \sum_{j=1}^{m} r_j \tag{16}$$

- Proportional: the replication of a content *j* is proportional to the query probability of the content. Thus, the number of replicas reflects the popularity of the content. To use this distribution, it is often assumed that the request probability for the *m* contents are known beforehand.

$$\text{Proportional:} \quad r_j \propto c q_j \tag{17}$$

  where c is an arbitrary constant and q_j the query probability for the *j*th content.

Regarding algorithms for proactive data placement, there are many heuristics proposed [TA92, KKM02, Chu99, KM02]. All of them assume an *a priori* knowledge about data access patterns and/or network topology and traffic. As one can observe from their results, the 'goodness' of the proactive placements are significantly affected by the following fact, whether there is any global information about system condition, network topology and service usage pattern, and which can be used for determining the placement. However, in contrast to a CDN or server-based systems, making information about the total system condition, globally and consistently available is extremely difficult, especially in fully decentralized and unstructured P2P systems. This is due to highly dynamic peer behaviors [SGG02, SGD+03].

To perform the proactive placement in this chapter, we basically use a *Random* placement scheme. Suppose *I* is a peer that creates *k* replicas for content *j* in the system. We assume that the number of target replicas *k* is determined already. Then peer *I* performs proactive placement for the content *j* as follows:

- *Random (RA)*. By using a random generator, peer *I* picks a peer node *v* with uniform probability, but without considering the node's supplying availability and up probability. If the node *v* has enough storage capacity available for holding the content *j* and it was not already selected as a replica node, then *I* puts *v* into the replica set, until the given number reaches *k*.

5.5.2 On-The-Fly Placement

So far we have concentrated on the proactive replica placement, ignoring the peers' dynamic behavior and contents' popularity. However, for the contents which are accessed frequently and thus become highly popular, it is important to dynamically select new replicas and replace them throughout the system for increasing their availability.

In this section, we investigate dynamic on-line replica placement which is termed as on-the-fly replica placement. A key requirement for determining the location of replicas on-the-fly is short decision time. In wide-area P2P systems, it is not necessary to make optimal choices which often lead to higher decision time; rather, it is sufficient to make good choices in a majority of cases and to avoid poor decisions. Thus, we can apply simple 'ranking-based' or 'improvement-based' heuristics [KKM02], instead of using exact methods, to the on-the-fly placement problem.

However, to take the intermittent connectivity of peers explicitly into account, we should not assume any global metadata about system characteristics such as the current network condition and topology. This means that each peer has only partial information about the system and content access history, which the peer can collect by exchanging the information with its neighbors. Furthermore, the information may be inconsistent due to the nature of frequently changing connectivity of the peers. Keeping these requirements in mind, we propose a number

of ‘ranking-based’, fully decentralized heuristics which are independently executed on every peer node in the system to determine the location of new replicas. Following is a summary of the used heuristics. Suppose *I* is a peer that issues the query to get content *j* in the system:

- *Local*. To create or replace a new replica during service runtime (i.e., simulation runtime), the peer that issues the access query places a new replica on its local storage. If the available storage capacity of the peer is less than the size of the replica, existing replicas can be replaced. The replica replacement policy bases its decision either on *least recently used* (LRU) or on *least frequently used* (LFU) concept.
- *HighlyUpFirst (UP)*. The basic principle of the *UP* heuristic is that peers with (relatively) higher uptime probability can potentially be reached by more peers. Thus, when a new replica should be placed, the peer first compares its own uptime probability with those of its neighbors and then places the replica on the peer which has the highest uptime probability.
- *HighlyAvailableFirst (HA)*. Instead of comparing the uptime probability of peers, this method checks the actual availability of each neighbor node. Thus, for each neighbor *v*, peer *I* calculates its own and *v*’s actual supplying availability value. The peers are then sorted in decreasing order of their actual availability value. Finally, the best peer is selected as replica node. Again, the use of the UP and HA heuristics does not assume any *a priori* knowledge about the network topology and traffic.
- *Combined (HA+UP)*. This method is a combination of the *HA* and *UP* algorithms. For this algorithm, peer *I* first calculates the average values of uptime probability and supplying availability for all directly neighboring peers. It then selects the peer as a replica node for which both values are greater than the average values: we first check the uptime probability value and then the availability probability value. However, there may be several combined forms of the heuristics.

The key difference between *Local* and the heuristics (*HA*, *UP* and *HA+UP*) is that the heuristics work in a cooperative way to decide the location of replicas to be created. The ‘goodness’ of the placements achieved by each of the algorithms is analyzed and discussed through an event-driven simulation study that we present in the following two sections.

5.6 Simulation

This section describes our experimental environment that we built to perform an event-driven simulation study for the dynamic replica placement problem. We present the simulation methodology, parameter settings and the metrics used in the simulation to evaluate the heuristics.

5.6.1 Simulation Methodology

To study the effectiveness of the proposed placement heuristics on service availability in P2P systems, we have performed simulation experiments. We built an event-driven simulation environment which enables us to conduct both proactive and on-the-fly replica placement simulation with varying up peers and query events.

For each set of simulations, we first selected the network size and topology, peers’ uptime probability distribution, and the query and replication distributions. By using the LEDA graph

library [LED02] and BRITE [MLMB01] topology generator, we generated several random and Power-Law topologies with different sizes. At the beginning of the simulation the graph was parameterized with the given availability and uptime probability distributions. Then, the number of contents, query events and simulation time slots was initialized. Additionally, a given number of contents were created and assigned randomly to the peers of the graph.

In the proactive placement phase, for each content *j* with replication ratio r_j or replica number *k*, depending on the fixed replica distribution, e.g., *Uniform* or *Proportional*, we generated *numPlacement* (e.g., 10) different sets of random replica placements: each set contains r_j random peers and each of them has only one replica of content *j*.

At the beginning of the on-the-fly placement phase, we randomly decided the number of up peers per each simulation time slot. Then, for each replica placement of the total *numPlacement* placements, we randomly chose *numQueryEvent* different peers from which to initiate a query for content *j*. For each query, we simulated the searching process using the designated search method: constrained *Flooding* with maximal hop counts (as default, smaller than 4). When the query was successful, we checked how many replicas (including the original content) are available and whether downloading the target content from them satisfies the service availability requirement of the querying peer. Regarding the downloading method, we assumed multiple downloading paths which are supported by some real P2P systems such as KaZaA. Depending on the query results we checked the achieved service availability. New replicas were created only, when there was at least one replica (or the original content) available, which supplied a lower service availability value than the required one of the querying peer.

5.6.2 Parameter Settings

To reflect the characteristics of real P2P systems such as Gnutella, we used the statistics and measurement data sets on peers' uptime and different content size from [SGG02, SGD+03, ABC+02]. Thus, we assigned different uptime probability values to each peer and only varied the average uptime values. For each experiment, we assumed 1,000 contents and considered three different content sizes for three different types of contents, e.g., 3MB, 15MB and 90MB for mp3 music audio, music video clips and video titles, respectively. The assignment of content size to each content was randomly distributed. For convenience, we supposed that the query probabilities for the various contents follow both *Uniform* and *Zipf*-like distributions.

Regarding storage capacity of peers, we assumed that each peer contributes a different amount of shared storage to the system. Based on the measurement study of [SGG02], we considered three different storage sizes: 60*b*, 100*b* and 300*b*, where 1*b* is 3MB. Figure 18 shows the basic distributions used in our simulation study. Table 7 summarizes the parameter settings and the random number functions used for our simulation. The simulation program is written in C/C++ and runs on Linux and Sun Solaris machines.

5.6.3 Metrics

This section presents the metrics used to evaluate the 'goodness', specifically the QoA offered by different replica placements for the modeled P2P system. We evaluate the achieved service availability or cost due to the placement produced by various placement algorithms, and not how far from the optimal cost function value a placement solution is. This is because one of the

primary goals of our dynamic replica placement study in this chapter is to quantitatively examine the effectiveness of different placement algorithms on the service availability of P2P systems. We use the following three metrics. These metrics, though simple, reflect the fundamental properties of the used algorithms:

- Satisfied QoA: as defined in Section 3.6, this metric indicates, for each querying peer, how much the availability is fulfilled at the selected placement. The value is given as the ratio of supplied availability to demanding availability for the querying peer. To evaluate the reached QoA both for each query and each simulation time slot, the following two formulas are used:

$$\text{Satisfied QoA for each query:} \quad QoA_{sat}(j) = QoA_{supplied}(j) / QoA_{satdemand}(j) \tag{18}$$

$$\text{Satisfied QoA per simulation time slot:} \quad QoA_{avg} = 1/n(\sum QoA_{sat}(j)) \tag{19}$$

with n : number of simulation time slot

$$\text{Number of successful queries}(NumQuerySuccess) = \sum (QoA_{sat}(j) \geq 1) \tag{20}$$

- Number of queries failed (*NumQueryFailed*): a query failed, when the target content (either the original or its replicas) is unreachable at the selected placement.
- Number of replica replacement (*NumReplace*): this metric reflects the cost due to the replacement required during the on-the-fly placement phase. Replacement occurs after a successful query, when the supplied availability is less than the required, even though there are replicas or the original content. Thus, the number of replacement can be calculated as:

NumReplace = NumQueryEvent - NumQuerySuccess - NumQueryFailed

Parameter	Values
test graphs	Random LEDA: G1(100,300), G2(1000,3000), BRITE: G3(1000, 3626)
peer up probability	0.0 - 1.0 (average in default: 0.3)
peer's storage capacity	180MB, 300MB, 900MB
content size	3MB, 15MB, 90MB
content popularity	0.01 - 0.99
range of demand availability values	0.50 - 0.99
range of peer's supplying availability	0.51 - 0.99
number of peers	100, 1000
number of origin contents	1000
number of query events	1000
number of simulation time slots	100
query distribution	Uniform, Zipf-like
proactive placement method	Random
on-demand placement heuristics	Local-LRU, UP, HA, HA+UP

Table 7: Simulation parameters and their value ranges

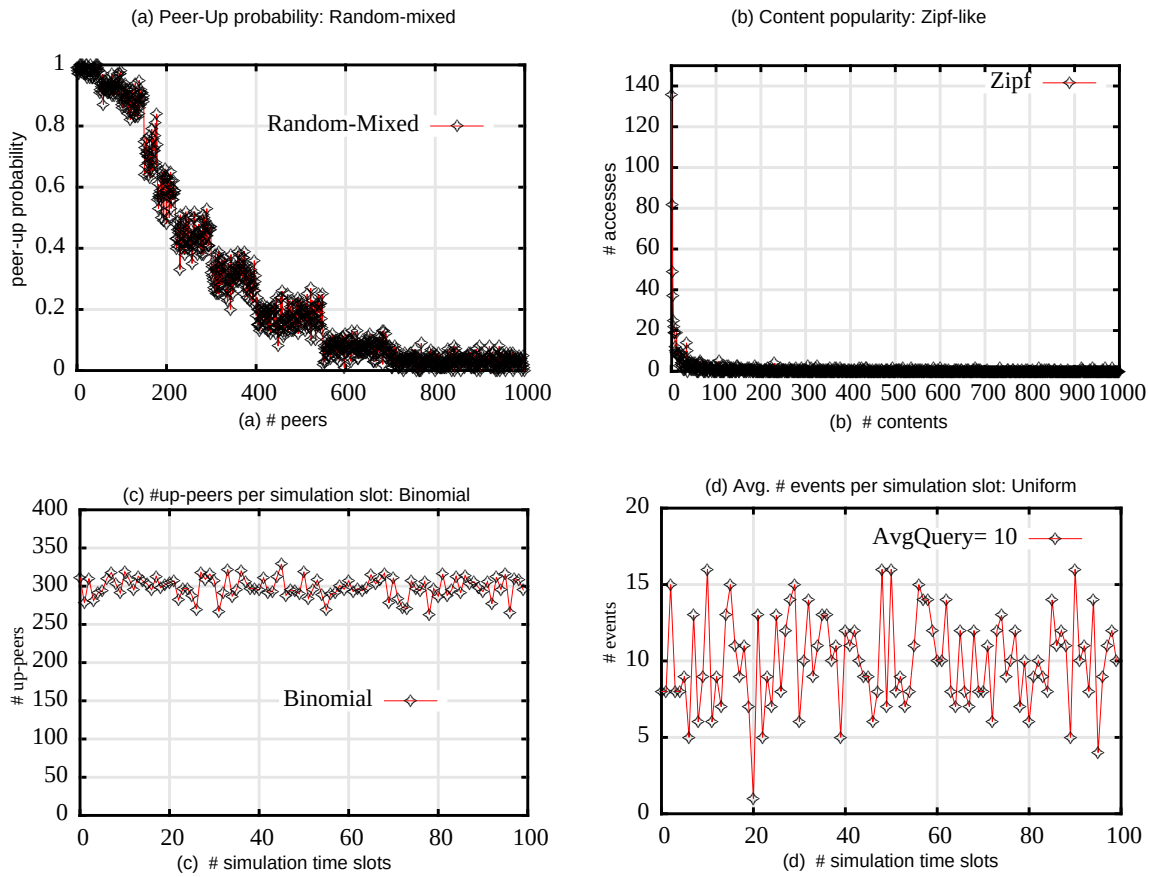


Figure 18: Basic distributions used in our simulation study

5.7 Goodness of Placement

This section presents the simulation results and analyses the service availability of both proactive and on-the-fly placement. We evaluated the reached QoA of the heuristics using topologies of different sizes and by varying the replication ratio. Statistics are collected from *numPlacement* (=10) simulation runs for each placement heuristic with each fixed parameter setting. We then calculated the aggregate results for the above metrics.

5.7.1 Effects of |R| on Satisfied QoA

The first experiment examines how the number of initial replicas affects the satisfied QoA. For this purpose, we varied the replication ratio from 0.05 to 1.0. Again, the replication ratio 0.005 means that 5% of the total 1,000 peers have a replica for a particular content. We used test graph G2 with parameter settings given in Table 7, where we fixed the peers' average up probability as 0.3. We activated both proactive and on-the-fly placements and used *Random* and *Local-LRU* heuristics as placement strategies for the two phases. Peer-up probability and query distributions followed the basic distributions shown in Figure 18.

Figure 19 shows the results from this experiment: it shows 12 graphs corresponding to the 12 different replication ratios. We plot the simulation time slot on the x-axis and the average satisfied QoA for each simulation time slot (*Avg.satisfiedQoA*) on the y-axis. As Figure 19

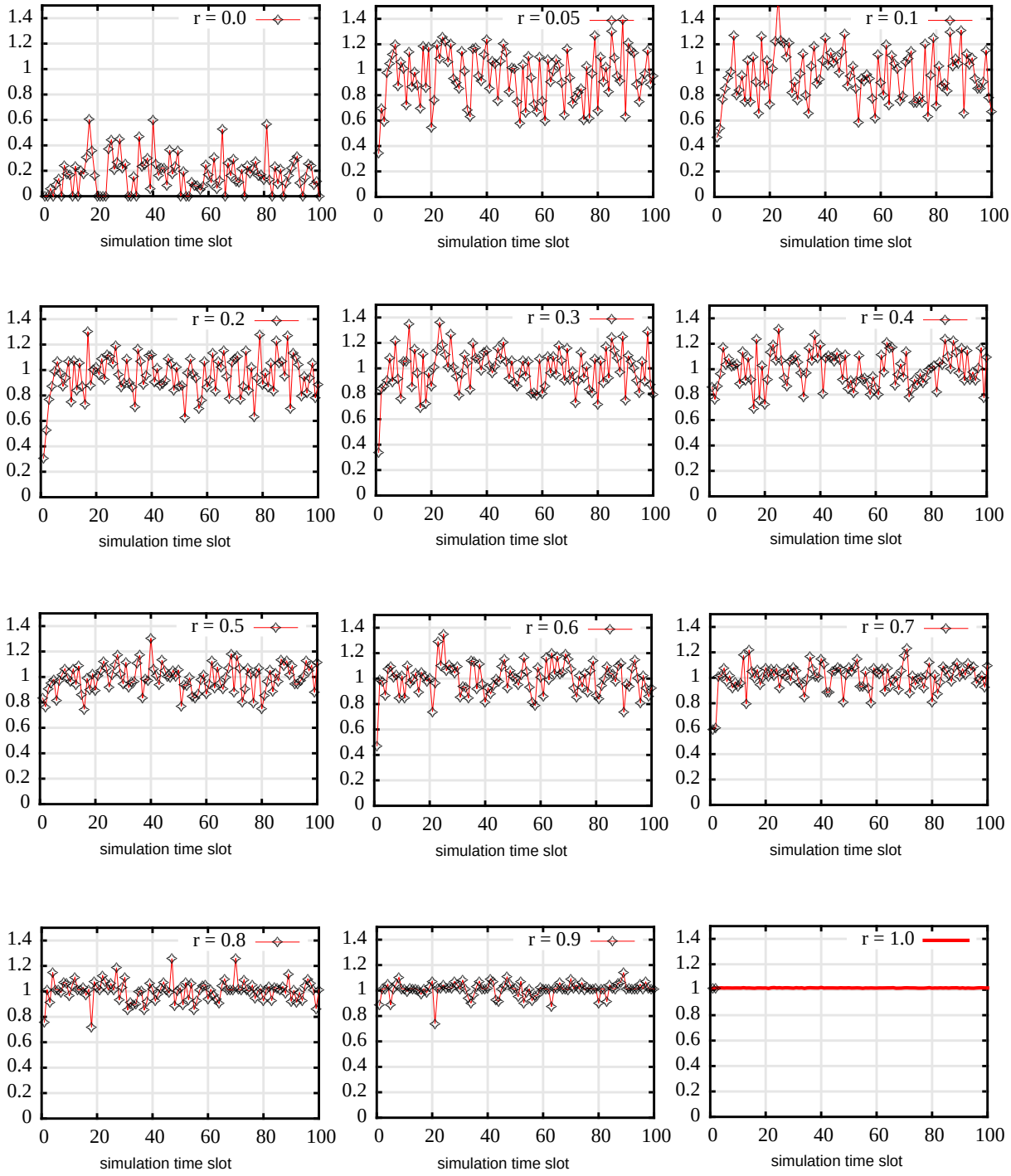


Figure 19: Effects of replication ratio on satisfied QoA where proactive placement: *Random*, #peers=1000, peers' up probability=0.3, and on-demand placement: *Local-LRU*. y-axis is the average satisfied QoA for each simulation time slot.

shows, by increasing the replication ratio, the average satisfied QoA values are converging towards 1. This means the number of peers which contain the requested content (or its replica) on their own local storage is proportional to the replication ratio.

5.7.2 Effects of On-the-Fly Placement Schemes on Satisfied QoA

The second experiment examines the effects of the used on-the-fly placement heuristics on the satisfied QoA. We took different on-the-fly schemes to create new replicas during the simulation run when the QoA which is supplied by the up peers holding replicas does not satisfy the demanding QoA of the querying peer. In addition to the *Local* scheme, we tested the three cooperative heuristics *UP, HA*, and *UP+HA*. The goal of this experiment is to investigate whether the on-the-fly placement increases the overall service availability of the system, and how effective the cooperative heuristics are, in comparison to the *Local* placement method. We used the same parameter settings as in the first experiment, but took a small number of replicas for each content, from 1 to 10, to reflect the typical replica number of many real P2P systems. For instance, Farsite [ABC+02] creates 3 replicas each for all content types whereas Pond [REG+03] and PAST [RD01b] create 4-6 replicas each to support fault tolerance.

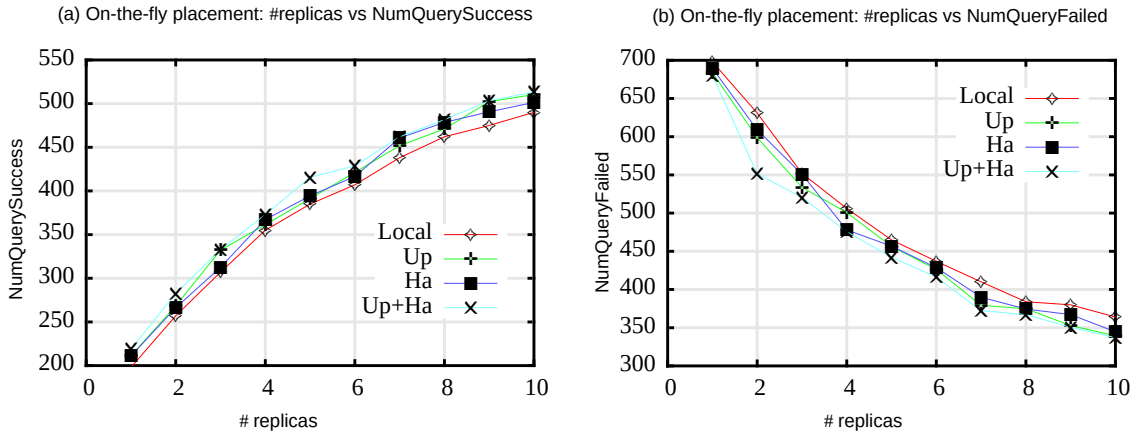


Figure 20: Effects of on-the-fly placement schemes on satisfied QoA - proactive placement: *Random*, #peers: 1000, peers' up probability: 0.3, and query model: *Zipf*.

Figure 20 shows the result from this experiment with test graph G2. We plot the number of replicas on the x-axis and the total number of querying peers on the y-axis, for which the availability requirement was fulfilled (i.e., the value of satisfied $QoA \geq 1$) or failed (i.e., the value of satisfied $QoA < 1$) in (a) and (b), respectively. For each heuristic, the simulation was performed 10 times. As we can see in Figure 20-(a), all the three cooperative heuristics (*UP, HA*, and *UP+HA)* outperform the *Local* scheme and fulfill the availability requirement for more peers (i.e, 5-50 more peers in average) than the *Local*. In particular, our simulation result shows that the placements offered by *UP+HA* are always the best due to its feature, i.e., taking both uptime probability and supplying availability of the peers into account. This further indicates that the cooperative placement schemes, offer in general better QoA than non-cooperative local placement approach.

In Figure 20-(b), we compare the number of failed queries, i.e., the queries that could not find any replica in the system. As we can see, the placements offered by the cooperative heuristics lead to less numbers of failed queries than that of the *Local*. In Figure 20, we can also observe that the number of failed queries decreased, as the number of initial replicas increased.

To examine whether the (better) QoA values achieved by the cooperative heuristics are invariant from replication degree, we took a higher number of replicas for the proactive placement and compared the on-the-fly heuristics again. For this purpose, we varied the replication ratio from 0.0 (no replica) to 1.0 (replicated to all peers) and checked the achieved availability. Figure 21 shows the satisfied QoA (y-axis) which was calculated as an average value for each simulation run as the replication ratio (x-axis) increased. We can see from the figures that the cooperative heuristics, though simple, achieved considerably higher satisfied QoA than the *Local* scheme. For example, the QoA improvement of the replication ratio range 0.1-0.5 is about 30-70%. Figure 21-(b) shows that this improvement pattern is observable, independent of the graph size: Peer100 and Peer1K in figure (b) are equal to the nodes size 100 (graph G1) and 1000 (graph G2).

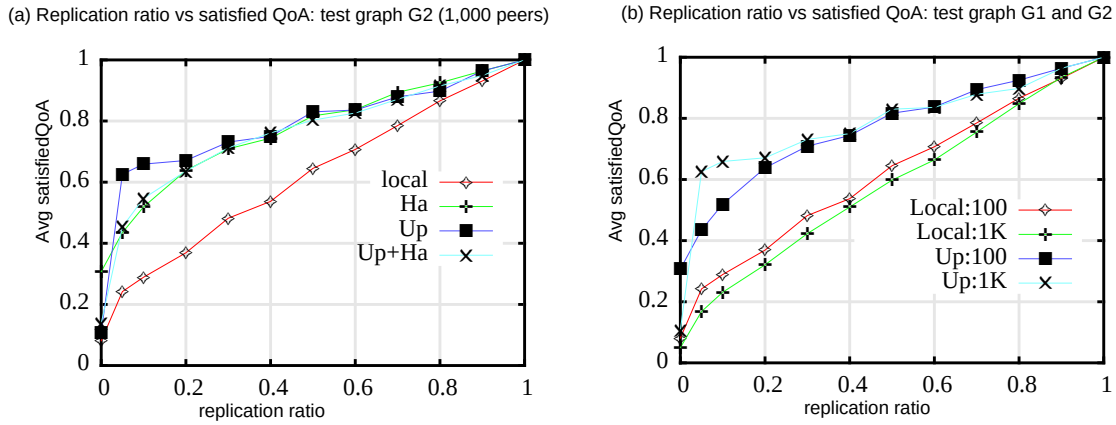


Figure 21: Effect of on-the-fly placement strategies on satisfied QoA. In (b), Peer100 and Peer1K mean 100 and 1000 peers, respectively.

To compare the cost due to the on-the-fly placement produced by the heuristics, we took the number of replica replacements into account, that occur after a successful query, when the supplied QoA value is less than the required value for the querying peer. The statistics are collected from the simulation runs of Figure 20.

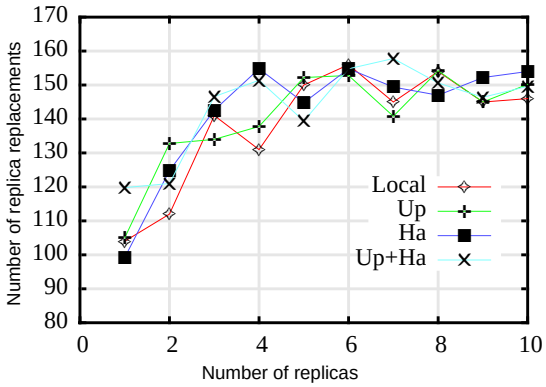


Figure 22: Cost of on-the-fly placement schemes: number of replica replacement

In Figure 22, we plot the number of replicas on x-axis and the number of replacements produced by the four heuristics on y-axis. As we can see from the figure it is not clear which heu-

ristic is particularly bad or good in terms of the replacement cost. Instead, we may argue that both local and cooperative placement schemes induce approximately the same amount of replacement cost.

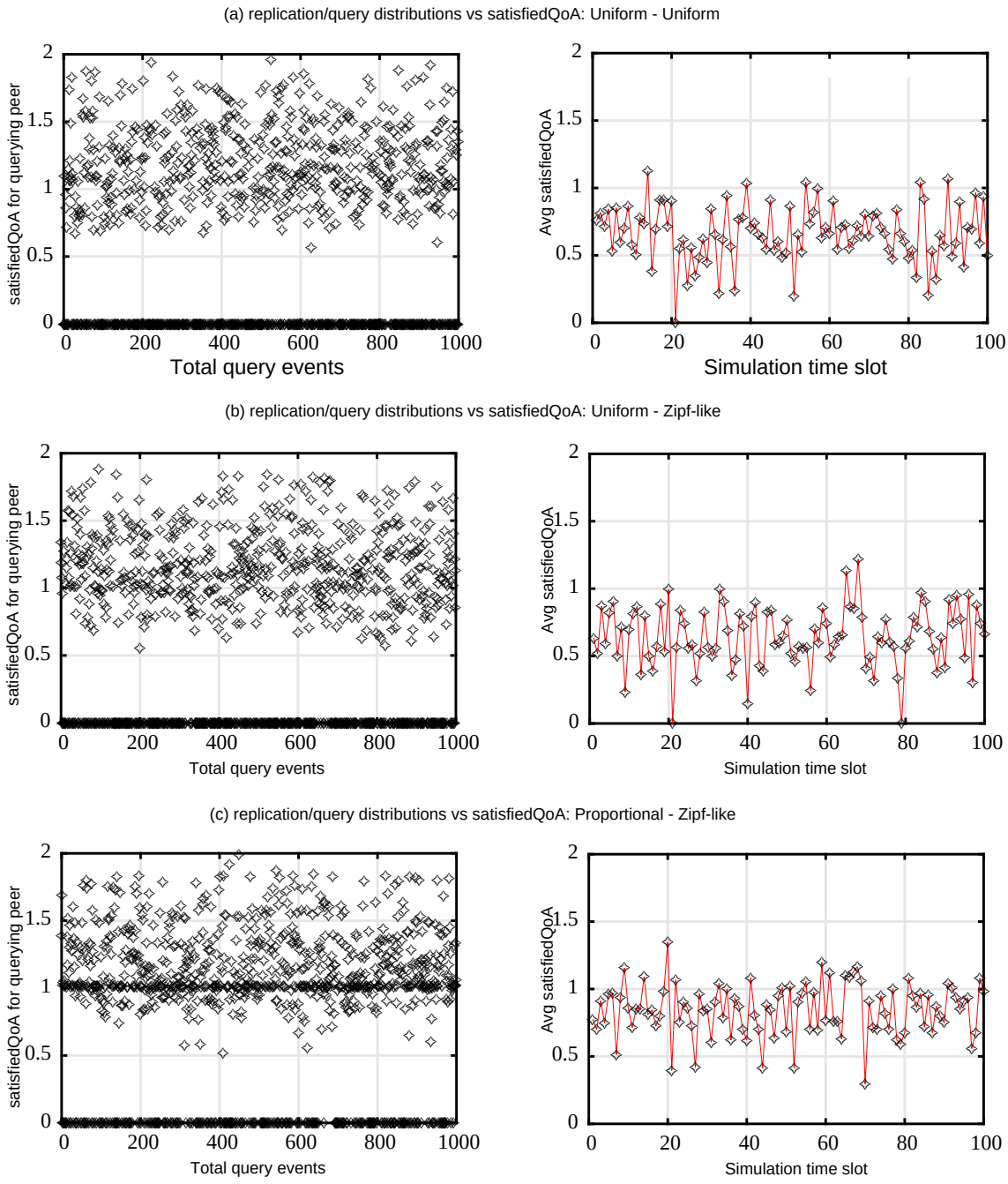


Figure 23: Effects of initial replica selection schemes on satisfied QoA

5.7.3 Effects of Initial Replica Selection on Satisfied QoA

In the third experiment, we compared the two replica selection schemes - *Uniform* and *Proportional* which decide, for a given fixed number of *k*, the target replicas among original contents at the proactive placement phase (i.e., service initialization phase). In this experiment, we

placed the *k* replicas on randomly chosen peers which do not contain the original content of the corresponding replica. Furthermore, the peer contains only one replica for each original content. For this experiment, we used test graph G2 with parameter settings given in Table 7, where we fixed the peers' average up probability as 0.3 and the initial replica number as 5. As on-the-fly placement scheme, we used *HA+UP* heuristic, because it offered the best placement, as shown in Figure 20. As query distribution, we used both *Uniform* and *Zipf-like* distributions.

Figure 23 compares the achieved satisfied QoA under all combinations of replication and query distributions. We can see from the figures that the *Proportional* scheme offers higher satisfied QoA than the *Uniform* scheme for the *Zipf*-like access query model. This further indicates that using popularity information clearly affects the placements and thus increases the service availability of the system. However where the replication distribution is *Uniform*, we can see that the query distribution is irrelevant for the achieved QoA.

5.7.4 Satisfied QoA versus Hit Probability

Maximizing hit probability is one frequently used goal for content replication [Kan02]. In Figure 24, we show a comparison between the two replication goals, i.e., satisfying required QoA and maximizing hit probability. In this comparison the hit probability is increased when the querying peer finds the target content. While for satisfying QoA the peer should additionally check the supplied QoA by calculating all the reachable paths to the peers containing the target content (or replica). We ran the simulation on the test graphs G1 and G2. The average up probability of peers was fixed again as 0.3 and we used *Random* and *UP* placement schemes for the proactive and on-the-fly phase.

As Figure 24 shows satisfying required QoA incurs higher cost, i.e., greater numbers of replicas are required than just maximizing hit probability. For example, at replica rate=0.2, the gap between *sqoa* (satisfied QoA) and *Found* (hit probability reached) is about 20% of achieved rate. To achieve the same rate of 80%, for satisfying QoA, we needed a 30% higher replication ratio.

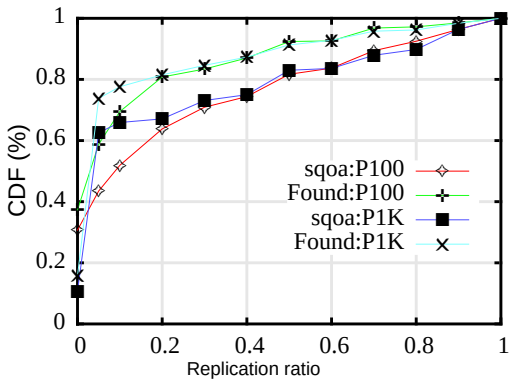


Figure 24: Comparison of replication cost for different replication goals: satisfying QoA vs. maximizing hit probability. P100 and P1K mean 100 and 1000 nodes, respectively. X-axis shows replication ratio. Y-axis shows the cumulative distribution function of the achieved QoA (sqoa) and hit rate (Found).

5.8 Summary

In this chapter, we have studied dynamic content replica placement in P2P-based content distribution and replication systems. We have investigated specifically dynamic content replication for availability, where our goal is choosing dynamically the number and location of replicas for a particular content to satisfy the service availability requirement for all individual peers, while taking intermittent connectivity of peers explicitly into account.

As a replication model, we have considered *partial* replication which takes both the fraction of replicated contents and the replication ratio into account to determine the number of replicas for each individual content. Additionally, we considered different times of replica creation and divided the replica creation time into two phases: proactively at the service initialization phase and on-the-fly during the service running phase.

We have modeled the P2P system as a dynamic stochastic graph, in which all node and edge elements are parameterized, statistically independent of each other, with known availability and up-time probabilities. Having specified replication and query distribution models, we have presented a number of placement algorithms which are fully decentralized, ranking-based heuristics which work in a cooperative way to replicate contents on-the-fly. Thus, the heuristics required no global information about the system condition or topology and work only with partial information which each peer can collect by exchanging its local information with its neighbors.

To quantitatively study the effectiveness of the proposed placement algorithms, we have developed an event-driven simulation model which captures the data access model as well as the peers' dynamic behavior. The following observations could be identified from our experimental results:

- The cooperative placement heuristics such as *HA, UP,* and *UP+HA* offer in general better QoA than the non-cooperative *Local* placement approach. Furthermore, the placements offered by the *UP+HA* which takes both uptime probability and supplying availability of the (neighbor) peers into account are always the best.
- Both *Local* and the four cooperative placement heuristics applied to the on-the-fly placement induce approximately the same amount of replacement cost.
- Regarding replication distribution, the *Proportional* scheme offers higher satisfied QoA than the *Uniform* scheme for the *Zipf*-like access query model. This means that using popularity information clearly affects the placements and it increases the service availability of the system. However, when the replication distribution is *Uniform*, we can see that the query distribution is irrelevant for the achieved QoA.

Our proposed replication and simulation model can be used for further study on the dual availability and performance QoS for dynamically changing, large-scale P2P systems. It can also be used on the replica placement for QoA guarantees for those systems.

Chapter 6 - Efficient Replica Placement with QoA Guarantees

Providing QoA guarantees with selected replica placements requires, as shown in Chapter 4, either a higher computational complexity (e.g., exponential time by the State Enumeration method) or a higher number of replicas than for simply improving QoA. In this chapter, we investigate an *efficient* placement algorithm for solving the server replica placement problem, of which the computational complexity grows *polynomially* with respect to the size of the input system, while ensuring QoA guarantees for all demanding nodes in a capacitated content distribution and replication system. Our algorithm is based on the notion of *admission control* which is known from the quality of service area. Thus, when deciding about replica locations it always regulates access to resources (replicas) to avoid resource overload which makes the service system unavailable. It also regulates access to protect it from access requests that cannot be fulfilled. The algorithm consists of three procedures: *highest available path*-based fast placement with QoA guarantees, *delete-and-merge* for reducing replica numbers and *move-and-update* for decreasing delay. As a consequence, for a given set of QoA requirements and network topologies, the algorithm finds replica placements in polynomial time, which not only guarantee QoA for all demanding nodes, but also improve the overall performance in terms of reduced access delay. We show its feasibility and practicality through a simulative study.

6.1 Motivation

Through the last two chapters we sought to understand how number and location of replicas affect the service availability of content distribution and replication systems. We also sought to evaluate classes of techniques both for improving and guaranteeing QoA. By performing several simulative experiments, we could observe that the exact method, the *State Enumeration* algorithm can find exactly the optimum and provide QoA guarantee with its placement results. However the algorithm requires exponential computation time ($O(2^{2^{|V|+|E|}})$) due to its nature of exact checks. In contrast to the exact algorithm, the heuristics such as *HA* and *COM* were fast ($O(|V|)$) and scalable yet they provide no QoA guarantees with their placements offered.

In this chapter, we intensively seek to develop new placement algorithms which are fast enough and satisfy the different service availability requirements for all demanding nodes with QoA guarantees.

One of the most well-known approaches to provide service guarantees is performing *admission control*. Typically, the admission control concept has been applied to resource management in distributed multimedia systems which need to provide quality of service (QoS) guarantees for applications that often have timely delivery requirements to be met [Wol96,

CDK01, Wan01, SN04]. Thus, to accomplish QoS guarantees for those applications, each resource manager has the admission control task to regulate access to resources to avoid resource overload and to protect the system from requests that cannot be fulfilled [CDK01].

However, in applying the admission control concept to solving our replica placement problem in this thesis, we do not deal with resource management in the data transmission context. Instead of that, the resource management in our CDRS means deciding which contents and replicas to keep in the system, which replica to purge, and which service request to accept from which replica (node), while guaranteeing resource availability and QoA.

As a target problem domain, we basically consider the static replica placement problem for guaranteeing QoA (the SRP-G problem) again, which has been tackled in Chapter 4. Given a set of QoA requirements and a parameterized, unconstrained and stochastic graph, where we assume a full and static replication model, we seek to find placements, i.e., the number and location of servers, at which to place replicas to serve a given set of client locations (i.e., service requesters) with QoA guarantees.

The main result of this chapter is a fast placement algorithm consisting of three procedures: *highest available path*-based fast placement with QoA guarantees, *delete-and-merge* for reducing replica numbers and *move-and-update* for decreasing delay. Important features and limitations of the algorithm are as follows:

- Through the three procedures, it needs only a polynomial time ($O(|V|^2)$) to find placements which provide QoA guarantees.
- It takes storage and link bandwidth constraints into account. Therefore, it checks for each node holding a replica, whether the total number of requests on node v which are active at time t neither exceeds the node's storage nor the access link bandwidth capacity.
- It also improves performance in terms of reduced delay (i.e., reduced hop count) for all service requestors (clients) by moving replica locations, while keeping QoA guarantees.

Through a simulative study, we experimentally evaluate the algorithm with random and power-law graphs of different sizes. We could observe that the placement achieved by the admission control-based algorithm always provides a QoA guarantee, and that the upper bound of the replication ratio required for QoA guarantees is approximately 20 percent of the total nodes. To the best of our knowledge, our admission control based placement algorithm is the first to quickly solve the replica placement problem for service availability guarantees.

6.2 Outline

The rest of this chapter is organized as follows. In Section 6.3, we describe the system model and metrics used in this chapter. We also outline the principle of the admission control based placement technique. Then, in Section 6.4, we present the scope, the placement decision scheme and algorithm details. In Section 6.5, simulation parameters are described and the placement results are evaluated. Related work is discussed in Section 6.6, and Section 6.7 concludes this chapter.

6.3 Admission Control Based Replica Placement

In this section, we first describe the system model and metrics, that we consider for the placement problem in this chapter. We then outline the principle of the admission control (AC)-based placement technique and define several key functions and terms used in our AC based placement algorithm.

6.3.1 System Model

The content distribution and replication system (CDRS) model considered in this chapter is basically the same one as described for the static replica placement study in Chapter 4. However, one key difference between the two architectures is that we take resource constraints explicitly into account in the model of this chapter. Thus, we are given a CDRS consisting of a set of content storage servers and connection links between them. The content storage servers and links have a limited amount of capacity available. The content storage servers are symmetric in terms of content access types (e.g., either supplying or demanding contents), which means that each content storage server can be selected as a replica location. A failure probability value is assigned to each content storage server and connection link. Additionally, each content storage server has a certain QoA requirement level that must be met by the content storage server selected as replica location, when the content storage server requests accessing any content. We model the AC-based placement problem as a *static*, *stochastic*, *capacitated* and *unconstrained* graph, as illustrated in Figure 25.

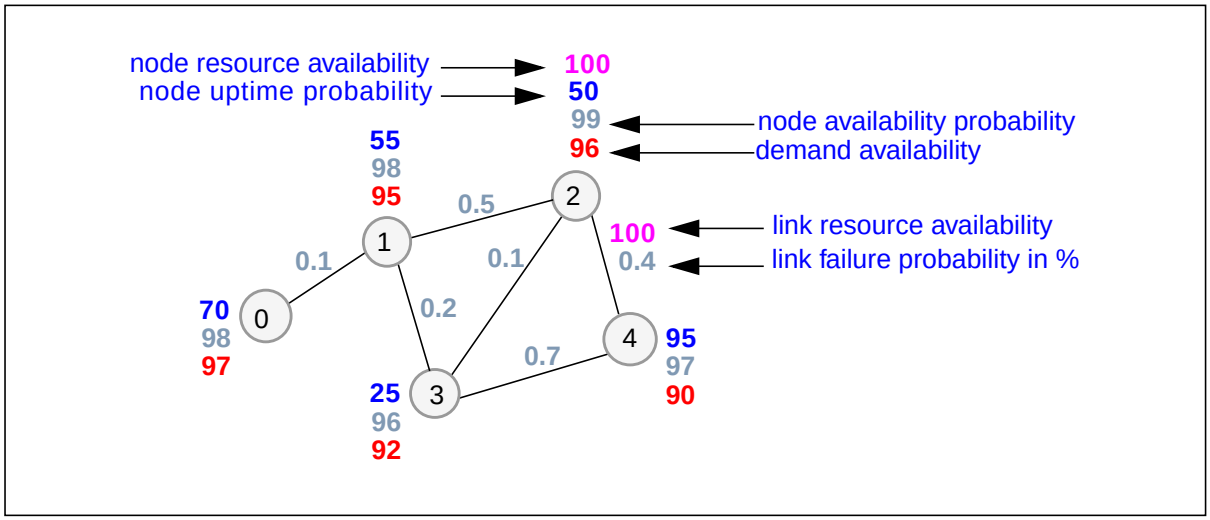


Figure 25: An illustration of a parameterized stochastic graph G(V,E).

6.3.2 Metrics

Important metrics used to evaluate the ‘goodness’, i.e., the placement offered by the AC-based placement algorithm are as follows:

- QoA: we specifically consider *guaranteedQoA* which indicates, as defined in Section 3.6, whether the service availability (QoA) achieved by the selected placement is greater, or at least equal to, the one required. Basically, the achieved service availability can be calculated

by Equation 6 in Section 3.4.3. In the case of the AC-based placement, the achieved availability is obtained as shown in the next section. The guaranteed QoA for a given service demanding entity (node) is represented as $QoA_{gua}(v)$, while for the whole demanding entities it is given as QoA_{avgGua}.

- Replication cost: it covers in general the overall overhead incurred due to replication support such as storing, replacing, transmission. In this chapter, we only take storage cost into account and define replication cost as the *number* of replicas required to provide QoA guarantees.
- Delay: in addition to guaranteeing QoA, we also address performance aspects of replication and seek to improve the performance in terms of decreasing delay, while selecting replicas. We define delay as number of hops between service supplying and demanding entities, i.e., between a replica and its assigned client node.

6.3.3 The Principle

The primary goal of our placement study in this chapter is finding a replica placement (i.e., a set of server replicas) which guarantees that, for every node/edge and access time *t*, the required levels of QoA for all demanding nodes are met. Moreover, the placement guarantees that the total number of requests on node *v* which are active at time *t* does not exceed the capacity of node's resources, e.g., the capacity of storage and access link bandwidth. Thus, the condition of an admission control is:

$$(u_{ST}(t) \le STOR_U) \wedge (u_{BW}(t) \le BW_U) \quad \text{with} \tag{21}$$

$STOR_U$: total storage capacity of the node *i*,
$u_{ST}(t)$: used storage capacity of the node *i* at time *t*,
BW_U : total access link bandwidth of the node *i*,
$u_{BW}(t)$: used access link bandwidth of the node *i* at time *t*.

We assume that every node is a 'decision maker' performing the admission control. The node is thought of a 'candidate' replica node, when the origin node holding the original content or the existing replica nodes cannot fulfill the QoA demand for a particular node issuing an access request. Concerning system and service information, we further assume that each decision maker has globally accessible information about the state of the system topology and condition. As replication model, we basically consider proactive replication, in which the placement decision is made prior to starting service and not during service running. This leads in turn to a *static* admission control policy that supports a relatively long-term service duration as is the case of the resource capacity planning problem [Chu99].

6.3.4 Definitions

In this section, we define several key terms and functions used in our AC-based placement algorithm.

Given an undirected (static, stochastic) graph $G(V,E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges, let $N = |V|$ and $M = |E|$. A (cycle-free) *path* is a finite sequence of nodes $p = (v_0, v_1, \ldots, v_h)$, such that for $0 \le n < h$, $(v_n, v_{n+1}) \in E$, and for all $0 \le m < n \le h$, $v_m \ne v_n$, and each node and edge are selected only once; h is then said to be the *number of hops* (or hop

count) of p. We denote H the maximal possible number of hops in a path. We assign to each node $v \in V$ and edge $e \in E$ a non-negative weight (i.e., failure probability) q_v and q_e, respectively. Corresponding a path p, there is a *path failure weight* $FW(p)$, which is a non-decreasing function of the weights of the links along the path. The path failure weight for a (directed) path from a distinguished source node v_0 to a distinguished terminal node v_h with a hop count h can be formally given as

$$FW(p) = 1 - \left(\prod_{i=1}^{h} (1 - q_{(v_{i-1}, v_i)}) \cdot (1 - q_{v_i}) \right) \tag{22}$$

Given failure weights for nodes and edges and destination nodes $s,t \in V$, a *highest available path* is a path between s and t of minimum failure weight. An *h*-hop constrained highest available path is a path of minimum failure weight, among paths between s and t with hop count of at most *h*. An instance of the highest available path problem is the problem of finding a highest available path for given destination nodes, with the minimum total failure weight.

The service availability supplied by a highest available path, $AW(p)$ is:

$$AW(p) = 1 - FW(p) \ . \tag{23}$$

For any source node v, a highest available path p provides QoA guarantee, if its supplied availability $AW(p)$ is greater than the demanding QoA of the node $QoA_{v_{req}}$:

$$AW(p) \geq QoA_{v_{req}} . \tag{24}$$

6.4 The Algorithm

The AC-based replica placement algorithm consists of three procedures: *highest available path* based fast placement with QoA guarantees, *delete-and-merge* for reducing the number of replica and *move-and-update* for decreasing delay. This section presents the algorithm in terms of the scope, placement decision scheme and computational complexity.

6.4.1 *Highest Available Path* based Fast Placement

In the first procedure of the AC-based placement algorithm, we find the highest available path (HAP) for each of the demanding nodes and then test whether the found HAP gives a QoA guarantee, i.e. whether the supplied QoA achieved by the selected path is greater than the required QoA for the demanding node, as given in Equation (24).

For implementation of the procedure for finding HAP, we adopted the Dijkstra's shortest path algorithm [HSM95], which uses basically a 'greedy' technique. We modified it to support undirected graphs and the availability probability parameters for both nodes and links. This modification also enables us to keep all the HAPs from the given source node to every node of the graph. A pseudo-code of the HAP algorithm is given in Figure 26.

The algorithm is given a source node s and a parameterized graph G as input. It returns arrays of the HAP's failure weight and of the individual HAPs from the source node to every node t of the graph.

The algorithm uses a priority queue *PQ* to maintain nodes of the graph G, whose highest available path from the source node is to be found. Initially, *PQ* contains only the source node.

At each step, we get a node u from *PQ*, whose failure weight is the smallest among all the nodes queued in *PQ* (the first node chosen is the source node). Then, for all adjacent nodes r of u, we check its failure weight from the source node s, which can be calculated by multiplying the availability values of its own and the adjacent connection link to the source node. When the adjacent node r is visited the first time, it is inserted into *PQ* with its failure weight, otherwise only its failure weight is updated, i.e., decreased and finally the path from s to r is updated. The computational complexity of the HAP algorithm depends basically on the implementation methods of the priority queue and the graph. The priority queue *PQ* is implemented by Fibonacci Heaps [Wei94], in which operations *insert*, *delete*, *del_min* take time $O(\log n)$, *find_min*, *decrease_failVal* take time $O(1)$ and *clear* takes time $O(n)$, where n is the size of *PQ*.

```
Algorithm HighestAvailablePath (node s, graph G, array fail, array pred)
// It finds the highest available paths from the source node s to every node of G = (V,E)
Input:    Source node s;
          An undirected capacitated stochastic graph G;
Output:   fail[i]: an array of the HAP's failure weight from s to every node i of G;
          pred[i]: an array of the HAP from s to every node i of G;

begin
1.    for all v ∈ V  do pred[v]: = nil; end-for
2.    fail[s]:= 0;
3.    PQ.insert(s,0);                  // put the source node into priority queue PQ
4.    while (!PQ.empty()) do           // for all |V| nodes
5.          node u := PQ.del_min(); // choose a node with the minimum failure weight
6.          for all adjacent edges e_adj of u do
7.                node r := G.get_opposite_node( e_adj , u );
8.                calculate the failure probability failVal until the node r;
9.                if (pred[r] == nil and r != s) then
10.                     PQ.insert(r, failVal);// r is visited first time
11.               else if (failVal < fail[r]) then
12.                     PQ.decrease(r, failVal);   // failure probability for r is updated
13.                     pred[r] := e_adj ;
14.               else continue;
15.               fail[r]: = failVal;              // value assignment for fail[]
16.               pred[r] := e_adj ;               // remembering path from s to r
17.         end-for
18.   end-while
end
```

Figure 26: Pseudo-code of *HighestAvailablePath (HAP)* algorithm.

To represent the graph, we use LEDA [LED02] which maintains graphs as an adjacency list, which in turn leads operations for accessing nodes to take time $O(\log n)$. As a consequence, the running time of the HAP algorithm of Figure 26 is bounded by $O(e \log n)$: the lines 1 and 3 take time $O(n)$, the loop of line 4 itself takes time $O(\log n)$ and the sub-loop of the lines 6 and 16 takes $O(e)$; thus, a total of e update will be made, each at a cost of $O(\log n)$ time, so that the total time spent in lines 4 and 18 is $O(ne)$. As a result, the running time of the HAP algorithm is bounded by $O(ne)$.

According to the path found and the QoA value supplied by this HAP, if the path cannot provide QoA guarantee for the source node, we select a node along the path as a new replica, which is closest to the terminal node (i.e. service supplying node) and has enough resource to be allocated and fulfils the QoA requirement. The node is then added to the replica node set.

```
Procedure HAP-basedPlacement
Input:      A stochastic graph G (V, E),
            Node c holding the original content
Output:     A placement R with QoA guarantees, i.e. A_R(G) = 1
Initialize: R := {};
            u_v := maxCap;      // maximal resource capacity available for all v ∈ V
begin
1.    for all v ∈ V do
2.     if ( v = c or v ∈ R ) then
3.          if ( v = c ) then Update (v , c , nil) else Update (v , v , nil);
4.     else
5.          find highest available path (HAP) from v to all r ∈ R and c ;
6.          select the best path p with highest supply QoA, AW(p)
7.          if ( AW(p) ≥ QoA_req(v) and u_h ≥ u_req(v) ) then
8.               Update (v , h , p );          // h is the terminal node of the path p
9.          else
10.              // create a new replica along the path p
11.              search a node t along the path p such that t is closest to p and fulfills:
12.                   AW(p_t) ≥ QoA_req(v) and u_t ≥ u_req(v) .
13.              // path p_t is a new HAP from v to t .
14.              if ( t is Found) then
15.                   R.insert(t );          // t is a new replica
16.                   allocate the t 's resource for the assigned node v
17.                   Update (v , t , p_t );
18.              else
19.                   // the source, querying node v becomes a new replica
20.                   R.insert(v );
21.                   allocate the v 's resource for v itself
22.                   Update (v , v , nil);
23.              end-if-else
24.         end-if-else
25.    end-if-else
26.   end-for
27.   return R ;
end
```

Figure 27: Pseudo-code of *HAP-based Placement* procedure.

Figure 27 shows the pseudo-code of our HAP-based replica placement procedure. It assumes that we are given a parameterized stochastic graph and a node holding the original contents as input. For every node of the graph, if it is a service demanding node (i.e., neither the original node nor a replica), it finds the HAP both to the original node and replicas by using the HAP algorithm and selects the best one. It then checks whether the path offers a supplied QoA value greater than the required one. If yes, the demanding node will be assigned to the

terminal node of the path, otherwise a new replica is created along the path. The querying node itself can become a replica node, if its demanding QoA cannot be fulfilled by the nodes residing along the path. Information such as demanding node assignment, achieved QoA state and hop count are kept and updated during the procedure run, which are then used for evaluating the placement determined. Figure 28 shows a pseudo code of a procedure updating these information.

```
procedure Update (node s, node t, path p)
// s: the source, querying node, t: the terminal node, and p: HAP from s to t
begin
        QoA_gua(s) := 1;
      assign s to node t ;
      if (p = nil) then
            hopCnt(s) := 0;
              QoA_supplied(s) = QoA_supplied(t)     ;
      else
            hopCnt(s) := length(p) ;
             QoA_supplied(s) = AW(p)  ;
      end-if-else
end
```

Figure 28: Pseudo-code of *Update* procedure.

6.4.2 Reduction of Replica Numbers

The placement, i.e., the replica set *R* offered by the HAP-based placement procedure guarantees QoA for all demanding nodes. However, *R* is neither the optimum nor has been optimized. Thus, this section deals with an optimization technique for reducing the number of replicas, while keeping the QoA guarantee. Our approach presented in this section is based on deletion of replicas. The basic idea is as follows: for every replica node *r* of *R*, it is checked whether all the demanding nodes (clients) assigned to the replica can find an other replica from *R*, which also provides QoA guarantee for them; if found, the replica node *r* can be deleted from the replica set *R*. The clients are then assigned to their best replica with respect to the satisfied QoA. The available storage capacity of the replicas which accept the new client(s) is also updated. We implemented this approach as a second procedure of our AC-based placement algorithm. Figure 29 illustrates this procedure.

```
Procedure delete-and-merge
Input:      A stochastic graph G (V, E),
            Replica node set R
Output:     A placement R2 with QoA guarantees, i.e. A_R2(G) = 1
Initialize: R2 := {};

begin
1.    for all r ∈ R  do
2.          hide r ;
3.          checkOK = 0;
4.          for all v assigned to r do
5.                find highest available path (HAP) from v to all w ∈ R except r ;
6.                select the best path p with highest supply QoA, AW(p)
7.                if ( AW(p) ≥ QoA_req(v)  and  u_h ≥ u_req(v) ) then
8.                      // h is the terminal node of the HAP p
9.                      remember h and p for v ;
10.                     checkOK++;
11.               end-if
12.         end-for
13.         if (checkOK == 'number of the total nodes assigned to r ') then
14.               // r can be deleted completely from the replica node set R
15.               R.delete(r );
16.               free the r 's resource allocated for the assigned nodes
17.               for all v assigned to r do
18.                     // Update QoA assignments for all v with h and p stored
19.                     Update (v , h , p );
20.               end-for
21.         end-if
22.         recover r ;
23.   end-for
24.   return R2 := R ;
end
```

Figure 29: Pseudo-code of *delete-and-merge* procedure.

6.4.3 Reduction of Hop Count

In this section, we describe the third procedure of our AC-based placement algorithm. The primary goal of the procedure is to improve performance by reducing the hop count between every replica and its assigned demanding nodes. As an approach to achieve this goal, we use a heuristic called '*move-and-update*.' The basic idea of this approach is that we replace the replica with its neighbor, if the neighbor can also provide, for the demanding nodes assigned to the chosen replica, a QoA guarantee and further has a total hop count smaller than that of the replica. Figure 30 illustrates this *move-and-update* procedure. By varying the radius of hops, we can control the set size of neighbors, which may affect the chance for finding such a neighbor achieving a placement with a better performance. One limitation of this algorithm is the (re-)assignment condition. As the lines 7-15 of Figure 30 show, a neighbor which can be selected as a new replica should cover all the demanding nodes assigned to the chosen replica.

An alternative approach could be checking a replacement possibility for each individual demanding node to each of different neighbors separately.

```
Procedure move-and-update
Input:      A stochastic graph G (V, E);
            Replica node set R;
            HopCnt;            // maximum hop count for testing move
Output:     A placement R3 with QoA guarantees, i.e. A_R3(G) = 1
Initialize: R3 := {};

begin
1.    for all r ∈ R do
2.          build a set R_adj with adjacent nodes r_adj of r;
3.          // nodes r_adj are reachable by r within the HopCnt;
4.          moveOK := FALSE; OK := 0;
5.          while (!checkedAll(R_adj) and !moveOK) do
6.                w := choose a node from R_adj;
7.                for all v assigned to r do
8.                      find highest available path (HAP) p from v to w;
9.                      if ( AW(p) ≥ QoA_req(v) and u_w ≥ u_req(v) ) then
10.                           // w is the terminal node of the HAP p
11.                           remember w and p for v;
12.                           remember individual hop count from v to w;
13.                           OK++;
14.                     end-if
15.               end-for
16.               if (OK == 'number of the total nodes assigned to r') then
17.                     if (totalHopCount(w) < HopCount(r) then
18.                           R.delete(r);
19.                           free the r 's resource allocated for the assigned nodes
20.                           for all v assigned to r do Update (v, w, p); end-for
21.                           moveOK = TRUE;
22.                           R.insert(w);
23.                           allocate the w 's resource for the assigned nodes
24.                     end-if
25.               end-if
26.         end-while
27.   end-for
28.   return R3 := R;
end
```

Figure 30: Pseudo-code of *move-and-update* procedure.

6.5 Simulation and Evaluation

To examine the effectiveness of the AC-based placement algorithm, we performed a simulation study. In this section, we first describe the simulation parameters for our simulation study and then present the results.

6.5.1 Simulation Parameters

The simulations were performed on a set of random network topologies. By using the LEDA graph library [LED02] and BRITE [MLMB01] topology generator, we generated several random and Power-Law topologies in different sizes. Node degree information of these test graphs are shown in Table 8.

random graph	graph size (V, E)	average degree	maximum degree	graph type
G0	(20, 30)	3.3	6	Random
G1	(100, 200)	4.7	10	Random
G2	(1000, 3000)	6.3	15	Random
G3	(10000, 50000)	10.1	26	Random
G4	(1000, 3626)	7.2	155	Power-Law

Table 8: Test graphs used

In these simulations, we assume the same replication model as that one used in Chapter 4, i.e., *static* and *full replication* in which the whole data items of an origin server system are replicated to other nodes located within the same network prior to starting service. Simulation programs of all procedures are in written in C/C++.

The following three sections present the results of the AC-based algorithm and analyse its effectiveness on QoA and performance (hop count). To evaluate the achieved QoA, the replica number needed, and the hop count required by the algorithm, we used the metrics defined in Section 6.3.2.

6.5.2 HAP based Placement for Guaranteeing QoA

In this section, we compare the achieved QoA values of the placement results and replica numbers needed by the HAP-based placement algorithm for providing the QoA guarantee. The baseline settings for this experiment are as follows. The first node which should hold the original contents is chosen randomly. Availability and failure probability distributions for nodes and edges follow a uniform distribution: the values are chosen from a range [90,99] and [1,10], respectively. First, we wanted to check how many nodes are needed to meet the QoA requirement value of all demanding nodes. For this purpose, simulations were performed 20 times for every graph with the parameter settings described above. Table 9 gives the test results. In columns 1 and 2, test graphs of different types (Random, Power-Law) and sizes are given, column 3 contains the replica number that the HAP-based algorithm needed to provide placements with the QoA guarantee. Column 4 is the replication ratio which means the ratio of the replica number needed to the total nodes. Columns 5 and 6 are the guaranteed and satisfied QoA on average. From Table 9, we can see that the HAP-based algorithm always guarantees the QoA, and that the minimum replication ratio required by those placements of the algorithm is about 0.4.

test graph	|V|	replica number	replication ratio	QoA_{avgGua}	satisfiedQoA (avg.)
G0	20	6	0.3	1.0	1.03096
G1	100	30	0.3	1.0	1.15212
G2	1,000	408	0.4	1.0	1.16178
G3	10,000	1223	0.12	1.0	1.00993
G4	1,000	402	0.4	1.0	1.15760

Table 9: Simulation results: HAP-based placement

6.5.3 Reducing Replica Number by Deletion

In the second experiment, we examined whether the number of replicas can be reduced and further how many replicas can be deleted, while keeping the QoA guarantee. Therefore, we took the same parameter settings as for the previous experiment and activated the deletion procedure of the AC-based algorithm. The simulation was performed 20 times for each test graph due to the random selection of the first node (source node). Table 10 gives the test results.

test graph	|V|	replica number			satisfiedQoA (avg.)	
		before	after	% Improv.	before	after
G0	20	6	5	20	1.03096	1.03474
G1	100	30	24	25	1.15212	1.13936
G2	1,000	408	277	47	1.16178	1.13738
G3	10,000	1223	1032	19	1.00993	1.00948
G4	1,000	402	376	7	1.15760	1.16540

Table 10: Simulation results: replica number reduction.

In columns 3, 4 and 5, we can see that the deletion procedure could achieve an improvement rate of the replication cost, i.e., the reduction rate of the replica numbers, between 7 and 47 percent: the highest reduction is achieved for the random graph with 1,000 nodes, whose replication ratio in the first placement procedure was the highest. However, regarding the satisfied QoA value, it is remarkable, as columns 6 and 7 of Table 10 show, that the deletion of replicas does not (significantly) affect the achieved QoA. Whereas the QoA decreased up to 1%, 2%, and 0.01% for the random graphs G1, G2, and G3, those QoA values for G0 and G4 (power-law graph) increased 0.4% and 0.7%, respectively.

6.5.4 Reducing Hop Counts by Move

The primary goal of the third experiment is to reduce the total hop counts between the replicas selected and all the demanding nodes assigned to one of the replicas, while the QoA guarantee must be satisfied and the number of replicas should be either smaller or at least the same one. For this experiment, we took the same parameter settings as for the two previous experiments and activated the *move-and-update* procedure of the AC-based algorithm. We varied the radius length for movement from 1 to 3 to build different set sizes of neighbor nodes. The simulation was performed 20 times for each test graph. Test results are given in Table 11, where we can see that the number of movements was very low: only from 0% to 1%, except in the case for G3 containing 10,000 nodes, where the movement rate is 2.6%.

Columns 3, 4 and 5 of Table 11 give the total hop counts between the selected replicas and all the demanding nodes assigned to them: in these three columns, we can see that the total hop counts after the deletion procedure have increased in all the test graphs. This indicates a correlation between the replica number and the hop count: the hop counts (i.e., path lengths) between replicas and demanding nodes become longer, when the replica number decreases. In columns 7 and 8 of Table 11, we can find the same phenomenon: the average hop count for each replica has increased by reduced replica number. Moreover, from Table 10 and Table 11, we can also see that the achieved QoA values can be affected by the hop counts: longer paths cause lower QoA achieved.

Table 11 also shows that the *move-and-update* procedure could decrease the hop counts. Additionally, we could observe in the experiments, that the varied radius length has not affected the movement results.

test graph	number of move	total hop counts			average hop counts per replica		
		HAP	Deletion	Move	HAP	Deletion	Move
G0	0	29	31	31	1.200	1.600	1.600
G1	1	161	214	196	2.169	2.856	2.661
G2	2	3,175	4,034	3,580	2.875	4.492	4.134
G3	258	53382	61152	51524	5.111	5.575	5.364
G4	3	1729	1846	1508	0.650	0.794	0.730

Table 11: Simulation results: moving replica places.

6.6 Related Work

Although there have been a lot of research efforts and work on solving the placement problem, none of them (explicitly) addresses the issue of replica placement for providing service (availability) guarantees. [CS00] adopts the quality of service (QoS) concept to networked storage service and maps the best-effort and guaranteed performance classes to caching and replication, respectively. Based on this concept, the authors investigated the resource allocation prob-

lem in [Chu99], which shows, through a simulation study, that services with deterministic guarantees require more network resources than those with statistical or stochastic guarantees.

Concerning resource-constrained replica placement, [KPR99] presents storage-constrained placement algorithms for hierarchical cooperative caching. The proposed algorithm seeks to optimize the cost of the final, complete placement and runs in a batch mode. [VDW01] extends this algorithm for solving the bandwidth-constrained placement problem for wide-area network replication. The goal of this study is placing replicas at a collection of distributed caches to minimize expected access time from clients subject to a maximum bandwidth constraints at each cache. Their hierarchical greedy-based algorithm generates a set of placements that are within a constant factor of the optimal.

The basic techniques we adopted and examined for providing service guarantees have been studied in other contexts. The admission control technique is often used in distributed multimedia systems [Wol96, CDK01, Sch01a, Wan01, SN04], to provide a certain QoS level to applications, as well as for the call control problem or the routing problem in a capacitated network [EJ97, PUW00, Erl02].

However, to the best of our knowledge, the admission control technique has not been applied (explicitly) to solving the replica placement problem.

6.7 Summary

For the first time an *admission control* approach is suggested as a means for solving the replica placement problem, where the goal is providing a QoA guarantee for all service requesting entities in a capacitated CDRS. Based on the admission control concept, we have developed a replica placement algorithm which solves the problem in a polynomial time, with respect to the size of the input system (graph). The algorithm consists of three procedures: *highest available path* based fast placement with QoA guarantees, *delete-and-merge* for reducing replica numbers and *move-and-update* for decreasing delay. We have presented each procedure of the algorithm in detail, in terms of their placement decision principle and implementation techniques.

Through a simulative study, we have experimentally evaluated the algorithm with random and power-law graphs of different sizes. We could observe that the placement achieved by the AC-based algorithm always provides a QoA guarantee, and that the upper bound of the replication ratio required for QoA guarantees is about 27% and 37% of the total nodes in the random and power-law graph, respectively. To the best of our knowledge, our admission control based placement algorithm is the first to efficiently solve the replica placement problem for service availability guarantees.

Chapter 7 - Conclusions and Outlook

7.1 Conclusions

In this thesis, service availability for large-scale content distribution and replication systems in wide-area internetworks has been investigated. The main focus was on issues of providing availability guarantees for all individual users of applications running on top of these systems. Specifically, by tackling the replica selection and placement problem, the thesis has examined the effectiveness of realistic replication strategies on the achieved service availability in those widely distributed and replicated systems such as the Internet.

There are several contributions that this thesis makes. The first contribution is the concept of QoA which enables one to treat availability as a controllable and observable QoS parameter. On the basis of this concept, three refinements of the existing availability definition have been investigated (decoupled, differentiated and fine-grained) in order to enable a more quantitative specification and evaluation of the service availability. For a technical realization of the QoA concept, we have also developed a QoA framework which comprises of the four modules, i.e., service specification, QoA mapping, admission control and replica management. Within the QoA framework, we have shown how users can specify their service requirements in terms of QoA, how the requirements are mapped into the low-level replication specification, and how the QoA guarantee can be controlled.

The second contribution is an evaluation of the replication techniques used to resolve the replica placement problem for different content distribution and replication system models. We have tackled the replica placement problem and investigated specifically the effects of the degree of replication, the granularity and location of replicas on overall service availability achieved by the selected placements. We divided the placement problem into two sub-problems, (1) the static full server replica placement in CDN-like replicated systems and (2) the dynamic partial data replica placement in peer-to-peer (P2P) systems. For solving the static full server replica placement problem, we first modelled the content distribution and replication system as a stochastic graph and then developed several ranking-based heuristics and an exact state enumeration algorithm for improving and guaranteeing service availability of the system. By simulating the behaviour of the proposed algorithms, we found that the location of replicas is a relevant factor for the service availability. Even though the QoA improvement could be achieved by increasing replica numbers, replicas' placement and their dependability affected the QoA more significantly.

In the case of the dynamic replica placement problem in P2P-based content distribution and replication systems, we took the intermittent connectivity of peers of the system explicitly into account. We modelled the system as a dynamic stochastic graph where each node (peer) goes up and down according to its assigned up-time probability. We considered different time scales for replica creation so that replicas could be created either proactively at the service initialization phase, or on-the-fly during the service running phase. The main concern of the proactive placement was the target and number of replicas, where we considered that each content can be

replicated either uniformly with an equal number or differently depending on its popularity, if known. For the on-the-fly placement, we concentrated on both placement and replacement schemes of replicas. We re-used the placement heuristics used for solving the static replica placement problem and modified them slightly so that they are fully decentralized, assume no global information about the system condition or network topology, and work in a cooperative way to replicate content on-the-fly. To quantitatively study the effectiveness of the used heuristics, we developed an event-driven simulation framework which captures the data access model as well as peers' dynamic behaviour. The simulation results indicated that the cooperative placement heuristics offer, in general, better QoA than non-cooperative local placement approach, while all of them induce approximately the same amount of replacement cost in terms of the number of replacements.

The third contribution is a new approach for determining replica placements in large-scale content distribution and replication systems, which always provides an availability guarantee for all service requesting entities in a content distribution and replication system. For the first time, we suggested an admission control-based replica placement scheme that solves the static replica placement problem in a polynomial time, with respect to the size of the input system. The algorithm consists of three procedures: *highest available path* based fast placement with QoA guarantees, *delete-and-merge* for reducing replica numbers and *move-and-update* for decreasing delay. Through a simulative study, we have experimentally evaluated the admission control-based algorithm with random and power-law graphs of different sizes. Our results show that the placement achieved by the admission control-based algorithm always provides a QoA guarantee, and that the upper bound of the replication degree (i.e., replica numbers) required for delivering QoA guarantees is about 27% and 37% of the total nodes in the random and power-law graph, respectively. As far as we know this is the first time that the admission control concept has been integrated with a replica placement scheme that solves the problem in a polynomial time while offering service availability guarantees.

7.2 Outlook

While several issues on providing service guarantees for content distribution and replication systems have been studied and replication strategies to achieving them have been provided in this thesis, this work has several possible avenues for future research. Below, we outline how the research in this thesis could be extended.

One direction for future work is to integrate our QoA framework with existing performance-centred QoS techniques. This can be done, for example, by extending our replication strategies so that they can capture major performance-specific QoS parameters such as jitter, throughput or delay bound, depending on target (often time-dependent) applications. They can provide replica placements, both with availability and performance guarantees. It might be valuable to study the trade-off between the two guarantee goals, i.e., availability and performance, and see how the trade-off could be achieved in a content distribution and replication system.

Other directions of future research include investigating the dynamics of Internet-based service applications and the contents shared between the application users in more detail. For example, in a real P2P-based applications such as KaZaA, the number of peers is not fixed, but instead varies in time due to the fact that new peers may join the system at any time or existing

peers may leave the system for ever. Furthermore, new content can be created and published into the system from several peers. When our stochastic model and simulation framework are used to study the replica placement problem in such highly dynamic systems, they should then be extended accordingly to capture these features.

We think, the area of large-scale content distribution and replication in wide-area internetworks and specifically mobile (often wireless) peer-to-peer networking holds much promise for future research. The transient nature of such mobile peer-to-peer networks will require new replication mechanisms and strategies for ensuring content delivery, both with availability and performance guarantees. These kinds of networks are likely to become commonplace and they need to be studied. Accompanying this research, we also need to investigate new replication mechanisms and algorithms to deliver target levels of service availability and even performance with guarantees.

Chapter 8 - References

[ABC+02] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R.P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 1–14. ACM Press, Bosten, Massachusetts, USA, December 2002.

[ABK+01] David Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*.

[ACH98] Christina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A Survey of QoS Architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[AG01] Pascal Anelli and Gwendal Le Grand. Differentiated Services over Shared Media. In *Proceedings of the 9th International Workshop on Quality of Service (IWQoS'01), Karlsruhe, Germany*, pages 288–293, Lecture Notes in Computer Science Vol. 2092, June 2001.

[AGH94] A.Campbell, G.Coulson, and D. Hutchison. A Quality of Service Architecture. *ACM Computer Communications Review*, 24(2):6–27, 1994.

[Aka04] Akamai. http://www.akamai.com.

[Apa04] Apache Web Server. http://httpd.apache.org/docs/.

[AR98] A. Aggarwal and M. Rabinovich. Performance of Dynamic Replication Schemes for an Internet Hosting Service. Technical report, AT&T Labs, October 1998. Available from http://citeseer.nj.nec.com/aggarwal98performance.html.

[Bar92] Yair Bartal. Competitive Algorithms for Distributed Data Management. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 39–50, Victoria, B.C. Canada, May 1992.

[Bar96] Yair Bartal. Probabilistic Approximations of Metric Spaces and its Algorithmic Applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–193, Burlington, VT, October 1996.

[BAZ+97] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura, Viren Shah, and Zongming Fei. Application-Layer Anycasting. In *Proceedings of the 16th IEEE Conference on Computer Communications (INFOCOM'96), Volume 3*, pages 1388–1396, Kobe, Japan, April 1997.

[BBD+98] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. RFC 2475 - An Architecture for Differentiated Services. Experimental RFC, December 1998.

[BCD+01] A. Biliris, C. Cranor, F. Douglis, M. Rabinovich, S. Sibal, O. Spatscheck, and W. Sturm. CDN Brokering. In *Proceedings of Sixth International Workshop on Web Caching and Content Distribution (WCW'01)*, Bosten, MA, USA, June 2001.

[BCF+99] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'99)*, pages 126–134, New York, NY, USA, March 1999.

[BCG+02] A. Barbir, B. Cain, M. Green, M. Hofmann, R. Nair, and O. Spatscheck. Known CN Request-Routing Mechanisms. Internet Draft (work in progress), May 2002. draft-ietf-cdi-known-request-routing-01.txt.

[BCS94] Robert Braden, David Clark, and Scott Shenker. RFC 1633 - Integrated Services in the Internet Architecture: an Overview. Informational RFC, June 1994.

[BDK+02] M. Bhide, P. Deolasse, A. Katker, A. Panchgupte, K. Ramamritham, and P. Shenoy. Adaptive Push-Pull: Disseminating Dynamic Web Data. *IEEE Transactions on Computers, Special Issue on Quality of Service*, May 2002.

[BG95] Kenneth P. Birman and Bradford B. Glade. Reliability Through Consistency. *IEEE Software*, pages 29–41, May 1995.

[BGM+97] E. Borowsky, R. Golding, A. Merchant, L. Schreier, E. Shriver, M. Spasojevic, and J. Wilkes. Using Attribute-Managed Storage to Achieve QoS. In *Proceedings of the 5th International Workshop on Quality of Service (IWQoS'97)*, pages 75–91, Columbia, NY, USA, June 1997.

[BKR+02] Rebecca Braynard, Dejan Kostic, Adolfo Rodriguez, Jeffrey Chase, and Amin Vahdat. Opus: an Overlay Peer Utility Service. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH'02)*, June 2002.

[BMSV02] R. Bhagwan, D. Moore, S. Savage, and G. Voelker. Replication Strategies for Highly Available Peer-to-Peer Storage. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo'02)*, Bertinoro, Italy, June 2002.

[BPLS02] Elias Balafoutis, Antonis Panagakis, Nikolaos Laoutaris, and Ioannis Stavrakakis. The Impact of Replacement Granularity on Video Caching. In *Proceedings of the 2nd IFIP-TC6 Networking Conference*, pages 214–225. IEEE/IFIP, Pisa, Italy, May 2002.

[Bre01] Eric A. Brewer. Lessons from Giant-Scale Services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[BRL01] Randal C. Burns, Robert M. Rees, and Darrell D.E. Long. Efficient Data Distribution in a Web Server Farm. *IEEE Internet Computing*, 5(4):56–65, 2001.

[BSS00] Andre Brinkmann, Kay Salzwedel, and Christian Scheideler. Efficient, Distributed Data Placement Strategies for Storage Area Networks (Extended Abstract). In *Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures, (SPAA'00)*, pages 119–128, June 2000.

[BW01] Mudashiru Busari and Carey L. Williamson. On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'01)*, pages 1225–1234, Anchorage, Alaska, USA, April 2001.

[Cam98] Richard Campbell. *Managing AFS: The Andrew File System*. Pearson Education POD, 1998. ISBN 0-13-802729-3.

[CAMN02] Francisco Matias Cuenca-Acuna, Richard P. Martin, and Thu D. Nguyen. Autonomous Replication for High Availability in Unstructured P2P Systems. Technical Report DCS-TR-509, Department of Computer Science, Rutgers University, November 2002. Available from http://www.cs.rutgers.edu/ mcuenca.

[CDK01] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems - Concepts and Design*. Addison-Wesley Publishers, 3rd edition, 2001. ISBN 0201-61918-0.

[CFKL95] Pei Cao, Edward W. Felten, Anna Karlin, and Kai Li. A Study of Integrated Prefetching and Caching Strategies. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 188–197, Ottawa, Ontario, Canada, June 1995.

[CFK+01] 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*, 23:187–200, 2001. Available from http://www.globus.org/.

[CGS03] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Proceedings of the 11th International Workshop on Quality of Service (IWQoS'03), Berkeley, CA, USA*, pages 381–398, Lecture Notes in Computer Science Vol. 2707, June 2003.

[Chu99] John Chung-I Chuang. Resource Allocation for stor-serv: Network Storage Services with QoS Guarantees. In *Proceedings of the Nework Storage Symposium*, October 1999.

[CI97] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, pages 193–206, Monterey, CA, December 1997.

[Cid01] Cidera. http://www.cidera.com/.

[CKK02] Y. Chen, R. Katz, and J. Kubiatowicz. Dynamic Replica Placement for Scalable Content Delivery. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, March 2002.

[Cla99] Ian Clarke. A distributed decentralised information storage and retrieval system. Technical Report Master's Thesis, University of Edinburgh, June 1999. Online at http://freenet.sourceforge.net/.

[CO02] L.Y. Cao and M.T. Oezsu. Evaluation of Strong Consistency Web Caching Techniques. *World Wide Web: Internet and Web Information Systems*, 5(2):95–123, 2002.

[Cor98] Oracle Corporation. Oracle 8i Advanced Replication. Oracle Technical White Paper, Oracle Parkway, Redwoord City, CA, USA, November 1998.

[CS00] J. Chuang and M. Sirbu. Distributed Network Storage with Quality-of-Service Guarantees. *Journal of Network and Computer Applications*, 23(3):163–185, July 2000.

[CS02] Edith Cohen and Scott Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the ACM Applications, Technologies, Architectures, and Protocols for Computer Communication Conference (SIGCOMM'02)*, pages 177–190, Pittsburgh, PA, USA, August 2002.

[CSR+98] Michel Cukier, Richard E. Schantz, Jennifer Ren, Chetan Sabnis, David Henke, Jessica Pistole, William H. Sanders, David E. Bakken, Mark E. Berman, and David A. Karr. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SDRS'98)*, West Lafayette, Indiana, USA, October 1998.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA*, pages 311–320, Lecture Notes in Computer Science Vol. 2009, 2000.

[CW99] Fabian A. Chudak and David P. Williamson. Improved Approximation Algorithms for Capacitated Facility Location Problems. In *Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization*, pages 99–113, Lecture Notes in Computer Science Vol. 1610, June 1999.

[Dav01] Brian D. Davison. A Web Caching Primer. *IEEE Internet Computing*, 5(4):38–45, 2001.

[DBB+93] A. Danthine, O. Bonaventure, Y. Baguette, G. Leduc, and L. Leonard. QoS Enhancements and the New Transport Service. In *Local Network Interconnection*, Eds.: R.O. Onvural, A. Nilsson, Plenum Press, NY, October 1993.

[DCGN03] Michael Dahlin, Bharat Chandra, Lei Gao, and Amol Nayate. End-to-end WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, April 2003.

[DDTT03] Mathilde Durvy, Christophe Diot, Nina Taft, and Patrick Thiran. Network Availability Based Service Differentiation. In *Proceedings of the 11th International Workshop on Quality of Service (IWQoS'03), Berkeley, CA, USA*, pages 305–324, Lecture Notes in Computer Science Vol. 2707, June 2003.

[DG00] Hans Domjan and Thomas R. Gross. Extending a Best-Effort Operating System to Provide QoS Processor Management. In *Proceedings of the 9th International Workshop on Quality of Service (IWQoS'01), Karlsruhe, Germany*, pages 92–106, Lecture Notes in Computer Science Vol. 2092, June 2000.

[Dig01] Digital Island. http://www.digitalisland.com.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*.

[DMP+02] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[DWAP94] Michael Dahlin, Randolph Wang, Thomas E. Anderson, and David A. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI'94)*, pages 267–280, Monterey, CA, USA, November 1994.

[EJ97] T. Erlebach and K. Jansen. Off-line and On-line Call-Scheduling in Stars and Trees. In *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes on Computer Science Vol. 1335, Springer-Verlag, 1997.

[EKK97] A. Eickler, A. Kemper, and D. Kossmann. Finding Data in the Neighborhood. In *Proceedings of the 23rd Conference on Very Large Data Bases (VLDB'97)*, pages 336–345, Athens, Greece, August 1997.

[Erl02] T. Erlebach. Call Admission Control for Advance Reservation Requests with Alternatives. In *Proceedings of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARANCE'02)*, Rome, Italy, 2002.

[Fas02] FastTrack. http://www.fasttrack.nu/.

[FAZ99] Zongming Fei, Mostafa H. Ammar, and Ellen W. Zegura. Optimal Allocation of Clients to Replicated Multicast Servers. In *Proceedings of the 1999 IEEE International Conference on Network Protocols*, pages 69–76, Toronto, Canada, November 1999.

[FBZA98] Zongming Fei, Samrat Bhattacharjee, Ellen W. Zegura, and Mostafa H. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *Proceedings of the IEEE Conference on Computer Com-*

munications (INFOCOM'98), Volume 2, pages 783–791, San Francisco, California, USA, March 1998.

[FC02] M. Feldman and J. Chuang. Service Differentiation in Web Caching and Content Distribution. In *Proceedings of the IASTED International Conference on Communications and Computer Networks, Cambridge MA, USA*, November 2002.

[FCAB00] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law Relationships of the Internet Topology. *Computer Communication Review*, 29(4):251–262, 1999.

[FGC+97] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP'97)*, pages 78–91, 1997.

[FGM+99] R.T. 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.

[For01] HP Forum. Providing Open Architecture High Availability Solutions, Revision 1.0. Avalable from http://www.mcg.mot.com/us/products/solutions/ha_solutions.pdf, February 2001.

[GDN+03] Lei Gao, Michael Dahlin, Amol Nayate, Jiandan Zheng, and Arun Iyengar. Application Specific Data Replication for Edge Services. In *Proceedings of the 2003 International World Wide Web*, pages 449–460, Budapest, Hungary, May 2003.

[GHOS96] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The Dangers of Replication and a Solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, June 1996.

[GJ79] M.R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[GJMT03] Pawan Goyal, Divyesh Jadav, Dharmendra S. Modha, and Renu Tewari. CacheCOW: QoS for Storage System Caches. In *Proceedings of the 11th International Workshop on Quality of Service (IWQoS'03), Berkeley, CA, USA*, pages 498–515, Lecture Notes in Computer Science Vol. 2707, June 2003.

[GMM01] S. Guha, A. Meyerson, and K. Munagala. Improved Algorithms for Fault Tolerant Facility Location. In *Proceedings of 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 636–641, 2001.

[Gnu04] Gnutella. http://www.gnutellaforums.com/.

[GRH02] Rahul Garg, Ramandeep S. Randhawa, and Huzur. A SLA Framework for QoS Provisioning and Dynamic Capacity Allocation. In *Proceedings of the 10th International Workshop on Quality of Service (IWQoS'02)*, Miami Beach, USA, May 2002.

[Gri00] Carsten Griwodz. *Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, June 2000.

[Gro98] Object Management Group. The Common Request Broker: Architecture and Specification, Revision 2.2, February 1998.

[GRR+98] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98): Workshop on Mobile Data Access*, pages 254–263, November 1998.

[GS96] James Gwertzman and Margo Seltzer. World-Wide Web Cache Consistency. In *Proceedings of 1996 USENIX Technical Conference*, pages 141–151, San Diego, CA, USA, January 1996.

[GWP99] B. Gröenvall, A. Westerlund, and S. Pink. The Design of a Multicast-based Distributed File System. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 251–264. ACM Press, New Orleans, Louisiana, USA, February 1999.

[Hec04] Oliver Heckmann. A Classification of Internet Service Providers. Technical Report TR-KOM-2004-01, Multimedia Communication Lab, Darmstadt University of Technology, January 2004.

[Hen99] John Hennessy. The Future of Systems Research. *IEEE Computer*, 32(8):27–33, August 1999.

[HKRZ02] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed Data Location in a Dynamic Network. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures, (SPAA'02)*, pages 41–52, August 2002.

[HLP91] Jay M. Hyman, Aurel A. Lazar, and Giovanni Pacifici. Real-Time Scheduling with Quality of Service Constraints. *IEEE Journal of Selected Areas in Communications*, 9(7):1052–1063, 1991.

[HPSS03] Oliver Heckmann, Michael Piringer, Jens B. Schmitt, and Ralf Steinmetz. Generating Realistic ISP-Level Network Topologies. *IEEE Communications Letters*, 7(7):335–336, July 2003.

[HSM95] Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta. *Fundamentals of Data Structures in C++*. W.H. Freeman and Company, 1995. ISBN 0-7167-8292-8.

[IBM99] IBM. DB2: Replication Guide and Reference. Number SC26-9642-00, New Orchard Road, Armonk, NY, USA, June 1999.

[INK03] Inktomi. http://www.inktomi.com.

[IRD02] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A Scalable Peer-to-Peer Web Cache. In *Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC'02)*, Monterey, CA, USA, July 2002.

[IS99] Ayal Itzkovitz and Assaf Schuster. MultiView and Millipage - Fine-Grain Sharing in Page-Based DSMs. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 215–218, New Orleans, Louisiana, USA, ACM Press, February 1999.

[JAC98] Anand Manikutty Jussara Almeida, Mihaela Dabu and Pei Cao. Providing Differentiated Levels of Service in Web Content Hosting. In *Proceedings of the ACM SIGMETRICS Workshop on Internet Server Performance (WISP'98), Madison, Wisconsin*, pages 91–102, 1998.

[JCDK01] Kirk L. Johnson, John F. Carr, Mark S. Day, and M. Frans Kaashoek. The Measured Performance of Content Distribution Networks. *Computer Communications*, 24(2):202–206, February 2001. Proceedings of WCW'00.

[JGPH91] T.W. Page Jr., R.G. Guy, G.J. Popek, and J.S. Heidemann. Architecture of the Ficus Scalable Replicated File System. Technical Report CSD-910005, UCLA, USA, March 1991.

[JJJ+00] Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. On the Placement of Internet Instrumentation. In *Proceedings of the 19th IEEE Conference on Computer Communications (INFOCOM'00)*, pages 295–304, Tel Aviv, Israel, March 2000.

[JJK+01] Sugih Jamin, Cheng Jin, A.R. Kurc, Danny Raz, and Yuval Shavitt. Constrained Mirror Placement on the Internet. In *Proceedings of the 20th IEEE Conference on Computer Communications (INFOCOM'01)*, pages 31–40, April 2001.

[JMS02] K. Jain, M. Mahdian, and A. Saberi. A New Greedy Approach for Facility Location Problems. In *Proceedings of the 34th ACM Symposium on Theory of Computation*, pages 731–740, May 2002.

[Kan02] Jussi A. T. Kangasharju. *Internet Content Distribution*. PhD thesis, University of Nice - Sophia Antipolis, France, April 2002.

[Kaz04] KaZaA. http://www.kazaa.com.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, pages 190–201. ACM, November 2000.

[KKM02] Magnus Karlsson, Christos Karamanolis, and Mallik Mahalingam. A Framework for Evaluating Replica Placement Algorithms. Technical Report HPL-

2002, HP Laboratories, Palo Alto, CA, USA, July 2002. Available from http://www.hpl.hp.com/personal/Magnus_Karlson.

[KM02] Magnus Karlsson and Mallik Mahalingam. Do We Need Replica Placement Algorithms in Content Delivery Networks. In *Proceedings of the 7th International Workshop on Web Caching and Content Distribution, (WCW'02)*, pages 117–128, Boulder, Colorado, USA, August 2002.

[KOM02] KOMSSYS. http://komssys.sourceforge.net/.

[KPR99] Madhukar Korupolu, Greg Plaxton, and Rajmohan Rajaraman. Placement Algorithms for Hierarchical Cooperative Caching. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 586–595, January 1999.

[KRR02] J. Kangasharju, J. Roberts, and K. Ross. Object Replication Strategies in Content Distribution Networks. *Computer Communications*, 25(4):367–383, March 2002. Proceedings of WCW'01, Boston, MA, USA.

[KRS00] P. Krishnan, Danny Raz, and Yuval Shavitt. The Cache Location Problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, October 2000.

[KRT02] Jussi Kangasharju, Keith W. Ross, and David A. Turner. Optimal Content Replication in P2P Communities. Manuscript, 2002.

[KSC02] S. Krishnamurthy, W. H. Sanders, and M. Cukier. Performance Evaluation of a QoS-Aware Framework for Providing Tunable Consistency and Timeliness. In *Proceedings of the 10th International Workshop on Quality of Service (IWQoS'02)*, pages 214–223, Miami Beach, USA, May 2002.

[Kue97] Geoffrey H. Kuenning. *Seer: Predictive File Hoarding for Disconnected Mobile Operation*. PhD thesis, University of California, Los Angeles, CA, USA, May 1997.

[LALT02] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services. In *Proceedings of the 10th International Workshop on Quality of Service (IWQoS'02)*, Miami Beach, USA, May 2002.

[LBCM03] Anukool Lakhina, John W. Byers, Mark Crovella, and Ibrahim Matta. On the Geographic Location of Internet Resources. *IEEE Journal on Selected Areas in Communications*, 21(6):934–948, August 2003.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, June 2002. ACM Press. ISBN 1-58113-483-5.

[LED02] LEDA - the Library of Efficient Data Types and Algorithms. Algorithmic Solutions Software GmbH. Available from http://www.algorithmic-solutions.com/.

[LGID99] B. Li, M. Golin, G. Italiano, and X. Deng. The Optimal Placement of Web Proxies in the Internet. In *Proceedings of the 18th IEEE Conference on Computer Communications (INFOCOM'99)*, pages 1282–1290, March 1999.

[LGO+99] Michael Liepert, Carsten Griwodz, Giwon On, Michael Zink, and Ralf Steinmetz. A Distributed Media Server for the Support of Multimedia Teaching. In *Multimedia Systems and Applications II, Boston, MA*, pages 452–463. SPIE, August 1999.

[LM97] C. Lucet and J.-F. Manouvrier. Exact Method to Compute Network Reliability. In *Proceedings of the 1st Mathematical Methods in Reliability*, Bucharest, Roumanie, September 1997.

[LN00] Alistair G. Lowe-Norris. *Mircosoft Windows 2000 Active Directory.* O'Reilly & Associates, Inc., 2000.

[Mau02] Andreas Mauthe. Content Management and Delivery - Related Technology Areas. Technical Report MPG-03-04, Computing Department, Lancaster University, UK, October 2002.

[MCH+01] A. Myers, J. Chuang, U. Hengartner, Y. Xie, W. Zhuang, and H. Zhang. A Secure, Publisher-Centric Web Caching Infrastructure. In *Proceedings of IEEE INFOCOM 2001*, Anchorage AL, USA, April 2001.

[MDZ99] Andy Myers, Peter A. Dinda, and Hui Zhang. Performance Characteristics of Mirror Servers on the Internet. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99), Volume 1*, pages 304–312, New York, NY, USA, March 1999.

[MH97] Nenad Mladenovic and Pierre Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24(7):1097–1100, 1997.

[MH00] M. Mauve and V. Hilt. An Application Developer's Perspective on Reliable Multicast for Distributed Interactive Media. *Computer Communications Review*, 30(3):28–38, July 2000.

[Mir04] Mirror Image Internet. http://www.mirror-image.com/.

[MLH00] Nenad Mladenovic, M. Labbe, and Pierre Hansen. Solving the p-Center Problem with Tabu Search and Variable Neighbourhood Search, July 2000. Available from http://www.crt.umontreal.ca/.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'01)*, pages 346–353, Cincinnati, Ohio, USA, August 2001. IEEE.

[MMGC02] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 31–44. ACM Press, Bosten, Massachusetts, USA, December 2002.

[MO02] Z. Miao and A. Ortega. Scalable Proxy Caching of Video Under Storage Constraints. *IEEE Journal on Selected Areas in Communications*, 20(7):1315–1327, September 2002.

[Moj02] Mojonation. http://www.mojonation.net/.

[MR95] Salvatore T. March and Sangkyu Rho. Allocating Data and Operations to Nodes in Distributed Database Design. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):305–317, April 1995.

[MS03] Jan Mischke and Burkhard Stiller. Specification of a Scalable Peer-to-Peer Search Infrastructure. Technical Report TIK Report Nr. 176, The Federal University of Technology (ETH), Zurich, Switzerland, July 2003. Available from ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-176.pdf.

[MT04] Andreas Mauthe and Peter Thomas. *Professional Content Management Systems - Handling Digital Media Assets*. John Wiley & Sons, Ltd, 2004. ISBN 0-470-85542-8.

[Nap00] Napster. http://www.napster.com.

[Nin01] A. Ninan. Maintaining Cache Consistency in Content Distribution Networks. Master's thesis, Department of Computer Science, University of Massachusetts, June 2001.

[NKS+02] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *Proceedings of the 11th WWW Confernece*, Honolulu, Hawaii, USA, May 2002.

[NLA02] National Laboratory for Applied Network Research (NLANR). http://moat.nlanr.net/rawdata/.

[NS95] Klara Nahrstedt and Ralf Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer*, 28(5):52–63, 1995.

[NS96] Klara Nahrstedt and Jonathan M. Smith. Design, Implementation, and Experiences of the OMEGA End-Point Architecture. *IEEE Journal on Selected Areas in Communications*, 14(7):1263–1279, September 1996.

[OLKC97] Giwon On, Bum-Sik Lee, Hak-Yeong Kim, and Dong-Hae Chi. QoS-based Session and Resource Management in Retrieval Multimedia Services. In *Proceedings of the 1st International Conference on Next Generation Communication Software (NCS'97)*, pages 177–181, December 1997.

[OLKC98] Giwon On, Bum-Sik Lee, Hak-Yeong Kim, and Dong-Hae Chi. Providing Quality of Service Negotiation in an Interactive Video on Demand System. In *Proceedings of the International Technical Conference on Circuits/Systems, Computers and Communications*, pages 521–524, July 1998.

[OSS01] Giwon On, Jens Schmitt, and Ralf Steinmetz. Design and Implementation of a QoS-aware Replication Mechanism for a Distributed Multimedia System. In

Proceedings of the International Workshop on Interactive Distributed Multimedia Systems, (IDMS'01), Lancaster, UK, pages 38–49, Lecture Notes in Computer Science Vol. 2158, September 2001.

[Pat96] Steve D Pate. *UNIX Internals - A Practical Approach*. Addison-Wesley Longman, 1996. ISBN 0-201-87721-X.

[PB02] Stefan Podlipnig and Laszlo Böszörmenyi. Replacement Strategies for Quality Based Video Caching. In *International Conference on Multimedia and Expo, Lausanne, Switzerland*, pages 49–52, August 2002.

[PB03] Stefan Podlipnig and Laszlo Boeszoermenyi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):1263–1279, December 2003.

[Pea84] Judea Pearl. *Heuristics - Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Co., 1984. ISBN 0-201-05594-5.

[PF95] Vern Paxson and Sally Floyd. Wide-area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[PKvST00] Guillaume Pierre, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum. Differentiated Strategies for Replicating Web Documents. *Computer Communications*, 24(2):232–240, 2000. In WCW'2000, Lisbon, Portugal.

[PQ00] N. Padmanabhan and Lili Qui. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *Proceedings of the ACM SIGCOMM Symposium on Communications Architectures and Protocols (SIGCOMM'00)*, pages 111–123, Stockholm, Sweden, August 2000.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall Inc., 1982. ISBN 0-13-152462-3.

[PSF+01] Christopher R. Palmer, Georgos Siganos, Michalis Faloutsos, Christos Faloutsos, and Phillip B. Gibbons. The Connectivity and Fault-Tolerance of the Internet Topology. In *Proceedings of International Workshop on Network-Related Data Management (NRDM-2001)*, 2001.

[PUW00] C.A. Phillips, R. Uma, and J. Wein. Off-line Admission Control for General Scheduling Problems. *Journal of Scheduling*, 3:365–381, 2000.

[PvS01] Guillaume Pierre and Maarten van Steen. Globule: a Platform for Self-Replicating Web Documents. In *Proceedings of the 6th International Conference on Protocols for Multimedia Systems*, pages 1–11, Lecture Notes in Computer Science Vol. 2213, October 2001.

[QPV01] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the Placement of Web Server Replicas. In *Proceedings of the 20th IEEE Conference on Computer Communications (INFOCOM'01)*, pages 1587–1596, Anchorage, Alaska, USA, April 2001.

[QSS02] Q. Lv, S. Ratnasamy, and S. Shenker. Can Heterogeneity make Gnutella Scalable? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, USA, March 2002.

[RA99] Michael Rabinovich and Amit Aggarwal. RaDaR: A Scalable Architecture for a Global Web Hosting Service. *Computer Networks (Amsterdam, Netherlands, 1999)*, 31(11):1545–1561, 1999. Also published in WWW'98, Toronto, Canada, 1998.

[Rab98] Michael Rabinovich. Issues in Web Content Replication. *IEEE Data Engineering Bulletin, Invited Paper*, 21(4):21–29, December 1998.

[Rat95] David H. Ratner. Selective Replication: Fine grain control of replicated files. Master's thesis, University of California, Los Angeles, CA, USA, March 1995. Available as UCLA technical report CSD-950007.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01), Heidelberg, Germany*, pages 329–350, Lecture Notes in Computer Science Vol. 2218, Springer-Verlag, Berlin, November 2001.

[RD01b] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, Chateau Lake Louise, Banff, Canada, October 2001.

[REG+03] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of the 2nd USENIX File and Storage Technologies (FAST'03)*, Berkeley, CA, USA, March 2003. USENIX.

[Ren01] Jennifer Ren. *AQuA: A Framework for Providing Adaptive Fault Tolerance to Distributed Applications*. PhD thesis, University of Illinois at Urbana-Champaign, IL, USA, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proceedings of the 18th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*.

[RGE02] P. Radoslavov, R. Govindan, and D. Estrin. Topology-Informed Internet Replica Placement. *Computer Communications*, 25(4):384–392, March 2002. Proceedings of WCW'01, Boston, MA, USA.

[RHKS02] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, New York, NY, USA, June 2002.

[RIF02] Kavitha Ranganathan, Adriana Iamnitchi, and Ian Foster. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer

Communities. In *Global and Peer-to-Peer Computing in Large-Scale Distributed Systems Workshop*, Berlin, Germany, May 2002.

[RRP99] D. Ratner, P. Reiher, and G. Popek. Roam: A Scalable Replication System for Mobile Computing. In *Proceedings of Workshop on Mobile Databases and Distributed Systems (MDDS'99)*, September 1999.

[RS02] Michael Rabinovich and Oliver Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002. ISBN 0-201-61570-3.

[SBL99] Yasushi Saito, Brian N. Bershad, and Henry M. Levy. Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 1–15, South Carolina, USA, ACM Press, December 1999.

[Sch01a] Jens B. Schmitt. *Heterogeneous Network Quality of Service Systems*. Kluwer Academic Publishers, 2001. ISBN 0-7923-7410-X.

[Sch01b] Henning Schulzrinne. Keynote: Quality of Service - 20 Years Old and Ready to Get a Job? In *Proceedings of the 9th International Workshop on Quality of Service (IWQoS'01), Karlsruhe, Germany*, page 1, Lecture Notes in Computer Science Vol. 2092, June 2001.

[SEQ01] Project SEQUIN. Quality of Service Definition. SEQUIN Deliverable D2.1, April 2001. Available from http://www.dante.net/sequin/QoS-def-Apr01.pdf.

[SFFF03] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. Power Laws and the AS-Level Internet Topology. *IEEE/ACM Transactions on Networking*, 11(4):514–524, August 2003.

[SGD+02] Stefan Saroiu, Krishna P. Gummadi, Richard Dunn, Steven D. Gribble, and Henry M. Levy. An Analysis of Internet Content Delivery Systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 315–327, Boston, MA, USA, December 2002.

[SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, USA, January 2002.

[SGK+85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the Summer 1985 USENIX Conference*.

[SKK+90] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies,*

Architectures, and Protocols for Computer Communications, pages 149–160, San Diego, California, United States, ACM Press, 2001.

[SN04] Ralf Steinmetz and Klara Nahrstedt. *Multimedia Systems*. Springer-Verlag, 2004. ISBN 3-540-40867-3.

[SPG97] Scott Shenker, Craig Partridge, and Roch Guerin. RFC 2212 - Specification of Guaranteed Service. Standards Track RFC, September 1997.

[SQU04] Squid Web Proxy Cache. http://www.squid-cache.org/.

[SS90] M. Satyanarayanan and E.H. Siegel. Parallel Communication in a Large Distributed Environment. *IEEE Transactions on Computers*, 39(3):328–348, March 1990.

[ST00] A. Shaikh and R. Tewari. On the Effectiveness of DNS-based Server Selection. Technical Report IBM Research Report RC 21785, IBM Thomas J. Watson Research Center, 2000.

[ST02] Sherlia Shi and Jonathan Turner. Placing Servers in Overlay Networks. In *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPETS'02)*, July 2002.

[SW97] Ralf Steinmetz and Lars C. Wolf. Quality of Service - Where are We? In *Proceedings of the IFIP 7th International Workshop on Quality of Service (IWQoS'97)*, Invited Paper, New York, USA, May 1997.

[SWCK02] W. Shi, R. Wright, E. Collins, and V. Karamcheti. Workload Characterization of a Personalized Web Site and Its Implications for Dynamic Content Caching. In *Proceedings of the 7th International Workshop on Web Caching and Content Distribution (WCW'02)*, pages 1–16, New York, NY, USA, August 2002.

[TA92] Raj Tewari and Nabil R. Adam. Distributed File Allocation with Consistency Constraints. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, pages 408–415, June 1992.

[Tan03] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, 4th edition, 2003. ISBN 0-13-066102-3.

[TDMV96] Renu Tewari, Daniel M. Dias, Rajat Mukherjee, and Harrick M. Vin. High Availability in Clustered Multimedia Servers. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE'96)*, pages 645–654. IEEE Computer Society, February 1996.

[TGJ+01] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network Topologies, Power Laws, and Hierarchy. Technical Report Technical Report 01-746, Computer Science Department, University of Southern California, CA, USA, June 2001.

[TIE01] Tiers Topology Generator. Software available from http://www.isi.edu/nsnam/dist/topogen/.

[TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems - Principles and Paradigms*. Prentice-Hall, 2002. ISBN 0-13-088893-1.

[Vah02] Amin Vahdat. Dynamically Provisioning Distributed Systems to Meet Target Levels of Performance, Availability, and Data Quality. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo'02)*, June 2002.

[Vaz01] Vijay Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001. ISBN 3-540-65367-8.

[VDW01] A. Venkataramani, M. Dahlin, and P. Weidmann. Bandwidth Constrained Placement in a WAN. In *Proceedings of the 20th ACM International Conference on Principles of Distributed Computing Systems (PODC'01)*, August 2001.

[VTF01] S. Vazhkudai, S. Tuecke, and I. Foster. Replica Selection in the Globus Data Grid. In *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID'01), Anchorage, Alaska, USA*, pages 106–113, IEEE Computer Society Press, May 2001.

[Wan99] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, 25(9):36–46, 1999.

[Wan01] Zheng Wang. *Internet QoS - Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, 2001. ISBN 1-55860-608-4.

[WC97] Duane Wessels and Kim Claffy. RFC 2186 - Internet Cache Protocol (ICP), version 2. Informational RFC, September 1997. Available from http://www.faqs.org/rfcs/rfc2186.html.

[Wei94] Mark A. Weiss. *Data Structures and Algorithm Analysis in C++*. The Benjamin/Cummings Publishing Co., 1994. ISBN 0-8053-5443-3.

[Wet99] David Wetherall. Active Network Vision and Reality: Lessons from a Capsule-based Systeme. In *Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP'99)*, pages 64–79, Kiawah Island, SC, December 1999.

[WNB97] A. Weigmann, Jorg Nonnenmacher, and Ernst Biersack. Center Placement Algorithms for Large Multicast Groups. In *Proceedings of the 7th IFIP Conference on High Performance Networking (HPN'97)*, pages 18–35, April 1997.

[Wol96] Lars C. Wolf. *Resoure Management for Distributed Multimedia Systems*. Kluwer Academic Publishers, 1996. ISBN 0-7923-9748-7.

[WPP02] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirection on CDN Robustness. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 345–360. ACM Press, Bosten, Massachusetts, USA, December 2002.

[WPS+00] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding Replication in Databases and Distributed Systems. In *Proceedings of the*

20th International Conference on Distributed Computing Systems (ICDS'2000), pages 264–274, Taipei, Taiwan, R.O.C., April 2000.

[YAL99] J. Yin, L. Alvisi, and C. Lin. Volume Leases for Consistency in Large-Scale Systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):563–576, September 1999.

[YV02] Haifeng Yu and Amin Vahdat. Minimal Replication Cost for Availability. In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC'02)*, pages 98–107, Monterey, California, USA, 2002.

[ZCB96] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to Model an Internetwork. In *Proceedings of the 15th IEEE Conference on Computer Communications (INFOCOM'96)*, volume 2, pages 594–602, San Francisco, CA, USA, March 1996. IEEE.

[ZCD97] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, 1997.

[ZDE+93] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.

[ZGJS00] Michael Zink, Carsten Griwodz, Alex Jonas, and Ralf Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In *Proceedings of the Seventh International Conference on Parallel and Distributed Systems, Iwate, Japan*, pages 281–286. IEEE, July 2000.

[Zin03] Michael Zink. *Scalable Internet Video-on-Demand Systems*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, September 2003.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

Chapter 9 - Author's Publications

9.1 Journal Articles

[1] Giwon On, Jens Schmitt, and Ralf Steinmetz. QoS-Controlled Dynamic Replication in Peer-to-Peer Systems. *Praxis der Informationsverarbeitung und Kommunikation*, 03(2):96–101, April 2003. ISSN 0930-5157.

9.2 Conference Contributions

[2] Giwon On, Jens Schmitt, and Ralf Steinmetz. The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P'03)*, pages 57–64. IEEE, September 2003.

[3] Giwon On, Jens Schmitt, and Ralf Steinmetz. Quality of Availability: Replica Placement for Widely Distributed Systems. In *Proceedings of International Workshop on Quality of Service (IWQoS'03), Montrey, CA, USA*, pages 325–342, Springer LNCS 2707, June 2003.

[4] Giwon On, Jens Schmitt, and Ralf Steinmetz. On Availability QoS for Replicated Multimedia Service and Content. In *Proceedings of International Workshop on Interactive Distributed Multimedia Systems 2002 (IDMS-PROMS02), Coimbra, Portugal*, pages 313–326. Springer LNCS 2515, November 2002.

[5] Giwon On, Jens Schmitt, Michael Liepert, and Ralf Steinmetz. Replication with QoS support for a Distributed Multimedia System. In *Proceedings of the 27th EUROMICRO Conference (Workshop on Multimedia and Telecommunications), Warszaw, Poland*. IEEE, September 2001. ISBN 0-7695-1236-4.

[6] Giwon On, Jens Schmitt, and Ralf Steinmetz. Design and Implementation of a QoS-aware Replication Mechanism for a Distributed Multimedia System. In *Proceedings of the Workshop on Interactive Distributed Multimedia Systems 2001 (IDMS'01), Lancaster, UK*, pages 38–49. Springer LNCS 2158, September 2001. ISBN 3-540-42530.

[7] Giwon On, Michael Zink, Michael Liepert, Carsten Griwodz, Jens Schmitt, and Ralf Steinmetz. Replication for a Distributed Multimedia System. In *Proceedings of 8th International Conference on Parallel and Distributed Systems (ICPADS'01), Kyongju City, Korea*, pages 377–384. IEEE, June 2001. ISBN 0-7695-1153-8.

[8] Carsten Griwodz, Michael Liepert, Abdulmotaleb El Saddik, Giwon On, Michael Zink, and Ralf Steinmetz. Perceived Consistency. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, June 2001.

[9] Carsten Griwodz, Michael Zink, Michael Liepert, Giwon On, and Ralf Steinmetz. Multicast for Savings in Cache-based Video Distribution. In *Proceedings of SPIE's Multimedia Computing and Networking Conference 2000 (MMCN'00), San Jose, USA*, pages 26–35. SPIE, January 2000.

[10] Michael Liepert, Carsten Griwodz, Giwon On, Michael Zink, and Ralf Steinmetz. A distributed media server for the support of multimedia teaching. In *Multimedia Systems and Applications II, Boston, MA*, pages 452–463. SPIE, August 1999.

[11] Giwon On, Bum-Sik Lee, Hak-Yeong Kim, and Dong-Hae Chi. Providing Quality of Service Negotiation in an Interactive Video on Demand System. In *Proc. of the International Technical Conference on Circuits/Systems, Computers and Communications*, volume 2, pages 521–524, July 1998.

[12] Giwon On, Dong Hae Chi, and Seok Han Yoon. An Integrated Dynamic and Visual Debugging for Parallel Applications. In *Parallel Computing: Fudamentals, Applications and New Directions*. Elsevier Science B.V. The Netherlands, February 1998. ISBN 0-444-82882-6.

[13] Giwon On, Bum-Sik Lee, Chul-Hee Hong, and Dong-Hae Chi. A Scheme for Building Visual Debugging Environment with Dynamic Debugging Method for Parallel Systems. In *Lecture Notes in Computer Science, Volume 1277 (PaCT-97)*, pages 241–246. Springer-Verlag, September 1997.

[14] Giwon On, Bum-Sik Lee, Chul-Hee Hong, and Dong Hae Chi. Visual Debugging with a Textual/Graphical View Mapper for Parallel Programs. In *Proceedings of International Joint Conference of Information Sciences, Durham, NC, USA*, volume 3, pages 219–222, March 1997.

[15] Giwon On, Bum-Sik Lee, Hak-Yeong Kim, and D.H. Chi. QoS-based Session and Resource Management in Retrieval Multimedia Services. In *Proceedings of the 1st Conference of Next Generation Communication Software (NCS'97)*, pages 177–181, December 1997.

[16] Giwon On, Chung Hoon Kim, and Dong Hae Chi. Application QoS Management for Retrieval Mutimedia Services. In *Proceedings of the 10th KIPS Joint Conference of High-Speed Networks and Multimedia Technologies*, pages 94–98, November 1997.

[17] Giwon On, Bum-Sik Lee, Chul-Hee Hong, Dong Hae Chi, and Ki-Uk Lim. Debugging Parallel Programs. *Korea Information Science Society Review*, 14(6): 63–72, June 1996.

[18] Chul-Hee Hong, Giwon On, Bum-Sik Lee, and Dong Hae Chi. Replay for Debugging MPI Parallel Programs. In *Proceedings of the 2nd MPI Developer's Conference, Notre Dame, Indiana, USA*, pages 156–160, July 1996.

[19] Giwon On, Chul-Hee Hong, Bum-Sik Lee, and Dong Hae Chi. Visual Debugging for Distributed and Parallel Programs. In *Proceedings of INFORNOR'96, Antofagasta, Chile*, pages 97–103, November 1996.

[20] Bum-Sik Lee, Giwon On, Chul-Hee Hong, and Dong Hae Chi. ParaDebug: A Parallel Visual Debugger. In *Proceedings of Asia-Pacific Software Engineering Conference, Seoul, Korea*, Volume II, pages 11–20, December 1996.

[21] Giwon On and Chul-Hee Hong Bum-Sik Lee. Textual/Graphical Mapping-based User Interface for the Graphical Parallel Debugger ParaDebug. In *Proceedings of the 5th KIPS Spring Conference, Korea*, Volume 3, pages 150–155, April 1996. (Korean).

[22] Giwon On, Dong Hae Chi, and Ki-Uk Lim. Specification and Recognition of Execution Patterns on Multithreaded Parallel Programs. In *Proceedings of the 4th KISS Fall Conference*, volume 22, pages 1407–1410, May 1995. (Korean).

[23] Giwon On, Bum-Sik Lee, and Chul-Hee Hong. Parallel Debugging Techniques - A Review. *ETRI Weekly Technology Trends*, TIS95-14, pages 1-18, May 1995. (Korean).

[24] Giwon On, Bum-Sik Lee, Chul-Hee Hong, and Dong Hae Chi. Specification and Error Recognition of Multithreaded Parallel Program Executions on a Distributed and Shared-Memory Computing Environment. *KISS Journal of Parallel Processing Systems*, 6(2):105–115, November 1995.

[25] Giwon On, Dong Hae Chi, and Kil-Rok Oh. Self-Organizing Optimal Process Mapping Strategy. In *Proceedings of the 2nd KIPS Fall Conference, Korea*, volume 1, pages 239–242, October 1994. (Korean).

[26] Giwon On, Bum-Sik Lee, and Chul-Hee Hong. Strategies for Dynamic Load Balancing in Parallel Systems. *ETRI Weekly Technology Trends*, Part I: TIS94-27 (pages 15-31), Part II: TIS94-28 (pages 15-30), July 1994. (Korean).

9.3 Technical Reports

[27] Giwon On, Jens Schmitt, and Ralf Steinmetz. Admission Controlled Replica Placement Problem. Technical Report TR-KOM-2003-09, Darmstadt University of Technology, October 2003.

[28] Giwon On, Jens Schmitt, and Ralf Steinmetz. The Quality of Availability: Tackling the Replica Placement Problem. Technical Report TR-KOM-2001-11, Darmstadt University of Technology, November 2001.

[29] Michael Liepert, Giwon On, Michael Zink, Stefan Hoermann, Carsten Griwodz, and Ralf Steinmetz. LoNode: Distribution for Project kMed. Technical Report TR-KOM-2001-06, Darmstadt University of Technology, May 2001.

[30] Giwon On, Michael Liepert, Michael Zink, and Carsten Griwodz. Replication for medianode. Technical Report TR-KOM-2000-03, Darmstadt University of Technology, September 2000.

[31] Carsten Griwodz, Michael Zink, Michael Liepert, and Giwon On. Analytical Model of Patching VoD Cache Hierarchies. Technical Report 03, KOM TU Darmstadt, September 1999.

Chapter 10 - Appendices

10.1 The Implementation Issues of a Replication Mechanism in medianode

10.1.1 Need of Replication in medianode

Replicating data and services at multiple networked computers increases the service availability of distributed systems. Replication of presentation materials and meta-data is an important key to providing high availability, fault tolerance and QoS in distributed multimedia systems. For example, when a user requires access (read/write) to a presentation material which comprises audio/video data and some resources which are not available in the local machine at this point of time, a local replication manager copies the required data from their original location and puts it into either one of the machines located nearby or the local machine without requiring any user interaction (user transparent). This function enhances the total performance of the distributed system, in this example, the presentation service system, by reducing the response delay that is often caused due to insufficient system resources at a given service time. Furthermore, because of the available replica in the local machine, the assurance that users can continue their presentation in a situation of network disconnection, is significantly higher than without replica.

In the following, the design and implementation issues of a replication mechanism in a distributed multimedia system *medianode* [LGO+99] are presented. The implemented replication mechanism supports the QoS characteristics of multimedia data and the availability of system resources. Each type of data handled and replicated are classified according to their QoS characteristics and replication requirements.

10.1.2 Architectural Overview of medianode

The distributed multimedia system medianode has been developed as a software infrastructure to share multimedia-enhanced teaching materials among lecture groups. The design of the medianode system addresses issues of availability, versioning, access control and consistent replication of data. To support the teachers, it allows for transparent access to shared content, and it enables the teachers to operate in disconnected mode since they do not have access to the network at all times during their presentations. The system architecture of medianode is intended for decentralized operation of the widely distributed system. Within this distributed system, each participating host is called a medianode and conceptually equal to all other participating hosts, i.e. a medianode is not considered as a client or a server. Client or server tasks are taken on by each medianode in the system depending on its resources and software modules. Figure 31 shows the basic model of medianode.

The central element of a medianode is called its core. The core performs two primary tasks: (a) it dynamically loads code which implements the medianode's operations and it instantiates

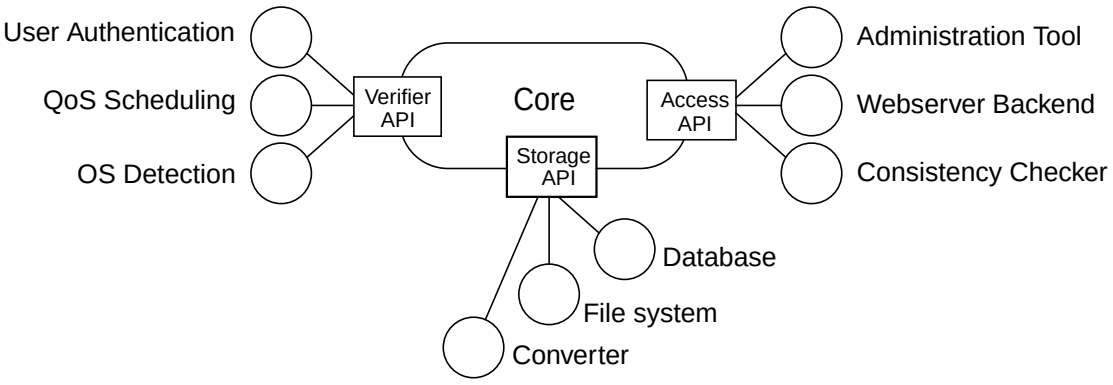


Figure 31: Example modules and their APIs assignment of medianode.

objects; (b) the core implements the routing of requests between the medianode components (called bows) that are instantiated in a medianode.

Each dynamically loaded module implements a child class of medianode's root class, the bow class. Some bows implement basic operations that are necessary for the start of a medianode. They are not loaded dynamically but statistically linked to the medianode binary and well known to the core. The bow class has three abstract subclasses which structure the operations of medianode in general. These subclasses are called Access Bow, Storage Bow and Verifier Bow.

Objects of the class Access Bow implement the visible activity of a medianode: e.g. an HTTP access bow implements means of requesting content from the medianode via the HTTP protocol, a Telnet access bow allows a user to connect to a medianode using the telnet application for basic information and management tasks. Storage Bows implement the functionality of distributed file systems and distributed database. In medianode, such storage bows are always capable of operating in disconnected operation modes. They implement all functionality locally, keep all relevant data locally, and are able to react to requests to unreachable data. Verifier Bows are intended to check the availability and accessibility of data and services that have been requested by access bows or storage bows.

10.1.3 The Replication System Model

10.1.3.1 Scope of our Replication System

By analysing the service requirements of medianode, we identified a number of issues that the design of our replication system need to address:

- **High availability.** The replication system in medianode should enable data/service access in both connected and disconnected operation modes. Users can keep multiple copies of their files on different medianodes that are distributed geographically across several universities in the german state of Hessen.

- **Consistency.** Concurrent updates and system failures can lead to replicas not being consistent any more, i.e. stale state. The replication system should offer mechanisms for both resolving conflicts and keeping consistency between multiple replicas and their updates.
- **Location and access transparency.** Users do not need to know where presentation resources are physically located and how these resources are accessed.
- **Cost efficient update transport.** Due to the limitation of system and network resources, the replication system should use multicast-based transport mechanism for exchanging updates to reduce resource utilization.
- **QoS support.** The specific characteristics of presentational data, especially of multimedia data should be supported by the proposed replication mechanism.

In medianode, we mainly focus on the replication service for accessing data in terms of 'inter-medianode', i.e. between medianodes, by providing replica maintenance in each medianode. Consequently, a replication manager can be implemented as one or a set of medianode's bow instances in each medianode. The replication managers communicate among each other to exchange update information through the whole medianodes. A replication service within a medianode, i.e., 'intra-medianode', is not considered. However, the replication concept implemented in medianode is straightforwardly applicable to the replication service for intra-medianode scope.

10.1.3.2 The Concept of Logically Centralized Database

For a technical realization of our replication system, we use the concept of a "logically centralized database" which especially enables the transparent access to presentation materials. Similar to the concept of location-independent identifiers in distributed database system [EKK97], the logically centralized database enables a mapping between logical and physical resources. With this concept, users have a single access point to the replicated presentation materials in medianode.

Thus, users do not need to know where the presentation resources physically locate and how, especially with which method, the resources should be accessed, and which replica from which physical location should be accessed. The requests from users, either for reading or writing any presentation materials, are first sent to the Access Bow of the local medianode that runs on the user's local machine. (In the early phase, we implemented an Apache web server [Apa04] module (`mod_menot.c`) as front-end access bow through which the user's requests are sent to medianode.) After successful check of the accessibility of the user and the availability of the requested resources, the corresponding storage bows send the target data to the users. Figure 32 illustrates the interface point, the bows building the logically centralized database as well as the interactions between the bows. Some additional remarks on the logically centralized database are in order:

- According to the data types, all of the presentation contents and their meta-data are stored in corresponding storage bows.
- The 'front-end' of the storage bow API provides unique interface functions, independent of the data types: this is similar to the VFS (virtual file system) interface in UNIX systems [Pat96].

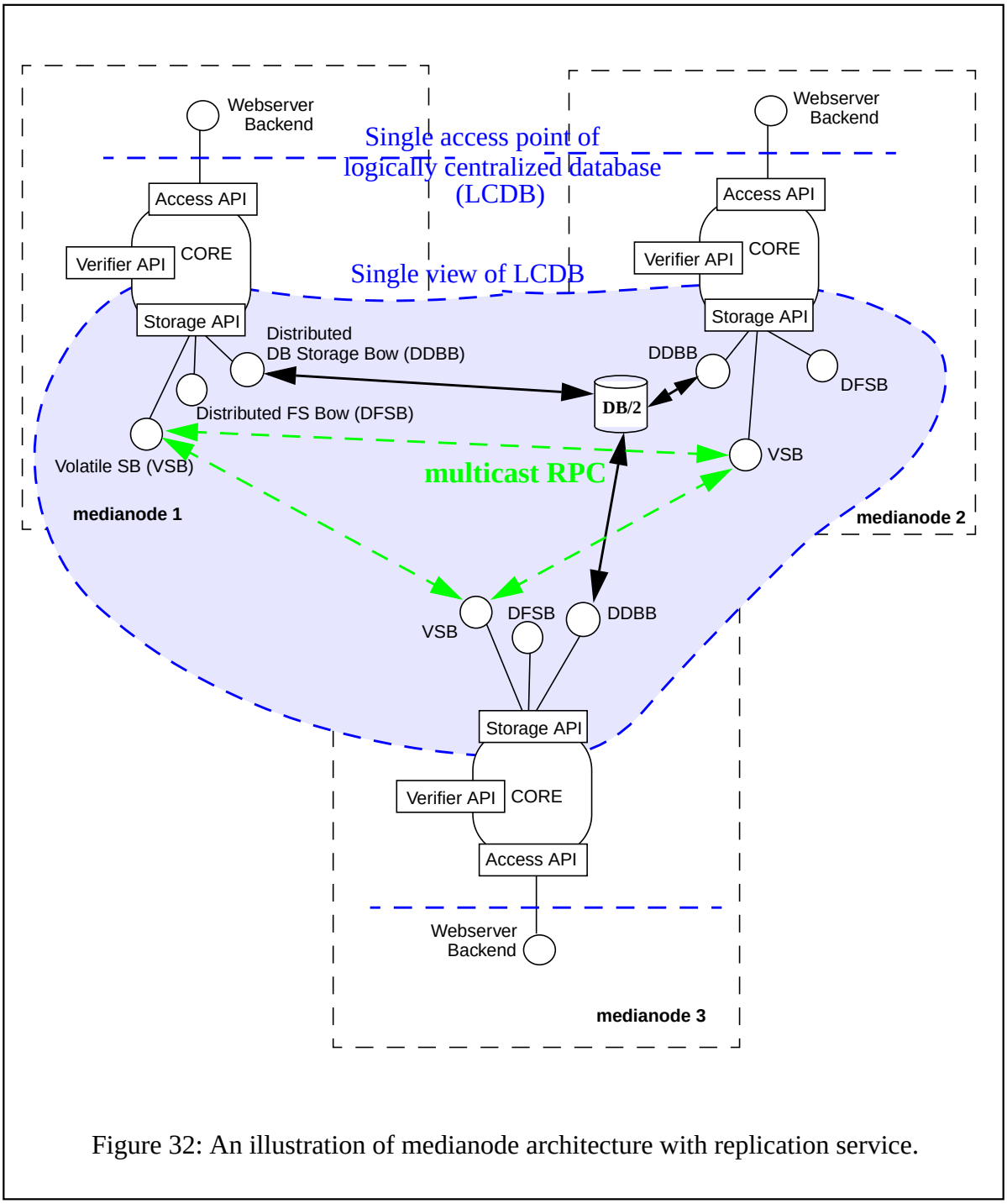


Figure 32: An illustration of medianode architecture with replication service.

- Replication has to be supported for most storage bows, although the number of replicas and the update frequency may differ between the individual bows.
- For the update propagation between replication managers, a multicast RPC (remote procedure call) [SS90] communication mechanism is used.

10.1.3.3 Different Types of Presentation Data

Data organization comprises the storage of content data as well as meta information about this content data in a structured way. The typical data types which can be identified in medianode are the following:

- Presentation contents: this type of data comprises text, image, audio/video files and can be stored in file systems which should handle automatic data distribution and access, and also support the multimedia characteristics of this content type.
- Presentation description data files.
- Meta-data of user, system, domain, and organization information. User's title, group, system platform, and university are examples for this meta-data category.
- Meta-data of system resource usage information such as memory usage, number of threads running within medianode process, number of loaded bows.
- Meta-data of user session and token information.

10.1.3.4 Classification of Target Replicas

The main goal of replication is to increase the high availability of medianode's services and to decrease the response time for accesses to data located on other medianodes. To meet this goal, data which is characterized by a high availability requirement should be replicated among the running medianodes. We classify different types of target replicas according to their granularity (data size), requirement of QoS support, update frequency and whether their data type is 'persistent' or not ('volatile'). Indeed, there are three classes of replicas in medianode:

- Metareplicas (replicated metadata objects) that are persistent and of small size. An example would be a list medianodes (sites) which currently contain an up-to-date copy of a certain file. This list itself is replicated to increase its availability and improve performance. A metareplica is a replica of this list.
- Softreplicas which are non-persistent and of small size. This kind of replicas can be used for reducing the number of messages exchanged between the local and remote medianodes, and thereby reducing the total service response time. I.e., if a local medianode knows about the available local system resources, then the local replication manager can copy the desired data into the local storage bow, and the service that is requested from users which requires exactly the data can be processed in a shorter response time. Information about the available system resource, user session and the validity of user tokens are replicas of this type.
- Truereplicas which are persistent and of large size. Content files of any media type, which also may be parts of presentation files are Truereplicas. Truereplicas are the only replica type from the three types, to which the end users have access for direct manipulation (updating). On the other side, these are also the only replica type which requires the support of really high availability and QoS provision.

All replicas which are created and maintained by our replication system are an identical copy of original media. Replicas with errors (non-identical copy) are not allowed to be created. Furthermore, we do not support any replication service for function calls, and elementary data types.

10.1.4 Design and Implementation Architecture

10.1.4.1 The Replication Mechanism

Basically, our replication system does not assume a client-server replication model, because there are no fixed clients and servers in the medianode architecture; every medianode may be client or server depending on its current operations. Peer-to-peer model with the following features is used for our replication system:

- Every replica manager keeps track of a local file table including replica information.
- Information whether and how many replicas are created is contained in the every file table. I.e. each local replica manager keeps track of which remote replica managers (medianode) are caching which replicas.
- Any access to the local replica for reading is allowed, and guaranteed that the local cached replica is valid until notified otherwise.
- If any update happens, the corresponding replica manager sends a multicast-based update signal to the replica managers which have the replica of the updated replica and therefore members of the multicast group.
- To prevent excessive usage of multicast addresses, the multicast IP addresses through which the replica managers communicate can be organized in small replica sub-groups. Examples for such sub-groups are file directories or a set of presentations about a same lecture topic.

10.1.4.2 Update Distribution and Consistency Control Mechanism

There are some known approaches for maintaining consistency in previous works:

- *Callback* mechanism for client-server model. In this model, a client which makes a local update propagates its updates to the server which holds the global replication table for all replicas. After a global commitment from server with other clients which hold the replica, the server sends a commitment message to the client, and the update is treated as global [Cam98, SKK+90];
- *Leases* for client-server model. This model is classically used for update distribution between web servers and cache proxies for web-based applications [YAL99];
- *Poll each Read & Poll* for client-server model. In this model, clients check always whether the cached files are updated. For this purpose, they first send the corresponding message to the server, before they access to the cached files for reading or writing [YAL99].
- *Current table* for peer-to-peer model. In this model, each peer maintains a current replication table which keeps the track of update histories for files and their replicas [GWP99];

The update distribution mechanisms in medianode differs between the three replica types and their managers. This is due to the fact that the three replica types have different levels of requirements on and characteristics of high availability, update frequency and consistency. Experience from GLOVE [PKvST00] and [JAC98] also shows that differentiating update distribution strategies makes sense for web and other distributed documents.

10.1.4.3 Updates Transport Mechanism

The medianode's replication system offers an unique interface to the individual update signalling and transport protocols which are selectively and dynamically loaded and unloaded from the replica transport manager that is implemented as an instance of medianode's access bow. The possible update transport and signalling protocols are:

- RPC protocol [CDK01, BG95] as a simple update distribution protocol. This mechanism is mainly used at the first step of our simple and fast implementation.
- A multicast based RPC communication mechanism. In this case, the updates are propagated via multicast other replica managers which are members of the multicast group. RPC2 [SKK+90, SS90] is a good candidate for the first implementation. RPC2 also offers the transmission of large files, such as the updated AV content files or diff-files, by using the Side Effect Descriptor. But, the RPC2 with Side Effect Descriptor does not guarantee any reliable transport of updates.
- LC-RTP based reliable multicast protocol: LC-RTP (Loss Collection-Real-time Transport Protocol) [ZGJS00, Gri00] is originally developed as an extension of RTP protocol to support the reliable video streaming within the medianode project. We adopt LC-RTP and check the usability of the protocol, depending on the degree of reliability required for the individual groups of replicas.
- FTP [Tan03] for the asynchronous transport of the updates, i.e. modified files and (meta-) data;
- File system mounting as of volume and directory mounting when the medianodes run within a same domain, and they have the same file system such as NFS [SGK+85] as server and clients running on them;
- SMTP mechanism [CDK01]. We use this mechanism for the case when there is no network connection between replica managers at the time of update distribution.

10.1.4.4 Approaches for Resolving Update Conflicts

The possible conflicts that could appear during the shared use of presentation data and files are either (a) update conflict when two or more replicas of an existing file are concurrently updated, (b) naming conflict when two (or more) different files are given concurrently the same name, and (c) update/delete conflict that occur when one replica of a file is updated while another is deleted. In most existing replication systems, the conflict resolving problem for update conflicts was treated as a minor problem. It was argued that most files do not get any conflicting updates, with the reason that only one person tends to update them [GWP99]. Depending on the used replication model and policy, there are different approaches to resolving update conflicts, of which our replication system uses swapping, dominating, merging strategies [SKK+90, GRR+98, RRP99, JGPH91]:

- Swapping - to exchange the local peer's update with other peer's updates;
- Dominating - to ignore the updates of other peers and to keep the local update as a final update;
- Merging - to integrate two or more updates and build one new update table;

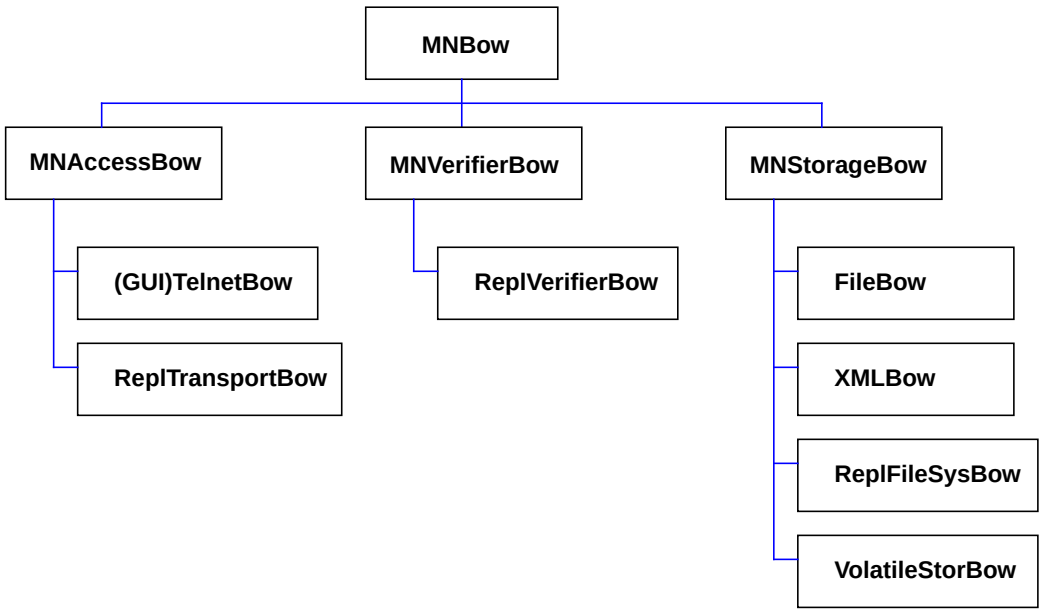


Figure 33: A class hierarchy of the extended medianode bows for the replication system.

10.1.5 Implementation

We have implemented a prototype of the proposed replication system model for Linux platform (Suse 7.0, Redhat 6.2). Implemented are the media (file) and its replica manager (ReplVerifierBow), update transport manager (ReplTransportBow), replica service APIs. The APIs are Unix-like file operation functions such as open, create, read, write, close (ReplFileSysBow), and a volatile storage bow which maintains user's session and token information.

10.1.5.1 Bow Description

Figure 33 shows a class hierarchy of medianode's basic bows and of extended bows for the replication system. MNBow is the root class and the three bow APIs, MNAccessBow, MNVerifierBow and MNStorageBow are implemented as MNBow's child class.

Table 12 gives a short descriptions of bows which implement medianode's basic functions and the services of our replication system.

Bow	Description
MNBow	• addressible via an unique bow identifier and version number • uses request and response queues and dispatcher threads • defines request processing routine
MNAccessBow	• child class of MNBow and implements access bow API • offers RPC server modules and enables RPC connection from web server • HTML-based presentation files are provided via this bow
(GUI)TelnetBow	• child class of MNBow and a variant of access bow • offers TCP server modules • acts as telnet server and provides information which medianode maintains such a list of bows loaded by the core, memory usages of a certain medianode's process running etc. • GUIBow implements a TelnetBow with a graphical user interface
ReplTransportBow	• child class of MNBow and a variant of access bow • implements the transport managers for replication service • offers transport protocol modules such as RPC, RPC2, and LC-RTP (LC-FTP)
MNVerifierBow	• child class of MNBow and implements verifier bow API • offers modules needed for user authentication
ReplVerifierBow	• child class of MNBow and a variant of verifier bow • implements the media (file) and replica managers for replication service • maintains the three replica tables
MNStorageBow	• child class of MNBow and implements storage bow API
FileBow	• child class of MNStorageBow • implements functions for local file operation • no interface routine support for replication service
XMLBow	• child class of MNStorageBow • implements functions for local file operation • offers modules for dynamic generation of presentation files • no interface routine support for replication service
ReplFileSysBow	• child class of MNStorageBow • implements functions for local file operation • implements interface routines for replication service
VolatileBow	• child class of MNStorageBow • implements functions for maintaining volatile data such as memory usage information • implements interface routines for replication service

Table 12: A short summary of medianode's bows which used for the replication system

10.1.5.2 Interaction Mechanism

The interaction model for medianode's bows which also implement our replication service is based on a 'request-response' communication mechanism. A bow which needs to access data or services creates a request packet and sends it to the core. According to the request type, the core either processes the request packet directly, or forwards it to a corresponding bow. The processing results are packed in a response packet and sent to the origin bow. The request and response packets contain all necessary information for the communication between bows as well as processing the requests.

10.1.5.3 Presentation Service without Replication Support

Based on this request-response mechanism, we experimented some presentation scenarios with and without replication service. Figure 34 shows one example of presentation services without replication system.

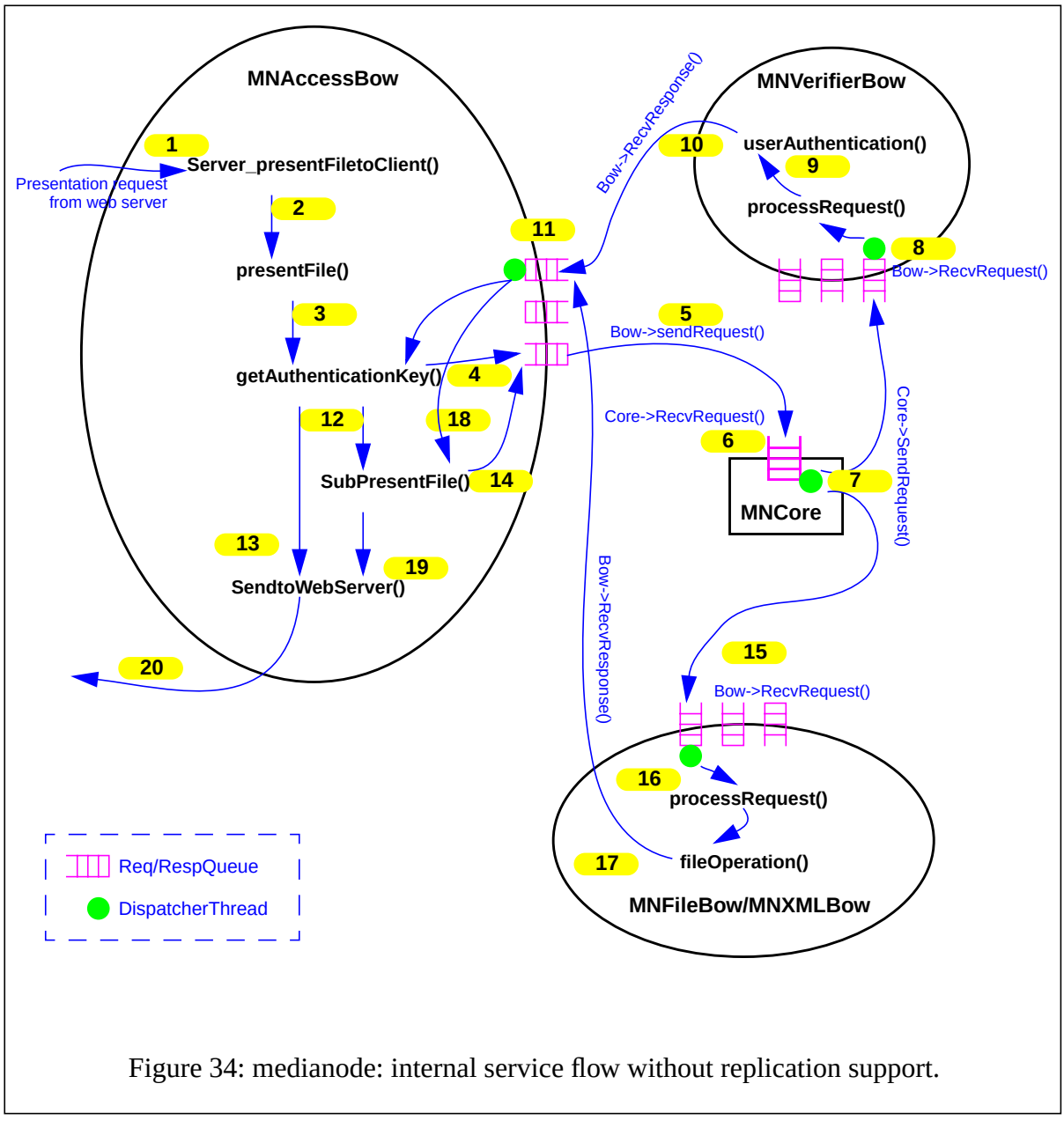


Figure 34: medianode: internal service flow without replication support.

Upon receiving a presentation request from user via web server, the MNAccessBow creates first a request packet to check user's authentication (steps 1~4) and sends it via the core (steps 5~7) to MNVerifierBow which puts authentication test value into a response packet and sends it to the origin bow, MNAccessBow (steps 8~11). In the case of a successful authentication, MNAccessBow creates a request packet to get the required presentation data and sends it via the core to a corresponding storage bow (steps 12~15). Either MNFileBow or MNXMLBow, it depends on the requested (media) data type, checks whether the data exists locally. The bow then creates a response packet which contains either a file handle or an error message, and sends it to the MNAccessBow (steps 16~20). MNAccessBow sends then to the web server a response which is either an authentication failure message or a presentation file.

10.1.5.4 Presentation Service with Replication Support

Initialization of MediaList and Replica Tables

We now describe the medianode's operation flow with the replication service. Basically, the replication service in medianode begins by creating media list and replica tables of the three replica types in each medianode. As shown in Figure 35, ReplFileSysBow sends a request packet via the core to ReplVerifierBow for creating a media list for media data which locate in the local medianode's file system (steps 1~2). Upon receiving the request packet, ReplVerifierBow creates a media list which will be used to check the local availability of any required media data (step 3). ReplVerifierBow then builds the local replica tables for the two replica types, Truereplicas and Metareplicas, if the replica information exists already. A medianode configuration file can specify the default location where replica information is stored. Every type of replica table contains a list of replicas with the information about organization, replica volume identifier, unique file name, file state, version number, number of replicas, a list of replica, a multicast IP address, and some additional file attributes, such as access right, creation/modification time, size, owner, and file type. The third replica table for the Softreplicas to which the local system resource, user session and token information belong may be needed to be created in terms of memory allocation. The contents of this table can be partly filled when users request some certain services. Once the replica tables are created, they are stored in the local file system and accessible persistently.

Maintaining Replica Tables

In medianode, these three replica tables are maintained locally by the local replication manager. Thus, there is no need to exchange any update-related messages for the files of which there is no replica created. This approach increases the system resource utilization, especially network resources, by decreasing the message numbers exchanged between the replication managers among the distributed medianodes. However, when any medianode wants to get a replica from the local replica tables, the desired replica elements are copied to the target medianode. Additionally, the replication manager at the target medianode keeps these replica elements separate in another replica table. This table is used only for the management of remote replicas, i.e. for the management of replicas for which their original files are stored in a remote medianode.

Acquiring a Replica to Remote Replication Managers

Upon receiving the service requests (data access request) from users, as shown in Figure 35, the local medianode attempts to access the required data in a local storage bow (ReplFileSysBow) (step 4~5). In the case, when the data is not available locally, the local ReplFileSysBow sends a request packet to ReplVerifierBow to get a replica for the data. The ReplVerifierBow then start a process to acquire a replica by creating a corresponding request packet which is passed to ReplTransportBow (steps 6~8). The ReplTransportBow multicasts a data search request to all the peer replication managers and waits for replication managers to respond (step 9).

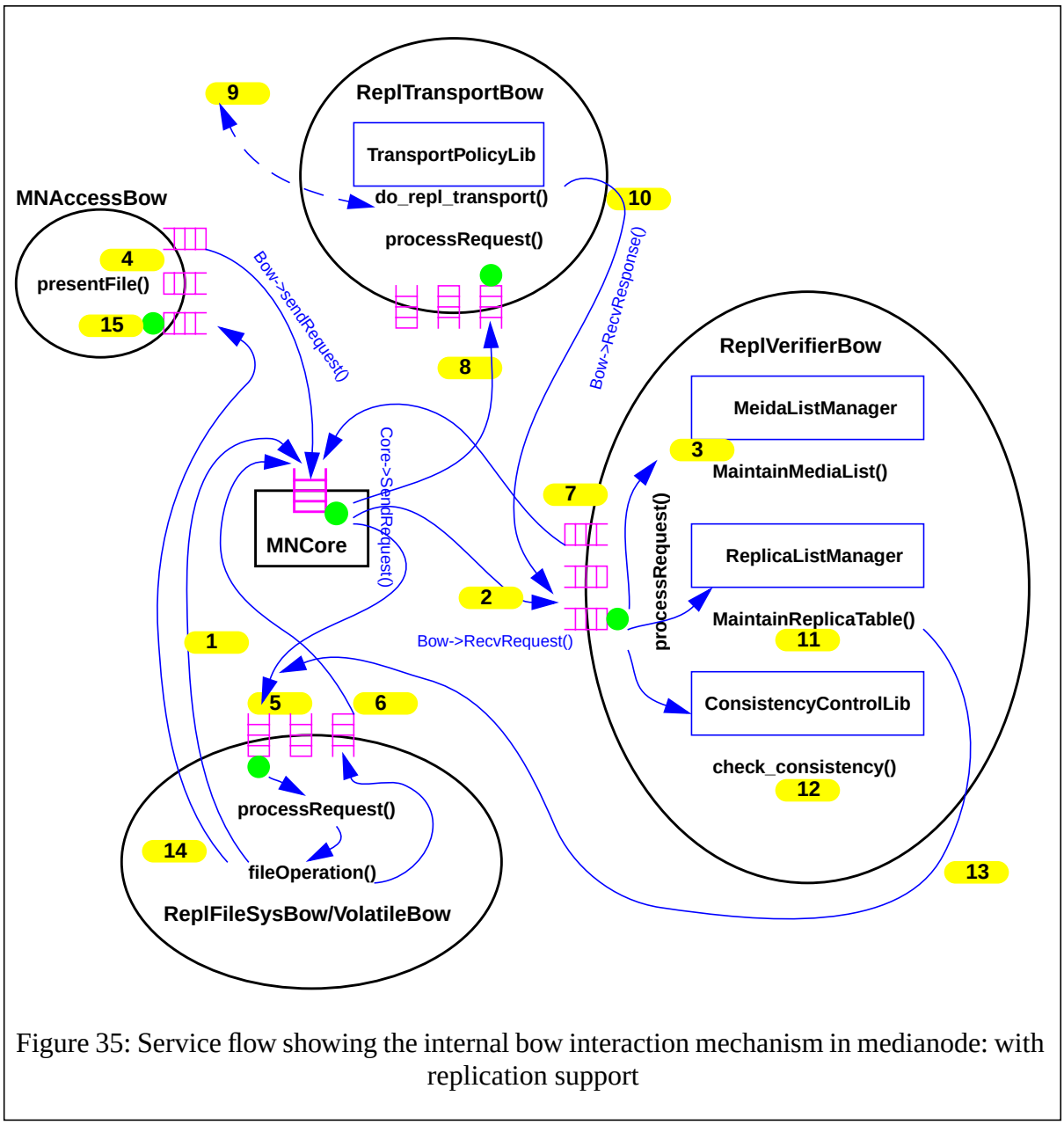


Figure 35: Service flow showing the internal bow interaction mechanism in medianode: with replication support

The list of medianodes to which the multicast message is sent can be read from the medianode's configuration file. Whether the ReplTransportBow waits for all responses or receives the first one is dependent on the optimization policy which is given as configuration flag. After

receiving the target replica, the ReplTransportBow sends a response packet to the ReplVerifierBow. The ReplVerifierBow then updates the corresponding replica tables, i.e. ReplVerifierBow adds the new replica element to the Truereplicas table and its metadata to the Metareplicas table, respectively (steps 10~13). Finally, the local ReplFileSysBow which originally issued replica creation request creates a response packet including the replica handle and then sends it to the MNAccessBow (steps 14~15).

10.1.6 Related Work

Several approaches to replication have already been proposed. The approaches differ for distributed file systems than those for Internet-based distributed web servers and those for transaction-based distributed DBMS. Well-known replication systems in distributed file systems are AFS [Cam98], Coda [SKK+90], Rumor [GRR+98], Ficus [JGPH91] and Roam [RRP99] which keep the file service semantics of Unix. Therefore, they make easy to develop applications based on them. They are based on either client-server model or peer-to-peer model and use often optimistic replication which can hide the effects of network latency. Their replication unit are mostly file system volume which lead to a large size and relatively a low number of replicas.

There are some optimization works for these examples in terms of update protocol and replica unit. To keep the delay small and therefore maintain the sense of real-time interaction, it was desirable to use the unreliable transport protocol such as UDP. In the earlier phases, many approaches have used the unicast-based data exchanges by which the replication managers communicated with each other via ‘one-to-one’. This has caused large delays and made the real-time interaction impossible. To overcome this problem, the multicast-based communication is used in some recent cases [GWP99, SS90, ZGJS00, MH00]. In the case Coda, the RPC2 protocol is used for multicast-based update exchange, which offers with Side Effect Descriptor the transmission of large files by using the Side Effect Descriptor.

For limiting the amount of storage used by a particular replica, Rumor and Roam developed the selective replication scheme [Rat95]. A particular user who only needs a few of the files in the volume, the user can control which files to store in his local replica with selective replication. A limitation or disadvantage of selective replication is the ‘full backstoring’ mechanism: if a particular replica stores a particular file in a volume, all directories in the path of that file in the replicated volume must also be stored.

JetFile [GWP99] is a prototype distributed file system which uses multicast communication and optimistic strategies for synchronization and distribution. The main merit of JetFile is its multicast-based callback mechanism by which the components of JetFile, such as file manager and versioning manager interact to exchange update information. However, the multicast callbacks in JetFile do not guarantee that they actually reach all of other replication peers, and the centralized versioning server which is responsible for serialization of all updates can lead to a overloaded system state. Furthermore, none of the existing replication systems does not support of the quality of service (QoS) characteristics of (file) data which they handle and replicate.

Tabellarischer Lebenslauf

NAME: Giwon On

Persönliche Daten:

Geburtsdatum:	22. Januar 1965
Geburtsort:	Kim-Je, Jeonbuk, Korea
Familienstand:	verheiratet (8. April 1995) mit Youjin Kim, 2 Kinder
Staatsangeh.:	koreanisch

Anschrift (privat):

Jahnstraße 119
64285 Darmstadt

Tel.: +49 6151 313884

Anschrift (beruflich):

Multimedia Kommunikationen (KOM)
Technische Universität Darmstadt
Merckstraße 25
64285 Darmstadt

Tel: +49 6151 166162
Fax: +49 6151 166152
EMail: giwon.on@kom.tu-darmstadt.de

Beruflicher Werdegang:

Datum	Berufsbezeichnung	Abteilung	Institution
1999-2004	wissenschaftl. Mitarbeiter	Fachbereich Elektrotechnik und Informationstechnik	TU Darmstadt
1993-99	Senior Research Staff	Computer & Software Technology Laboratory	ETRI Korea

Studium:

1986-92	Diplom-Informatiker, FB Informatik, Uinversität Dortmund
1985-86	PNDS, Deutschkurs bei OESV, Bochum
1983-85	Studium der Physik, Jeonbuk National University, Korea

Sprachen:

Koreanisch (Muttersprache), Deutsch (fließend), Englisch (fließend)