<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Data availability analysis in P2P networks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Sanaullah, Nazir</td>
</tr>
<tr>
<td><strong>Publication Date</strong></td>
<td>2012-09-28</td>
</tr>
<tr>
<td><strong>Item record</strong></td>
<td><a href="http://hdl.handle.net/10379/3606">http://hdl.handle.net/10379/3606</a></td>
</tr>
</tbody>
</table>

Some rights reserved. For more information, please see the item record link above.
Data availability analysis in P2P networks

Sanaullah Nazir

Submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy

SUPERVISORS:
Prof. Dr. Manfred Hauswirth

EVALUATION COMMITTEE:
Prof Dr. Siegfried Handschuh
Prof Dr. Anwitaman Datta

Digital Enterprise Research Institute, National University of Ireland, Galway
September 2012
Abstract

P2P network architectures have gained popularity as applications for sharing files between users. A P2P network provides a scalable, robust, and economical storage architecture. These features have led to the extended use of P2P network applications, ranging from file sharing to data sharing for video and telecommunication domains.

The shift in storage system being used from high cost, reliable servers to user-centered storage devices led to reliability and availability problems for the P2P network. Peers are machines of users that can go offline at any time. The data stored on the machines are not available during the offline time. Data replication is a common approach for handling data unavailability, which is where multiple copies of files are placed on different peers in the network. In data replication, peers transfer complete/partial data to other nodes. Therefore, data replication provides higher data availability in case of churn.

I present data replication algorithms in this thesis to improve the availability of data in the network. With an increase in availability and overhead, the basic challenges faced during the development of data replication algorithm are: (i) How many replicas for a data object should be created? (ii) On which peer(s) should the replicated data objects be stored? (iii) Which files should be replicated?

Initial work in data replication considered the static replication of data based on the overall availability of nodes in the network. These approaches overestimated the number of replicas, which lead to high maintenance costs. Dynamic approaches for estimating replica numbers were developed to handle this issue. From the analysis of the current approaches, I found that the proposed mechanisms for dynamic approaches to replication did not provide a balanced replication of data. Data were only replicated to highly available nodes, which were overloaded with data. The second issue was the inability to adapt to the changing behaviour of peers. In this thesis, I present an approach that selects a node set comprised of both highly available and lowly available nodes, in order to provide load balancing in the network. I provide a feedback-based approach where previous behaviours are incorporated.
in the next behavioural analysis. Compared to the existing approaches to replica calculation, this approach is able to determine the appropriate number of replicas and placement locations with the changing dynamics of the system.

The replication system relies on node behaviour prediction algorithms using Monte Carlo simulation and Time series analysis. Each node performs an analysis on the historical traces of its online and offline times in the network. Each node shares the availability log with the replication initiator node, and the prediction of future behaviour is made based on the logs received. The data-owning peer uses this information to run the replica placement algorithm to select nodes that are present for a particular duration, supporting the presence of each others in the network. Partial data replication is supported by the system by applying Zipf distribution to calculate the most popular files.

I performed the evaluation using my replication approach and dynamic replica placement algorithms, based on the following parameters: replica count, reliability of data, average availability of nodes in the replica set, and failure analysis for querying data. The replica count analysis shows that the number of replicas required were almost half compared to the previous dynamic approaches. The reliability analysis shows that overall reliability of the data was better in this approach compared to the other dynamic replica placement algorithms. My replication algorithm produced replica sets with a lower average availability compared to the replica set of the other approaches, but the reliability analysis suggests that my approach distributes data more evenly between nodes, resulting in better overall data availability. The availability of data in the network was higher than other approaches. The failure analysis for request failures for data shows that my replication algorithm has a better node selection mechanism compared to other approaches, with better data availability.
Acknowledgements

I would like to thank Allah (SWT), for the courage and strength he gave to finally complete my dissertation. I am thankful to him to provide me the opportunity to work under the supervision of Prof. Dr. Manfred Hauswirth.

I would like to express my deep gratitude to Prof. Dr. Manfred Hauswirth for the support, guidance and patience throughout my study. His guidance to identify and polish the ideas and writing, has helped me to finally complete this dissertation.

My special appreciation is given to Dr. Tomas Vitvar and Dr. Stefan Decker, for giving me the chance to joining DERI, and entering into the exciting world of research. With their continuous guidance during the initial research and encouragement to pursue the Phd, has helped me to go through the tough times.

I would like to expressed gratitude to Dr. Christian von der Weth, for his support in reviewing the thesis and provide invaluable feedback in improving the quality of the dissertation.

Special thanks to Dr. Brahmananda Sapkota and Dr. Kashif Iqbal, for the guidance during my initial research study. It has been a pleasure working with them. They have been very helpful with the discussions during the research problem identification and to refine my research work.

During my stay in DERI I have had the pleasure to work and interact with very highly talented people, through the discussion with them has helped me to improve my research. I would like to thank my friends there which were Ratnesh Sahay, Aftab Iqbal, Waseem Akhter, Simon Scerri, VinhTuan Thai, Marco Zuniga, ZhangBing Zhou, Maciej Zaremba, Xia Wang, Adrian Mocan, Matthew Moran, Emilia Cimpian, Doug Foxvog, Conor Hayes, Aidan Hogan, Andreas Harth, Peyman Nasirifard, Wassim Derguech, Sami Bhiri, Walid Gaaloul, Vassilios Peristeras, Smitashree Choudhury, Jrgen Umbrich and Uldis Bojars.

I am very thankful for all the adminstration and technical support team in DERI, including Michael Turley, Brendan Smith, Hilda Fitzpatrick, Claire Browne, Carmel Fennell, Michelle
Treacy, Maria Smyth, Brian Wall, Sylvia McDonagh, Andrew Gallagher and Gerard Conneely for all the administrative and technical problems I have been putting them through.

It has been a very interesting and enjoyable life in Galway, Ireland. The weather there, gives you no room other than to do research.

I am very thankful for the AOL Dublin, management, Brendan Smith, Aengus Maclean, Gary Clark, Jennifer O’Dowd, Dairmuid Gill, Paul Downey and Anthony Ryan for giving me the chance to join and work with very talented teams at AOL. Very thankful to all the support I got from my friends in AOL Dublin including, Ronan O’Rafferty, Philip McGovern, Aidan Foley, Divid Kelly, Niall O’ Beaglaoich, Gareth Daly, Cian Hamill, Martin Boyle, Przemek, Fernando Baigorri, Gonzalo Garcia, Marcin Kluczynski, Eamon McCormack and Kieran O’Hara.

I am very thankful for the friendly and relaxing environment provided during my research by Gulraiz, Mohsin, Umair, Adnan, Talal, Imran Khan, Zubair, Irfan, Amir, Ahmad, Zaid, Baqar Raza, Alok and Shehroz Khan.

This dissertation is dedicated to my parents. I express my deepest gratitude to my parents, brothers, my wife and the whole family. There are no words that can express my gratitude to my parents for all the support they gave me and prayers they have been doing during the tough times in my Phd and helping me during my disc rupture surgery during my Phd.

I gratefully acknowledge the financial support provided to me by EU project Adaptive Service Grid (ASG), SemanticGov and Science Foundation Ireland (SFI).
Contents

1 Introduction 3
  1.1 Solution outline .................................................. 5
  1.1.1 Problem statement ............................................. 6
  1.1.2 Research statement ........................................... 7
  1.2 Contributions ..................................................... 8
  1.3 Thesis structure .................................................. 10

I Background 13

2 P2P overlay networks 15
  2.1 Unstructured networks ............................................ 16
  2.2 Structured networks .............................................. 17
    2.2.1 Chord ....................................................... 17
    2.2.2 DKS ......................................................... 18
    2.2.3 Kademlia .................................................... 22
    2.2.4 P-Grid ....................................................... 24
  2.3 Super-peer network .............................................. 25

3 Data replication 27
  3.1 Basic concepts .................................................... 27
  3.2 Replication parameters .......................................... 28
  3.3 Synchronous and asynchronous replications ....................... 29
  3.4 Single-master and multiple-master replications .................... 30
  3.5 Partial replication, full replication, and erasure code .......... 30
# Contents

4 Fault management | 33
---|---
4.1 Reliability analysis | 34
4.1.1 Reliability configurations and reliability probability | 34
4.1.2 Failure analysis | 36
4.2 Availability analysis | 37
4.3 Monitoring availability | 37

5 Prediction models | 39
---|---
5.1 Time series analysis | 39
5.1.1 ARIMA | 40
5.2 Monte Carlo simulation | 41

II Related work | 45
---|---

6 Replication in P2P networks | 47
---|---
6.1 Replication in unstructured P2P networks | 48
6.1.1 The number of replicas required | 48
6.1.2 Where to place the replicas | 49
6.2 Replication in structured P2P networks | 50
6.2.1 CHORD | 50
6.2.2 DKS | 51
6.2.3 Kademlia | 53
6.2.4 P-Grid | 54
6.3 Other replication strategies in a P2P network | 54
6.4 Summary | 56

7 Behavioural Replication approaches | 57
---|---
7.1 Replication using node availability distribution | 57
7.2 Markov model-based approaches | 60
7.3 Summary | 62

8 Application of prediction models in computing domains | 65
---|---
8.1 Time series analysis | 65
8.2 Monte Carlo simulation | 66
8.3 Summary | 67
III  Design and analysis of availability based replication 69

9  Data gathering and data analysis 73
   9.1  Data gathering ........................................ 73
   9.2  Data analysis .......................................... 74
      9.2.1  Time series modelling ............................. 74
      9.2.2  Monte Carlo modelling ............................ 88

10  Behaviour prediction 91
    10.1  Time series modelling of N-step prediction ............... 91
    10.1.1  Forecasting future online time with an ARIMA model .... 92
    10.2  Monte Carlo modelling future predications ............... 93
    10.2.1  Forecasting future online time with a Monte Carlo simulation .... 94

11  Modelling reliability 95

12  Algorithm 99
    12.1  Replication cost model .................................. 100
    12.2  Replication algorithm .................................... 101

IV  Evaluation 109

13  Time series analysis 113
    13.1  Evaluation setup ....................................... 113
    13.2  Required replicas ...................................... 113
    13.3  Reliability analysis ..................................... 114
    13.4  Replica set availability .................................. 115
    13.5  Lookup failure ........................................... 116
    13.6  Message overhead ........................................ 118

14  Monte Carlo simulation analysis 119
    14.1  Evaluation setup ....................................... 119
    14.2  Required replicas ...................................... 120
    14.3  Lookup failure ........................................... 120
Chapter 1

Introduction

A Peer-to-Peer (P2P) network is a distributed application layer architecture. Tasks are distributed among nodes/peers. The peers share their resources (storage space, processing power and network bandwidth) with the network. This approach eliminates the single point of failure, as in client server architectures, providing robustness against node failures, scalability, and load balancing. A P2P network is widely used in document sharing, exchanging documents, instant messaging, and cooperative work. The popularity of this type of file sharing system has led to an increased number of users upwards of millions of individuals. A secure, highly available data sharing/information retrieval system is required to deal with this heavy load of data sharing. P2P networks are mainly dependent on a lightweight user machine, unlike its predecessor, the client-server environment. The decentralized approach exposes the unreliability of lightweight user machines as a trade-off for having a robust, scalable network architecture and the presence of unreliable nodes. The P2P system has to maintain a high level of availability together with improved search algorithms. I address the issue of data replication to improve data availability in P2P networks in this thesis. In data replication, data are replicated to other nodes in the network to handle fault-tolerance in the P2P network. Nodes failures affect the availability of data. With data replicated between nodes, the availability of data in the network does not decrease. Only one node containing the data is required to consider all data available in the network.

There are two basic approaches for replication in P2P systems: replicating data items themselves and location information (index)-based replication. Replicating locations do not improve the reliability and availability, as these do not increase the physical copies of the data. Data replication replicates fully/partially data to other locations.
The maintenance overhead is higher in data replication approaches, but the overall availability is improved.

The P2P network’s dynamic environment poses a challenge for maintaining data and information retrieval. There are many replication techniques available, but the lack of consideration of peer availability information in the overhead of maintenance is very high. It is quite evident that nodes that are readily available would allow for sustaining data in the network for a longer period of time. Due to the autonomous behaviour of nodes, it is not possible to govern the nodes of a P2P network. However, it is possible to utilize availability and reliability information of the nodes to improve data durability. Nodes with varying failure/churn rates in the network require an approach that updates the replica locations and the number of replicas based on the churn/failure rate. Current dynamic approaches assume a constant churn rate. In a system with a varying churn rate, a replica maintenance algorithm using a constant churn rate creates an incorrect number of replicas, resulting in unnecessary replicas and burdens the system with the cost of maintenance overhead for these replicas. Generating less numbers of replicas without considering the churn rate, would lead to lower data availability and lead to data loss.

Traditional replication approaches [86], [29], and [43], have been used to provide fault tolerance, but these approaches have not been able to model the behaviour of the machines involved in the network. Efforts in data replications considering availability [60], [74], [8] and [20], do not completely cover the dynamics of the network. As will be seen in the analysis section, the selection of nodes with high availability will not always result in an increase in data durability. These proactive management approaches pose a threat to reliable prediction, which can lead to high management cost and loss, due to the maintenance of long traces of probes. The ability to have a feedback system in the model for replication makes the overall system static to changes. I have modelled the system to update the replication approach based on the node’s behavioural changes. The selection of nodes for replication is equally important as selecting the number of replicas. The replica placement improves data availability in the system. Current dynamic approaches concentrate more on the number replicas to be created, and not on the placement strategies. My node selection approach selects nodes from the network that complement each node present in the network.
1.1 Solution outline

I have developed an approach for data replication using the node availability information. Peers in the network predict future behaviour by using previous peer network connectivity history logs. The data owner requests the information about the predicted availability in the future from other peers, and calculates the overall availability of all of the neighbours for a particular duration. The data owner peer selects the set of peers that covers the complete time duration or maximizes the overall availability for that particular duration. Only partial replication is applied by selecting highly requested files for replication. The data owner replicates these files to a selected set of peers.

The peers use Monte Carlo simulations and Time series analyses as behavioural analysis approaches for predicting the behavioural changes on the nodes in the network. The underlying replication approach used for both of these algorithms is to maintain the history of data online and offline time by the each node. The data owner node uses the calculated future online time predictions based on the two behavioural analysis approaches from the neighbours. The data owning peer applies the replica placement algorithm after collecting this neighbour availability information in order to determine replicas for a particular duration in the future, which determines the nodes that will be available. Data are distributed to nodes that cover the time gaps for that time period. I used a Zipf distribution to calculate the number of the most popular files required. Based on the analysis of the demand of data, applying a Zipf distribution allows for the determination of the most popular files at a node. The algorithm calculates the time required to transfer those files using the network bandwidth available to the nodes.

I compared the behavioural and replica placement algorithms to existing approaches. A heterogeneous network was set up with varying bandwidth and storage availability. The evaluation covered the following points: determining the number of replicas, availability of the replica set, reliability of the replica set, and failure analysis of nodes in the network. My algorithms generated almost half the number of replicas compared to the existing approaches, with the same level of availability information sharing overhead. This algorithm reduces the maintenance overhead by generating fewer replicas. A reliability analysis of the replica set suggested that even with the lower number of replicas, the reliability of the replica set was higher than in the existing approaches. Overall, the data were available for a longer duration without requiring maintenance. The average availability for the replica set was lower in my algorithm, suggesting that it is not highly dependent on the nodes with high availability. Finally, a failure analysis showed that the query failure rate for my algorithm was much lower than the failure rates of existing approaches.
1.1.1 Problem statement

A file sharing application, called Napster [3], for downloading music was developed over a P2P network. The platform gained interest and usability in a wide range of communities. The main reason for this success was the flexibility offered by the network to its users. A P2P network provides users with a plug-in platform to share and gain access to resources. A peer connects to the network using a given set of initial peers in the network. The P2P network handles the bootstrapping process where the new peer is integrated into the network. Due to P2P network scalability, ease of use, cost efficiency, and robustness against failure of peers it is used in a wide variety of domains (e.g., telecom, video sharing etc). Researchers are dedicated to contribute to the evolution of P2P networks. In addition, this network has allowed users to move away from the strict client-server model of data sharing. Several lines of research are working towards the optimization of P2P network [22][5], including:

- **Servers**: A user of the network is itself a client and a server, responsible for hosting and requesting content.

- **Total distribution**: There is no central responsibility or control. There are no dedicated servers for dedicated tasks in the network.

- **Routing with local knowledge**: The lack of global knowledge of a user leads to only routing of data based on the local knowledge of the users.

These P2P network characteristics are not strict rules, as can been seen in one of the first networks, Napster, which included a central server to manage available files in the network. This was used to maintain the availability of the files and their location. With all of these characteristics, a new range of challenges emerged. This file sharing approach led to the development of other networks such as Kazaa [1], and Limewire [2], which used the same design principles. A P2P network is about utilizing the resources around the world that are being offered by the users. It relieves the system from the traditionally expensive and dedicated infrastructures, controlled by the service provider, where the user was not responsible for the privacy of the data or its availability.

This peer-to-peer design has created the following important challenges:

**Resource responsibility**: A P2P network provides the freedom and flexibility to a peer of the network to connect to the network at any time without being restricted to one server. The bootstrapping process of the peer connecting to the network is handled by the P2P network by offering the new peer a set of possible peers to connect to at startup. The underlying commu-
nication protocol handles the network neighbours of the peer. The peers have no contractual binding with the network, and can leave the network at anytime. This flexibility comes at a cost to the user responsible for maintaining of the network. The user, as part of the network, is responsible for providing the hardware and software support. The user has to assure network and computing resource availability to the P2P network to keep the network performing in an optimal fashion. These responsibilities of the user become a design parameter for optimizing algorithm design, which can be analyzed based on network users.

*Data Availability:* The unpredictable behaviour of users influences the network. The ability of the network to retrieve the data depends on the presence of the users. Such a change in the topology of the network requires a solution to handle the issue of data availability in the network, when a central node is not responsible for it. A central point of failure is the major problem of a client/server network. Data availability in a P2P network is maintained by copies of a data in the network.

### 1.1.2 Research statement

This thesis offers algorithms and models that can help predicting user behaviour in a P2P network using time series analyses and Monte Carlo simulations. I address the issue of choosing the optimum number of replicas based on the peer availability and the availability of neighbour peers in the network. Based on the sharing of peer availability information in the P2P network, high data availability can be maintained by sharing availability information between the nodes/users.

With a highly distributed environment without any central control over the nodes, the main problem is predicting the behaviour of users/nodes based on local knowledge. Existing approaches use static maintenance algorithms to predict this behaviour. Due to the freedom offered by a P2P network, the replication of data does not only depend on the node availability, but also on the resources provided by the users of the network. Maintaining the fair sharing of data load, reducing the overhead of extra replicas, and enhancing the availability of data are addressed by this algorithm.

*My aim is to provide high availability of data with a reduced replication overhead compared to existing approaches*

This hypothesis is important because an accurate prediction about the behaviour of a user can optimize the selection and placements of replicas. It is hard to obtain this information
in a distributed environment, considering the integrity of nodes/users network connectivity information. Some false claims to the nodes availability information can be made as well by the nodes in the network, when sharing this information with other nodes.

Although current research addresses the issue of data availability, few studies have considered approaches used in other fields that are used for similar purposes. Time series analysis is used by the financial industry to predict future trends for determining the investment possibilities in various sectors. Stock markets utilize time series analysis-based prediction systems to predict future stock prices trends. The level of accuracy that these approaches have shown in other fields can be demonstrated in P2P networks. It is crucial to reduce the overhead of data sharing in P2P networks. However the complexity of the approach as well as number of dependencies on the size of the traces to be used for analysis is quite high as well.

In the rest of the thesis, I will show how approaches used for analyses of financial systems can be used in P2P networks and will address the main issue of data availability while considering the heterogeneous nature of the network.

1.2 Contributions

The contribution of the thesis based on the hypothesis is to improve data availability in a highly dynamic network. Given the current work in availability analysis and P2P networks, the following are the main contributions of the thesis:

*Analysis of current replication approaches for P2P networks*: I provide an overview of relevant research in the field of replication in current P2P networks. I determine how the basic overlay network handles replication to provide fault-tolerance. I describe the current state of the art for replication as a survey in chapter 6, and classify it based on the type of network and the type of replication they provide. Using this survey, I gather the requirements of the replication provided in this thesis.

*Analysis of current availability-based P2P networks*: As part of the analysis of replication in a P2P network, I determine algorithms in chapter 7 for replication of data that are not provided by the basic P2P networks, but are based on availability requirements. These approaches are based on probabilistic analyses of the network. These approaches provide a dynamic way to handle replication and they are not bound to any specific type of P2P network.
Various techniques based on the Markov model, lifetime awareness, and overall availability of the nodes are used.

**Analysis of reliability theory and its application to P2P networks:** From the analysis of the replication approaches for P2P networks, I found that the parameters to analyze a P2P network are very similar to the research work conducted in the reliability analysis domain. These deal with similar issues to determine the reliability of components in a system, compared to my reliability of nodes in a network. Chapter 4 covers the basic background details required to model the system based on the reliability theory. In section 11, I use the background details provided in chapter 4, to model the P2P network.

**Analysis of behaviour prediction algorithms:** As mentioned earlier, current availability-based approaches are based on computing domain applications, I have assessed other domains that are highly dependent on prediction. I have applied the algorithms used in those domains to P2P network given in chapter 5. These approaches have been used in the computing domain as well, which has also inspired me to use them for replication in P2P networks. In chapter 9, I have modelled the P2P network using prediction model approaches and described each step required to process the prediction for forecasting future behaviour. I provide examples of how these models are used in a P2P network. I also show the process of calculating parameters and forecasting future values based on a real trace from a P2P network.

**Design and analysis of behavioural algorithms and reliability theory with application towards a P2P network:** I utilize the system reliability model to determine the availability of the peers in the network. Each node predicts its online and offline time in the network using previous connectivity history information. Time series analyses and Monte Carlo simulations are used to predict the online and offline time. Using availability monitoring approaches in section 4.3, the predicted values are used by the data owner peer to calculate data availability for a particular duration. The data owner replicates data to peers that are available during that time duration.

**Evaluating the algorithm:** Evaluations were performed using the proposed algorithm and existing availability-based approaches. I show how the requirements for replication, such as availability of data, reliability of the replica set, and lookup failure are improved. The evaluation was performed using live traces of a P2P network, and the details can be found in section IV. With the evaluation I have shown that my approach requires a fewer number of replicas to maintain the high availability of data, compared to other approaches. This reduces the replication overhead of transferring data to unnecessary locations. I have shown that the reliability of the replica set chosen for replication is higher compared to other approaches. My
replication algorithm distributes the load evenly, and does not overload the data on only highly available nodes. It utilizes the availability of lesser available nodes as well.

### 1.3 Thesis structure

The sharing of data between users and developing a support system based on their activity in the network has been very inspiring not only for the P2P network community, but even social network systems. The study of user behaviour can provide an abundance of information that can be useful for planning and optimizing resource usage.

Through the analysis of the current systems and developing the requirements of the thesis, I developed the core section of the thesis with algorithms inspired from multiple domains. The thesis captures the reliability and availability analysis contribution from reliability theory. The prediction model is used in financial systems, and shares and maintains this information using the core P2P system data sharing algorithms.

The complete outline of the thesis is as follows: Chapter 2 gives a background to P2P networks. Chapter 3 provides an introduction to data replication. Chapter 4 and Chapter 5 provide the background to my approach for improving data availability using reliability analysis and prediction model based on time series analyses and Monte Carlo simulations. Chapter 6 presents related work regarding replication in P2P networks. Chapter 7 presents replication approaches that utilize node behavioural data to place replicas. Chapter 8 gives an overview of how the prediction model used in my work has been used in a reliability analysis in the computing domain.

At the core of the thesis I focus on the design and modelling of the system based on the reliability and the prediction models introduced in the background chapters. The core includes developing algorithms as well as integrating data from different sources and models. The data analysis phase is given in Chapter 9. Chapter 10 presents a behavioural model for the P2P network and gives examples for the prediction model to be used for analysis. Chapter 11 describes the model for the P2P network based on the reliability theory, and Chapter 12 elaborates on the data sharing and replication algorithm using the two models.

Based on the algorithm, an analysis was performed to determine the appropriate number of replicas, message overhead, and failure and reliability analysis. Chapters 13 and 14 contain
evaluations performed using the two predication models. Finally Chapter 15 summarizes my findings and concludes the thesis.
Part I

Background
Chapter 2

P2P overlay networks

With the development of file sharing systems, there has been a paradigm shift in the usage of the internet. The peer-to-peer technology has been used for not only file sharing purposes, but also applications related to communication. For example, the widely used Skype utilizes a peer-to-peer technology to develop its communication network. Apart from having to bear criticism for violating copyright and increasing piracy, legitimate business models have been developed based on this technology, including telecommunication and social networking applications.

A Peer-to-Peer network is an architecture consisting of large networked distributed systems. It provides the ability to share the load between equally privileged users. In terms of application, Peer-to-Peer architecture provides the ability to utilize resources from un-utilized users around the world. These resources range from disk storage to bandwidth and processing power. Before the development of this type of network, it would have been accomplished with dedicated machines based on a central server architecture. The previous centralized approach has the problem of bottlenecks and issues of privacy.

From a technical point of view, Peer-to-Peer technology is about management of individual unreliable resources dynamically bound together based on principles in order to meet a set of requirements. This approach has led to a network without global knowledge or central control.

Development of the P2P network has led to the discussion of usage of central control to facilitate the network. Central repositories have been previously used to maintain users and aid routing of data across the network. Most of these dedicated servers are used for user maintenance in the network. In addition, these central servers are a single point of failure.
and bottleneck, and pose a threat to shareability of the overall architecture. Based on these problems, the development of a distributed approach is needed to overcome the issues of using dedicated servers. Maintaining user information at a dedicated server is also a breach of user privacy. A P2P network is divided into the following network strategies:

- Unstructured network.
- Structured network.
- Super-peer network.

The following sections briefly provide details of these networks.

2.1 Unstructured networks

In unstructured P2P networks, nodes connect arbitrarily to the network. With the passage of time, the joining node develops its neighbour table by collecting the neighbour tables of another node. To retrieve data in a network, a node floods the network with the query. Based on the time to live set for the query, the query is forwarded from one node to another in the network. This approach of data retrieval suffers from the problem of incomplete information retrieval, and the query may not always be resolved. Because popular content is available on many nodes and the query is very likely to be resolved for popular data, rarely used data would be missed by query forwarding, as it is available on only a few nodes. Another problem with the flooding is the high amount of traffic generated due to unnecessary query forwarding. A general protocol used by a node joining the network is that it is given a set of bootstrapping nodes. The joining node communicates with each of the other nodes one at a time and obtains a list of more nodes. Based on the capacity to have another neighbour, a link is established between the nodes from the given list and the joining node.

Gnutella [48], which was one of the earliest unstructured networks, avoided centralized indexing and used an approach of flooding/broadcasting to query the network. The work carried out in the paper [48] addresses the network creation and topology of the network. The overall Gnutella network exhibited small-world clustering using the mechanism of a preferential attachment [76] growth model. This preferential attachment approach is based on the way in which new nodes connect preferentially to nodes that are well connected to the network. The protocol of the Gnutella network allows the new node to first connect with the few known servers. The Gnutella network maintenance is based on two messages: ping and pong. The
ping message is sent by a host to announce the arrival to the network, and the pong response is sent in reply to the ping, with the responding node IP address and number and size of files shared. The lookup of data is done based on time-to-live (TTL), which is the the number of hops the query message must travel in the network. With each query message traveling from one node to another, the TTL decreases until the value reaches 0. The query forwarding stops after TTL reaches 0. Each host then forwards the query to the neighbours. The query response is sent using the same path as taken to forward the query.

2.2 Structured networks

Structured networks have been able to address some of the issues arising in the unstructured networks. A Structured P2P network provides a controlled environment for node overlay topology and placement of data in the network. Data in the network is placed based on a precise location calculation approach, which maps the data to the node in the network, based on the identifier. Distributed hash tables (DHTs) have been the main structured P2P networks. A DHT is a hash table interface based on a key value pair, where each node is responsible for storing some data that has a corresponding identifier key range. Each node maintains a table for a node that it knows, which is called a neighbour table. The neighbour table is used for routing by associating these nodes with addresses. As part of the data retrieval, DHT starts with a lookup operation to find an address for a node holding the required data. Before a direct communication between the requester and provider takes place, the request has to go through several lookup steps and a few hops before reaching the required node.

Query routing is quite efficient and allow for the lookup for a key in O(logN), where N is the number of nodes in the network. A typical query in DHT is restricted to an exact match for a keyword search. Research towards enhancing the query to support range queries [31] are an active part of structured P2P network research. Some examples of structured P2P networks are Chord [81], CAN [18], Pastry [71], Freenet [16], Tapestry [90], and P-Grid [4]. Detailed discussions of some of these networks will follow.

2.2.1 Chord

Chord [81] is one of the most popular DHT-based P2P networks. The nodes in the network are arranged in a logical ring. Chord nodes and data use consistent hashing to assign an m-bit
identifier. Data with key K are assigned to nodes with the same id; otherwise, they are assigned to the next node in the ring. Consistent hashing has the ability to re-map the identifier space based on the number of nodes in the network. Consistent hashing allows for evenly assigning keys to nodes. When nodes leave or join the network of size N, only O(K/N) keys move to or from the joining or leaving of the node. A node is assigned an identifier by hashing the IP address, while a data identifier is generated by hashing the key. The nodes in Chord are ordered into a ring of modulo \(2^m\).

Chord uses indexing tables called finger tables and successor pointers. Finger nodes are arranged exponentially around the identifier space. Each node has a constant number of successor and m fingers. The finger table indexes are arranged in intervals and the i’th finger for a node p, is an interval from start \(2^i - 1\) to \(p + 2^i\), where \(1 < i < m\). The successor of a node p is from the first node in the given interval. As shown in Figure 2.1, the node n points to the first node in \(1/2, 1/4, \ldots, 1/2^{\log(N)}\) of the identifier space starting from n. In a network with N nodes, the lookup cost is \(O(\log N)\). Each node needs to maintain \(O(\log N)\) neighbours.

![Figure 2.1: Chord Routing Identifier Space.](image)

Figure 2.2 illustrates a chord ring with 3-bits identifiers. The white circles represent the available peers in the ring. The finger table of each peer contains 3 intervals. Peer 0 has an interval between Peer 1 to Peer 2, from Peer 2 to Peer 4, and from Peer 4 back to Peer 0. The successor node is the first available node in the interval. When peer 0 receives the request for key 5, peer 6 is responsible for data at key 5, as no node is available at key 5. Therefore the request from peer 0 is forwarded to peer 6.

### 2.2.2 DKS

DKS [25] is a DHT-based P2P network, that has the same structure as Chord. It uses a distributed K-ary search, where the routing region is divided into k equal intervals. The suc-
P2P overlay networks

cessor table is enhanced to point to not only a nodes successor, but also a node ranging from \( n + (2^1)...n + (2^L) \). At each step in the routing, the distance between the current node and destination is halved. This yields \( O(\log_2 N) \) hops at worst, where \( N \) is the number of nodes. With \( K = 2 \), DKS routing has the same characteristics as Chord.

DKS nodes exchange routing information with lookups. Most systems use periodic stabilization, where each node periodically probes all of its routing entries for changes. The topology is maintained by two approaches, Correction-on-change and correction-on-use. Correction-on-change is fast, reactive, and inexpensive. Correction-on-use is a lazy approach, where updates are only made upon demand. Correction-on-use is based on the assumption that the routing tables are normally out-of-date entries, and up-to-date information can be retrieved with exchange of lookup messages.

Each node in DKS has a routing table of size \( \log_k N \). The routing table contains \( k-1 \) levels, where each level has a view interval of the identifier space. Each interval at a given level has a size of \( N/k^L \), where \( L \) is the level number. Each node maintains a routing table of size

![Chord Routing Structure](image)

**Figure 2.2:** Chord Routing Structure.
\[(k - 1) \log_k N\]. The first node of the interval is the successor node. For each level \(l\), \(V(l)\) is partitioned into \(k\) intervals \(I(l, i)\), for \(0 \leq i \leq k - 1\). \(I(l, i) = [n + i(N/k^l), n + (i + 1)(N/k^l)]\)

Each level \(l \in L\) has a view of identifier space of \(V(l) = [n, n + N/k^{l-1}]\). With network size \(N\) of 64, \(n = 0\) and \(K = 4\), \(V(1) = [0, 0 + 64/4^1] = [0, 64]\). \(V(2) = [0, 0 + 64/4^1] = [0, 16]\). \(V(3) = [0, 0 + 64/4^2] = [0, 4]\). Figures 2.3 and 2.4 show routing intervals for DKS node 0, and network size of \(N=64\) and \(k=4\) at different levels. The network is divided into \(k=4\) levels. In each level there are \(k\) intervals. The following bullets list the intervals for level 1:

- \(I(1, 0) = [0 + 0 \times (64/4^1), 0 + 1 \times (64/4^1)] = [0, 16]\).
- \(I(1, 1) = [0 + 1 \times (64/4^1), 0 + 2 \times (64/4^1)] = [16, 32]\).
- \(I(1, 2) = [0 + 2 \times (64/4^1), 0 + 3 \times (64/4^1)] = [32, 48]\).
- \(I(1, 3) = [0 + 3 \times (64/4^1), 0 + 4 \times (64/4^1)] = [48, 0]\).

The routing table at level \(l\) and view \(i\), \(R(l, i) = < n + i(N/k^l), \text{succ}(n + i(N/k^l)) >\). The routing table for node 0 per interval is shown in Figure 2.5. The squares in between the index line represent the end of the intervals. Black circles are nodes in between the intervals. The routing for an Id starts first with a lookup from the predecessor of the lookup node. The test is to lookup the Id in the interval of the predecessor and the lookup node. If the id is not
Figure 2.4: DKS level 3 intervals.
found, then the lookup node forwards the query to the nodes in the routing intervals. Figure 2.6 shows how the lookup for id 20 from node 0 takes place. Node 0 checks the predecessor index, and then forwards the request to nodes 24 and 36. Node 24 finds 20 in its predecessor index. As node 24 is responsible for the data for Id 20, and no node is available at Id 20, the lookup is completed.

### 2.2.3 Kademlia

Kademlia [53], is a DHT-based P2P network, that uses a non-Euclidean distance matrix. The distance between two objects is measured as a bitwise exclusive-or (XOR) of their Ids. Each node in Kademlia maintains a routing table with $\log N$ buckets. An i’th bucket in the table contains up to $k$ entries with a distance between $2^i$ to $2^{i+1}$ from the current peer. The routing table contains the IP address of the node and the last time of contact. The node consistently updates its buckets by using a least recently used approach to delete old entries. Figure 2.7 shows the 4-bit identifier space. Kademlia treats nodes as leaves in a binary tree. Starting at the root for a given node, the space is divided into a series of successive lower subtrees that do
not contain a node. A node in Kademlia maintains at least one node from each subtree. For each subtree there is a k-bucket.

The lookup in Kademlia searches for a node with the closest Id to the Id of the data. The node queries the node closest to the data Id in its routing table, and then it learns about a

Figure 2.7: Identifier space for Kademlia.

Figure 2.8: Routing in Kademlia.
much closer node. This process results in $O(\log N)$ steps. As shown in Figure 2.8, each node maintains a bidirectional indexing of each other in the routing table. In the example, if node 0 is looking for Id 15, then it forwards a lookup request to node 8. Node 8 has node 12 in its routing table closest to the Id for the lookup. Node 12 then searches its own routing table for the closest and finds node 14 in its table. The request is sent to node 14. Since there is no node 15, node 14 is responsible for Id 15. Kademlia supports parallel lookup. When the parameter for parallel lookup is set to one, Kademlia’s lookup is similar to the lookup in Chord.

### P-Grid

P-Grid [4] is a P2P data management system based on building a virtual distributed trie. Data keys are composed by a number of bits. The data key space is recursively bisected so that the resulting partitions carry approximately the same load. One or more peers are associated with each partition. Each partition is uniquely identified by a bit sequence. The bit sequence of a partition is called the path of the peer associated with the partitions. These bit sequences induce a trie structure that is used to implement prefix routing by resolving a key lookup one bit at a time. Each peer maintains one or more randomly selected references to a peer for each bit position of its path that has a path with the opposite bit at this position. In prefix-based routing, a match is made with an increasing number of destination addresses per hop, until the destination is reached. For a given destination address of 110001011, the prefix routing starts with 1XXXXXXX. It next takes the path of 11XXXXXX, 110XXXXX until the destination values is reached, where X is a wildcard. The P-grid network is established based on a self-

![Figure 2.9: P-Grid Network.](image)
3 is responsible for the data in the key with prefix starting with 1. When a request for key 101 is received by Node 1, it will be forwarded to Node 3, as the prefix of the request is matched with the data available held at peer 3.

Epichord [41] is a P2P network based on Chord. It improves the lookup of data restricted in Chord to $O(\log N)$ by providing a reactive routing state maintenance strategy and improving resilience, by issuing parallel lookup queries. It provide $O(1)$-hop lookup in lookup-intensive workloads, and under churn, at least $O(\log N)$-hoops for a lookup. Epichord has the same architecture as Chord, with some extensions. It also maintains a predecessor list of $k$-nodes. Nodes communicate with their neighbour to exchange successor and predecessor lists. Epichord maintains a cache to guarantee at least an $O(\log N)$ hop performance instead of finger a table. It adopts two approaches to update routing entries. First, by approaching, a node upon joining the network, it receives the entire cache from its two immediate neighbours. Second, by updating the cache when queried for data in each query, a node not already in the cache is added to the cache. Epichord initiates $p$ parallel lookup by directing to both successor and predecessor. Each entry in the cache has an associated time. Node entries are cleared when associated nodes do not respond to a number of queries or the lifetime has exceeded the time limit $\tau$.

2.3 Super-peer network

The super-peer network is considered to be a distraction from the core P2P network design methodology. Compared to unstructured and structured networks, that are considered to be pure P2P networks, super-peer networks are a hybrid between client-server systems and a pure P2P network. With features of client-server systems, some nodes known as super-peers are dedicated servers for some sets of nodes and perform some extra operations, such as indexing, access control, query routing, and data management.

Super-peers are connected through a pure P2P network approach. Super-peers are elected based on their superior resources and are replaced upon failure. Super-peers act as a gateway to a subset of nodes. When looking for data, queries are routed directly to the relevant node in the subset. If the relevant node is not found in the subset then the query is routed to one of its neighbour super-peers. Some of the main benefits of super-peer network are the ability to efficiently route data, and better control of access to nodes. Fault-tolerance is one of the main problems with a super-peer network, and super-peers are single point of failures for the subset
of nodes. Some super-peer networks include, Napster [3], Gnutella (The Gnutella Protocol Specification v.0.6), and Edutella [58].
Chapter 3

Data replication

In this chapter, I present an introduction to data replication strategies in P2P networks. This section covers an overview of data replication, including traditional pessimistic replication and optimistic replication. The main focus will be on an optimistic replication strategy, as it provides a solution for a dynamic environment as well as and important replication parameters to be addressed and techniques to be used for replication in a P2P network.

3.1 Basic concepts

Data replication is a process that allows maintaining multiple copies of data, sharing resources, and ensuring consistency between redundant resources. The minimal replicable entity is an object. The definition of an object can vary from a specific field in a table to a file. A copy of an object that is stored in another location is called a replica. The sites to which it is stored are referred to as nodes.

There are a range of replication types available, such as database replication, disk storage replication, memory level replication, and file-based replication. In database replication, multiple copies of a database are created. In this type, there is mainly a master/slave relationship between the original and the copies. Updates are passed from the master to the slaves. Although a multiple master, approach is also supported, this leads to an increase in the complexity of maintaining consistent content as well as transaction management issues.

Disk replication leads to the replication of a block of data from the disk to another location. Memory level replication allows sharing of memory in a distributed way. Many nodes share
the same page of memory, resulting in all nodes maintaining different copies of the same page. At the operating system (OS) level, files are managed as blocks of data. Disk level replication allows for block level replication, and OS manages the synchronization between these blocks. Whereas disk-based replication handles block level replication, a logical level replication of files is performed by file-based replication. This approach of replication considers replicating files to multiple location and maintaining synchronized copies of them. Data replication addresses the problems in a distributed system related to data availability, fault tolerance, reducing communication overhead of forwarding queries, increasing scalability, and increasing throughput.

The two important parameters involved with these replications are, Where and When [29]. Where addresses which replicas can be updated, and the parameter of When addresses the time at which the updates are propagated to the nodes. The strategies to be selected for these parameters depend on the characteristics of the network, such as with database replications, where the parameter can have single-master or multiple-masters solutions. When a parameter corresponds to an update propagation strategy in the computing world, they can be of two types, synchronized and unsynchronized. Data maintained by a node exits in different states, which helps in determining the consistency between the nodes for specific data.

### 3.2 Replication parameters

The following are some of the challenges in a distributed systems environment faced by replication approaches:

- **Data Consistency**: consistency of data in a replicated environment is very important. Domains requiring high precision of data retrieval need data to be strictly consistent for efficient and accurate retrieval.

- **Data Availability**: due to the number of nodes leaving the network in the process of a lifetime, the data have to be replicated to further locations in order to maintain a specific level of data availability.

- **Maintenance overhead**: this is the overhead of replications including space and processing for moving data. Multiple copies of files have to be stored, which results in the consumption of extra communication usage.
• Write performance: multiple updates need to be made to replicated copies, resulting in a slow write.

3.3 Synchronous and asynchronous replications

Synchronous replication is also referred to as eager replication, which attempts to maintain all replicas synchronized by updating all replicas in a single transaction. In contrast is asynchronous replication, which is also referred to as lazy replication, asynchronously transmits replica updates to other nodes after replicating transaction commits. Data access and updates are performed through transactions. Transaction is defined as a sequence of server operations guaranteed by the server to be atomic in the presence of multiple clients and server crashes. The update transaction in the synchronous environment results in changes to all of the replicas, which allows for maintaining a consistent state between the replicas. Consistency between replicas can be achieved by using various consistency achieving protocols with time stamp-based algorithms or two-phase consistency algorithms. Synchronous replication allows up-to-date data for each read operation, by maintaining updated copies on nodes, and avoiding divergence among replicas. The major drawback for the synchronous approach, is the synchronous update to replicas. If an update to a single replica cannot be made, then the transaction fails, which blocks the update operation. Due to this synchronous replication, it is not suitable for dynamic networks. In addition, the extra overhead of response time and communication cost does not allow scaling of the synchronous replication.

Asynchronous replication relies on the consistency assumptions of optimistic and non-optimistic approaches to consistency. The optimistic approach to consistency, also known as eventual consistency, relies on the theory that replicas are guaranteed to converge given that the system has been running for a long period of time. Replicas are assumed to have started in the same state and conflicts are assumed to rarely occur. The non-optimistic approach assumes that update conflicts are likely to occur and implements a propagation strategy to prevent update conflicts. The main advantage of an asynchronous approach is that the update operation is not blocked due to a missing replica, as updates are propagated at a later time. This reduces the load of coordination between the nodes, and updates can be sent at any appropriate time, such as a request upon a demand of for a replica. This handles dynamic networks where nodes leave and join a network abruptly. The main disadvantage of an asynchronous approach is that a local read may not return up-to-date values, and replicas may not diverge. The major
advantage of non-optimistic replication over optimistic replication is the high probability of finding up-to-date values upon a local read.

### 3.4 Single-master and multiple-master replications

These replications are the possible scenarios for data replication required for replica management. Single-master replications address situations where a master copy of the data is maintained at a single location, called the master. Multiple-master replications are where multiple copies of the data can be maintained by multiple clients or consumer servers, which have read-only replicas. The server maintains change logs for the data at the master server. Due to the layout of the configuration, the concurrency control is simplified, as only one site has to be updated, which propagates the updates to the replicas. This leads to the issue of single point of failure and potential bottlenecks. Therefore, a possible failures of the master server would lead to the loss of data.

In multiple-master replications, several master servers maintain read-write data permission. Based on the multiple master servers there can be multiple clients. As in the previous approach, the client holds the read-only copy. This approach improves data availability upon server failures, but requires concurrent updates, and maintain of a high level of consistency on the master servers.

### 3.5 Partial replication, full replication, and erasure code

Based on the dynamics and popularity of data in the network, replication has be parameterized to adopt full or partial replication. In full replication, all of the replicas maintain the same copies of data as the master server. The replica node keeps updates of all those data from the master. The overhead of communication is very high using this approach. It is not possible to maintain consistency in a dynamic network, as either the master or the client node may leave the network before the updates are complete. The bandwidth limitation affects the data transfer between the nodes as well and the Time to transfer increases with low bandwidth nodes.
Due to the space and communication bandwidth limitations networks rely on optimizing the processing and storage space by using partial replication. In partial replication, nodes replica subsets of data to replica nodes. The criteria for choosing the data to replicate varies based on the constraints that a network faces. Some networks utilize data usage distributions [17] to determine popular data to replica. Partial replication reduces the overhead of replication to a subset of data. The communication usage overhead is also reduced.

An erasure code-based replication [85] is a technique that partitions the data to be replicated into fragments and distributes these fragments to varies nodes. An object $O$ is divided into $m$ fragments and recoded into $n$ fragments, where $n > m$. The original object is reconstructed from $m$ fragments. These $n$ fragments are distributed, and to generate the original data, $m$ fragments are required. The effective redundancy factor $k_c = n/m$, where the rate of encoding $r$, is $r = m/n < 1$. This rate of encoding increases the storage cost by a factor of $1/r$. For example, an encoding rate of 1/2 and a block divided into fragments $m=8$ would lead to 16 fragments, thereby increasing the storage cost by a factor of two.
Chapter 4

Fault management

Fault management and fault-tolerant computing [68][75] has been studied for a long time. Initial work has focused on electronic and micro-electronic equipment. Fault-tolerant computing means that a system maintains a certain level of performance under the presence of faults. This level of performance comes at a cost of providing the tolerance, reliability, and availability. The study of these parameters for reliability are based on the theory of probability, reliability, failure rates, and failure density functions.

There are various ways to handle faults, such as through functional redundancy with another component, or providing backup support that is only activated upon primary component failure. The reliability of the backup component must be very high. In addition the activity to be performed by the redundant or backup component, aside from providing support at the performance level, can be to repair the primary component.

Failure detection is another important component of fault-management. Detection of failure takes place through voting, where the output of the component requested by multiple other components is matched. In case of fault management in networks, failure in one of the communication paths results the delivery of data through a redundant alternative path. This approach to redundancy and voting comes at a cost to the communication path and the need for extra systems. The redundancy at the message level can also be performed by transmitting data multiple times to avoid data corruption. through transmission, bit comparison data is validated. These techniques are called error-detecting and error-correcting codes.

Attaining reliability and availability in a complex system is a very difficult task. A highly reliable and available system is achieved by considering factors such as cost, performance,
development time, and risk of failure. A design of such a system is based on several key features and desirable features. The system should meet the requirement of key features, but can afford to miss some of the desirable features for achieving a better design.

4.1 Reliability analysis

The system is considered to be a set of components and units when performing a reliability analysis. Each unit has a functional or non-functional state. Various probabilistic approaches are used to compute the system reliability based on the reliability of the sub-components. There are a few configurations of these components studied in a reliability analysis and two of them that are related to distributed systems are discussed here. These components occur as a series or parallel arrangement. Reliability analysis is performed based on this relationship between the components. Reliability of a system is measured by different parameters based on the specific requirement and situation. Reliability can be measured by the mean time to failure (MTTF), failure rate, survival probability (reliability probability), and availability.

4.1.1 Reliability configurations and reliability probability

A system with a series component requires all of the components to be functional. The performance of the system depends on all of the components in the system. Failure or reduced functionality of any component either results in the failure of the system or reduced functional behaviour of the overall system. For a system with an n-series of connected components, this can represent a network of n nodes connected in series or ring. $X_n$ represents the functional state of the component, whereas $X_n'$ represents the non-functional state of the component. If probability that a component is functional is $P(X_n)$, and the probability that a component fails is $P(X_n')$, then the overall probability of success of the system is $P_s$, which is $R$, the reliability of the system. The probability of the system to fail is given as:

$$P_f = 1 - P_s$$  \hspace{1cm} (4.1)

The probability of system success is the intersection of components $X_1, X_2, ...X_n$. The probability of successful operation, which is the reliability of a system, is given as:

$$R = P(X_1, X_2, X_3, ...X_n)$$  \hspace{1cm} (4.2)
Fault management

which based probability theory is

$$R = P(X_1)P(X_2|X_1)P(X_3|X_1X_2)...P(X_n|X_1X_2...X_{(n-1)})$$  \hspace{1cm} (4.3)

The reliability of the system is less than the minimum reliable component. The probability of failure can be calculated based on the unit failures. The overall system failure is the union of failures as.

$$\bar{R} = P_f = P(\bar{X}_1 + \bar{X}_2 + \bar{X}_3 + ... + \bar{X}_n)$$  \hspace{1cm} (4.4)

In a parallel configured system, a failure of one or more paths or components would not result in a complete system failure. Alternative paths will be available for the functioning of the system. The system is operating if any of the parallel paths are available. The system will only cease to operate when all of the parallel components fail or are unavailable. A parallel configuration of components is also referred to as a redundant configuration. The probability of success is given by the union of $n$ successful events.

$$P_s = P(X_1 + X_2 + X_3 + ... + X_n)$$  \hspace{1cm} (4.5)

Probability of failure is the intersection of the component failures.

$$P_f = P(\bar{X}_1\bar{X}_2\bar{X}_3...\bar{X}_n)$$  \hspace{1cm} (4.6)

As $P_s = 1 - P_f$. For dependent components, the probability of success for a system is:

$$P_s = 1 - P(\bar{X}_1)P(\bar{X}_2|\bar{X}_1)P(\bar{X}_3|\bar{X}_1\bar{X}_2)...$$

$$\hspace{1.5cm} P(\bar{X}_n|\bar{X}_1\bar{X}_2...X_{(n-1)}))$$  \hspace{1cm} (4.7)

For independent components, such as in a P2P network, the probability of success is given as:

$$P_s = 1 - P(\bar{X}_1)P(\bar{X}_2)...P(\bar{X}_n)$$  \hspace{1cm} (4.8)
4.1.2 Failure analysis

In this section I address two more parameters for reliability measurement:

- Failure rate
- Mean time to failure (MTTF)

Failure model is devised based on the test failure results, and failure rates are determined based on probability theory. Failure rate is determined based on the sample data, by plotting the failures against time. Based on the plot, the failure density function can be determined. Starting at time $t = 0$, a node fails with the passage of time. The density function for the time interval $t_i < t < t_i + \delta(t)$, is the ratio of the number of failures in the interval divided by the number of nodes, divided by the length of time, given as:

$$f_d(t) = \frac{[n(t_i) - n(t_i + \delta(t_i))]/n}{\delta t_i}$$  \hspace{1cm} (4.9)

Using this density function, the failure distribution function can be calculated as:

$$\int_0^\infty f_d(T) dT$$  \hspace{1cm} (4.10)

Modelling of the failure requires modelling the operational and repair intervals. The MTTF is the time during which the component is functioning. This is used to represent the time until the failure occurs, or the operational interval. The mean time between failure (MTBF) can be used interchangeably for MTTF. Considering the failure time of a component to be $t_1, t_2, t_3...t_n$, the MTTF is the average of the n-failures, given as follows:

$$MTTF = \frac{1}{n} \sum_{i=0}^n t_i$$  \hspace{1cm} (4.11)

In continuing time, based on the probability distribution, the mean time to failure is given as:

$$\int_0^\infty tf(t)dt$$  \hspace{1cm} (4.12)
In probability analysis, this is equivalent to the expected value of a random variable $t$, where $f(t)$ is the density function for failure.

The modelling of repair can be done using a similar approach. In reliability theory, repair time is called the mean time to repair (MTTR) or mean down time (MDT). Considering the $n$-downtime intervals as $d_1, d_2, d_3...d_n$, the mean time to repair is given as:

$$\frac{1}{n} \sum_{i=0}^{n} d_i$$

(4.13)

### 4.2 Availability analysis

In reliability analysis theory, the availability of a component plays an important role in determining the overall reliability of the system. Availability is defined as the probability of the component to be operational at a particular time $t$, represented as $A(t)$. Based on this availability, reliability is the probability that a component has been in operation until $t$. A repairable system is a system where components fail and are repaired. The repair time is equivalent to the time that the node remains disconnected from the network. For nodes connected in a series configuration, such as a ring, the availability $A(t)$ is greater than reliability $R(t)$. For a network with a parallel configuration, the repair effects both reliability and availability functions. Reliability is a function of MTTR divided by MTTF. I will use the average availability in my analysis, which is the average time a node remained connected to the network. The average availability $A_{av}$ is given as:

$$A_{av} = \frac{MTTF}{MTTF + MTTR}$$

(4.14)

### 4.3 Monitoring availability

Monitoring node availability is a very challenging task, considering the distributed nature of a P2P network. Since there is no central control for monitoring nodes in the network that exhibit churn[70], the monitoring of a network has been useful for handling issues with respect to routing for data completeness [51][50] and to determine non-faulty nodes in the network[63][39].
The monitoring of nodes in a network can provide very useful availability information. Using this information, a node can predict future behaviour of its neighbours. To determine a stable set, it is important to monitor the nodes in a constant interval by sending messages, called the heart beat probing method [12] where the sending node expects a response within a given time frame. If the message is not received, the destination node is considered to be offline. Given the overhead of the message after constant intervals, this approach is very simple; however, in order to overcome some of the issues, but to overcome some of the issues, [13] the approach reduces the message overhead by controlling message probe results. Some approaches [69][9] use heart-based probing to determine availability, but this lacks the ability to predict the future behaviour of the nodes.

Monitoring [52] of a network to detect failure has been a core research issue for distributed systems in various applications, ranging from Grid computing, cluster computing, and P2P networks. A hierarchical-based monitoring system is proposed to handle the challenges offered by the domains in order to detect failure at runtime and to share this information. A network is divided into clusters, and each nodes send multicast messages to monitor the state within the clusters. It also uses tree-based point-to-point connections that represent clusters of nodes to aggregate the results. The heartbeat approach is used to maintain network membership. Non-reception of the message signals the failure of a node in the cluster.
Chapter 5

Prediction models

This section details the two prediction models to be used in the replication algorithm for predicting user behaviour for data replication.

5.1 Time series analysis

Time series analysis has been applied to data analysis in multiple fields. It has proven to be accurate for forecasting in the fields of economics and sales. In a time series analysis, sequences of observations are analyzed by statistical methods to determine the dependency between the observed values. Based on this analysis, the objective is to model the behaviour of the stochastic process and forecast future values. The autoregressive integrated moving average (ARIMA) is used to model the non-stationary processes. Autoregressive moving average (ARMA) model is used for a stationary process. The ARIMA model for analysis is based on the work by Box and Jenkins [49].

Stationary processes have values that fluctuate around a fixed mean and constant variance. They hold no trend or seasonal behaviour. A pair of values with fixed lag has the same covariance as two other values with the same distance lag. A white noise process is a random variable of a discrete process that is time independent and identically distributed. The expected value and variance is constant. A white noise process becomes important when it is considered as a component of moving average processes within ARIMA models. A non-stationary process on the other hand has trends, seasonal behaviour, or both.

The forecasting approach using the ARIMA model undergoes the following three steps:
• **Model identification**: Using Auto-correlation function (ACF) and partial autocorrelation function (PACF) for determining the dependencies between data. Apply differentiation for a non-stationary process to be converted into a stationary process.

• **Determining parameters**: by fitting the identified model to the observed data. This is to determine the coefficients of the linear combination. There are a number of techniques which can be used to determine this value, such as maximum likelihood estimation and least squares (ELS).

• **Prediction**: The final step in forecasting is to predict the future value.

### 5.1.1 ARIMA

The ARIMA model is represented by ARIMA(p,q,d), where the d parameter defines the number of times the differentiation is performed to reach stationarity with p autoregressive (AR) terms and q moving average (MA) terms. An autoregressive model (AR) is based on the idea that a the current value of the series is calculated from p past observations. The value of p determines the number of values in the past needed to predict the current value. For time series \( y(t) \) of order p, the auto-regressive process is denoted by AR(p) given by:

\[
y(t) = a_0 + \left( \sum_{i=1}^{p} a_i \ast y(t - i) \right)
\]  

(5.1)

It is a linear combination of p coefficients \( a_0, a_1 ... a_p \). The current value is the sum of the product of the coefficient and the past values of the series and the sum with the current coefficient. The moving average is simply the average of the last N observations. The moving average of order q, is denoted by MA(q) and given as:

\[
y(t) = e(t) + \left( \sum_{i=1}^{q} b_i \ast e(t - i) \right)
\]  

(5.2)

which contains q white noise factors \( e(t), e(t - 1), ..., e(t - i) \) and weight factors \( b_i \).

A model based on the mixture of these two processes, and based on stationary series, is represented as an ARMA model consisting of two parts, autoregressive (AR) and moving
average (MA):

\[ y(t) = a_0 + a_1.y(t-1) + a_2.y(t-2) + ... + a_p.y(t-p) \\
+ e(t) + b_1.e(t-1) + b_2.e(t-2) + ... + b_q.e(t-q). \] (5.3)

The \( p \) and \( q \) parameters of the ARMA model are determined by using a model identification algorithm. The coefficients are determined by using the parameter estimation during the second step. A time series function is a linear function of past values and random shocks.

Before parameter estimation can be applied, the non-stationary process has to be converted into a stationary series. A series with trends can be converted to a stationary series using iterative differentiated pair values. A detailed discussion on the modelling using an ARIMA time series model is provided in Chapter 9.

### 5.2 Monte Carlo simulation

A Monte Carlo (MC) simulation is based on the generation of repeated sampling to reach an expected value of a random variable. A MC simulation is a systematic trial and error method for solving complex problems. A MC simulation relies on 2 steps:

- **Modelling**: This steps involves modelling the possible project inputs and outputs
- **Random sampling**: Based on the input distribution for the random variables, iteration is performed until the expected outcome is reached

In an MC simulation, simulations are performed based on the model generated. An MC simulation is not an optimization technique, as it does not result in an optimal solution. Rather, it is a numerical technique that models a probabilistic system to predict a system’s behaviour. Based on the input of probability distribution, the MC simulation is expected to reach the expected result. This distribution is estimated based on trails. After a sufficient number of trails, the outcome distribution of these trails is examined to approximate the solution distribution. These trails help understand what combination of input values will lead to the output expectation. This is a single value deterministic model.

The simulation process for an MC simulation follows an iterative process, controlling many possible solutions. The process starts with the generation of many possible trails. Each step generates a possible behaviour of the project. The repeatable process creates many cases
until a predetermined number of trials or a stopping condition is reached. The following are the steps that a usual simulation process adapts:

- Determine the probability distribution of the random variable. This distribution follows a standard distribution, such as, Weibull, Poisson, or exponential-based on the historical analysis of data.

- Conversion of the frequency distribution to the probability distribution.

- Random sampling from cumulative probability distribution in order to determine the specific variable values in the simulation.

- Perform the simulation for a large number of observations for the operation under analysis.

Figure 5.1: Monte Carlo simulation process.
Analysis of the log calculation is performed to average the expected values of the trails and frequency distribution. Averaging the resulting log values gives the expected values for the project. The precision and expected value improves as the number of trails increases. A histogram can then be generated based on the averaging values to determine the shape of the density function, or the data can be used to determine the cumulative frequency distribution. The shape of the frequency distribution approximates the shape of the solution probability distribution. The choice of frequency distributions is important for the overall simulation of the system. This distribution is determined based on the historical record.
Part II

Related work
Chapter 6

Replication in P2P networks

Based on the parameters presented in the previous section for data replication, different networks adopt these parameters to improve the challenges faced for data availability, fault tolerance, and improved search. P2P networks replicate data to reduce lookup time and improve data transfer time by replicating data to nodes that are geographically closer to the consumer/client of the data. The P2P network uses replication to reduce the probability of losing data due to the nodes leaving the network. As presented above, replication comes at a cost of storage and maintenance. Updates for the data are either initiated by the owner of the data (push-based updates) or the client requests for those updates (pull-based updates). Nodes in a P2P network are autonomous with updating and deleting data stored in them. This can result in stale data if not updated properly. Data are often associated with a timeout value, as the a P2P network tends to provide soft-state guarantees. In soft state replication, data are not stored durably, and therefore reconstruction is needed at some costs.

Another important aspect for replication in a P2P network is whether to select data for replication or only replicating indexes of those data. Replication based on the indexes reduces the storage requirement and data transfer overhead, but the challenge for achieving high availability and reliability is not addressed in this approach. Instead, data replication instead of index replication improves data availability and reliability, but at a cost of storage and data transfer cost. In the following sections, I present a survey of replication approaches used in unstructured and structured P2P networks.


6.1 Replication in unstructured P2P networks

Replication strategies for a P2P network depend on the type of strategy used for locating the data. The strategy to be used in the unstructured P2P network depends on the following parameters:

- The number of replicas required.
- The location to place the replicas.

6.1.1 The number of replicas required

The following two common ways of maintaining the number of replicas are:

- Owner replication: in this approach, the owner who initiated the query request maintains a copy of the replica, such as in Gnutella.

- Path replication: in this approach, which is used in Freenet [16], the query keeps track of the nodes it has to pass through until there is a successful retrieval of data.

Replication in Gnutella as shown by Edith et al. [17], [47], used Erdos-Renyi random graphs[26] to model the P2P overlay network. The requesting node initiates the replication of data. Uniform replication is the simplest approach for replicating data, whereby the same number of replicas are created for each object. The main drawback of using this approach is that unpopular data are replicated in the same number, leading to a waste of space. Proportional replication[82] is the optimal configuration for minimizing search time, as search time is the shortest distance from the inquired node where the replica was found. The Proportional replication approach eventually leads to a number of replicas that is proportional to the requests for the object.

Both Proportional and Uniform replications have the same average search sizes, which suggests that the average search size is independent of the query distribution. It has been shown that to achieve minimized query probations to nodes in the network, square-root replication provides an optimal configuration. Let \( P_i \) be the number of replicas and \( Q_i \) be the query rate for the data where each replica of data is proportional to the square root of the query data \( P_i \propto \sqrt{Q_i} \). In case of a node that has no more replica placement capacity, various refreshing strategies, such as first-in-first-out (FIFO), least-frequently-used (LFU), least-recently-used (LRU), or random deletion, can be applied to deleted items. Since global information is not
available at a node, different nodes use simple distributed protocols to achieve the square-root replication strategies. The number of copies created is the number of nodes searched in a successful search.

In path-based replication, each node in the path is required to maintain a copy of the data. In Freenet, each node maintains a pointer to the original node as well. The replacement/deletion policy used by Freenet is based on LRU. Although the data are deleted, each node maintains a pointer to the original node. This pointer can be used in a later search. A path-based replication is most similar to the square-root replication strategy.

Path replication can only be used on searches that are based on random walks. In some cases of a random walk, it fails to determine the search size, as there can be multiple walkers. Only those nodes where a query has been successful are selected for replicas for the path, while the rest are ignored. The pull and push strategy [42] is an approach, where the inquiring peer is responsible for the replication of data. First, the querying node pulls the desired data, and then in the push phase, it sends the data around for the other nodes to hold the data. In order to achieve square-root replication, the number of nodes in the push phase should be equal to the number of nodes in the pull phase.

6.1.2 Where to place the replicas

In the previous section, we looked into maintaining a consistent number of replicas in the network in order to minimize the search size. The second most important issue in replication is where to place the replicas. Based on the initial unstructured P2P overlays, Gia et al. [15] proposed to utilize the node heterogeneity and to use an adoptive overlay topology and search algorithm. They proposed a one-hop replication strategy, whereby the index of each node is replicated to its neighbours. This approach ensures that the high capacity nodes are the ones that have a high degree. As queries reach the high degree nodes more frequently, the probability of finding the data increases if the indexes for the data are replicated to those nodes.

Power-law network-based [34] replica placements use flooding, percolation-based flooding, random walks, and high-degree random walks. High-degree random walk, are walks in which nodes select nodes randomly from their neighbours that have highest number of neighbours. As the spreading the location information spreads along the high-degree random walk, more information reaches high-degree nodes more quickly. It has be shown that a search query
gravitates towards the high-degree nodes in the network. They\cite{34} only considered replicating indexes to improve overall performance.

The local minima search (LMS) approach \cite{54} to replication is based on the idea that data and nodes receive identifiers from the larger set. The replication approach places data at nodes that are close to their identifier. Such nodes are called the local minimum for the items, where the node is $h$-hop away from the node, and $h$ is the parameter set for the network. The process of replication starts with a random walk to find the closest Id for the data. If the node is found and the node does not have the replica, then a replica is created, otherwise the process continues with a random walk length that is doubled. The same approach is used to search for the item.

\section{Replication in structured P2P networks}

Replication techniques are used for both workload and data balance in structured P2P networks. Structured P2P networks use replication to handle availability upon node failure. Replication is also used to improve scalability and performance of the network. Replication leads a constant search time in most cases, as structural networks provide logarithmic search time for a successful search. In DHT-based P2P networks, replicas are $k$ neighbours away from the owner of the data, where $k$ is the number nodes to which data is replicated. Nodes closer to each other in the overlay are more likely to be geographically closer to each other, since the Id of the node is based on the hash of their IP address. Data can be replicated to multiple locations in the DHT by a multiple hash function. Therefore data are mapped and stored in more than one place. This improves availability and reduces latency, as data can be retrieved from a geographically closer node. A DHT also provides caching or storing data to the requestor, which is similar to the path replication and owner replication in unstructured networks.

\subsection{CHORD}

Chord does not provide any replication approach; this is left for implementation at the higher application level. Multiple hashing \cite{36} can be used where data are mapped and stored at multiple locations based on the key generated by varying the hash parameters. Another approach to replication\cite{19} in Chord is where a node can store data to $k$ nodes succeeding it.
Each node in Chord maintains a list of \( r \) nearest successors in the Chord ring. When a node’s successor fails, it replaces it with the first live node from the successor list. Figure 6.1 shows replication factors \( k = 1 \) and \( k = 2 \). It can be seen that with \( k = 2 \), the successor node holds data from the previous node. Each node is responsible for the immediate successor data. With replication factor 2, each node is responsible for data for 2 nodes. As shown in Figure 6.1b, node 0, holds its own data as well as data from Node 6. The extra communication cost is one of the problems with this approach where a failure of a node would have to get updates from all the replica nodes.

### 6.2.2 DKS

DKS uses symmetric replication [27]. This replication approach is based on having replicas nodes to \( k \) closest neighbours, for a given replica degree \( k \). This approach allows concurrent access to any specified replica. Multiple requests to different replicas do not have to pass through the same node. Distributed voting can be used to compare the results and increase security. DKS also provides proximity-based neighbour selection. The replication approach follows the idea that each identifier in the system should be associated with \( f \) other identifiers. Based on this association, any data with identifier \( i \), having association with \( j \), and storing data for \( j \), should also store data for the \( i \) identifier. The identifier space is partitioned into \( N/f \) equivalence classes, and identifiers in the same equivalence class are associated with each other. To search for data with identifier \( i \), requests will be made to any of the identifiers associated with \( i \). For the symmetry requirement to always be true, the replication factor \( f \)
Figure 6.2: Symmetric Replication with factor 2.
has to divide the size of the identifier space N. As shown in Figure 6.2, the replication factor of 2 for node 16 leads to 8 equivalence classes. Node 0 is responsible for data keys that are responsible for its own self keys (15, 14, and 13) and data from Node 8. Node 0 holds data from equivalence classes of the nodes for which it is responsible for as well.

### 6.2.3 Kademlia

A Kademlia network can implement any DHT-based resiliency approach. Another approach to resilience for Kademlia is based on the lookup procedure. Kademlia supports parallel lookup with the support of active replication. Kademlia looks for the k closest nodes to the key (Id). It sends a STORE message to create k replicas of the data. As shown in Figure 6.3, data with key 1111, and replication factor k=3 are stored in the nodes with the closest match to the key. The key for 1111 is stored at nodes 1111, 1110, and 1101. Therefore, any lookup for key 1111 can be resolved if the request arrives at one of these nodes.

It also uses an approach similar to the path-based replication approach by replicating to a node that is closest to the key, but does not contain the required data. This helps to converge the lookup along the same path, as placing data on the lookup path leads to a faster search, and helps to avoid hot spots. Kademlia ensures the freshness of the replicas through periodic re-publishing.
6.2.4 P-Grid

P-Grid implements two forms of replication for fault-tolerance. First, multiple nodes are associated with the same key space. This is called structural replication. Then, multiple references are kept in the routing tables, which provides alternative access paths. As shown in Figure 6.4, Nodes 3 and 4 are responsible for the same set of data with keys starting with 10. When multiple nodes become responsible for the same key, it provides fault tolerance in case of churn, when one of them leaves the network. The second form of replication in P-Grid is based on the routing table references. Each node maintains a reference to an alternative path. As shown in Figure 6.4, Node 1 maintains a routing table entry to the node on the other side of the tree, pointing to the key starting with a prefix of 1 and containing 01. This helps in path lookup in case of node failures as well.

6.3 Other replication strategies in a P2P network

Beehive [66] replication is also a DHT-based P2P network. It provides a constant lookup performance for common Zipf-like query distributions. It also provides O(1) lookup performance, with low storage, bandwidth overhead, and network load requirements. In addition, it improves latency by using static caching, and adapts to the changing environment based on failure and changing popularity. Beehive provides a replication framework on top of any DHT
that uses prefix-routing, where each node and data are identified by an unique identifier, and each node is aligned into a circular identifier space. Data are stored at a node that is close to its identifier. Based on prefix-based routing, after k-steps, the query reaches the node with k-th matching prefix. Search space is reduced exponentially, resulting in $O(\log_b N)$ lookups on average.

Replication is based on the idea of replicating data along the path of the lookup, by replicating the data on the last node before the destination node. This reduces the lookup latency by one hop. Beehive extends this idea by associating the replication number based on the popularity of data. It assigns replication levels to each data point to control the extent of replication in the network. A data with level i is replicated to a node with at least i matching prefix with the data. Data stored at the home node, being the node with the closest matching Id, is at level $\log_b N$, whereas data with level 0 are replicated to all of the nodes in the network. The objective of a beehive is to assign a minimum level to each data point, such that the lookup is restricted to a constant C hops. This approach assigns highly popular nodes to low level nodes and low popular data to high level nodes. As shown in Figure 6.5, D2 is highly popular data, which is assigned to the lowest level Node 1 with at prefix 012X. It is replicated to two other nodes. Data D4 is less popular and is assigned to a high level node with no replicas.

A path reduction-based replication approach [73] deals with hot spots, which replicates files close to the requester. The approach chooses nodes in the network that are high traffic hubs. Based on the varying popularity of the files and node interests, the replica placements are updated with the underlying approach to select the frequent querying nodes or node that are part of a high traffic path. The work carried out addresses the issue of balancing load between the replica nodes and reducing the lookup path. The approach has a self-adaptive
approach to popularity-based data storage in order to determine the time at which to delete the files when the popularity falls below the required threshold.

6.4 Summary

The replication approaches presented in this chapter for P2P networks address the issue of the number of replicas and where to place the replicas. The approaches, such as path-based replication and replication based on the data popularity, address the varying popularity requirement of data. Replicating only a portion of the data that the node holds reduces the data transfer overhead. However, the placement of the replicas is not determined based on the availability information of the nodes on which the data is replicated. Such placement of data leads to a need for frequent replica management to maintain the required number of replicas. In the next chapter I present replica placement approaches, which perform some availability analyses on the nodes.
Chapter 7

Behavioural Replication approaches

In this section I will introduce some replication strategies that address the issue of data availability. These replication algorithm take into consideration the availability requirements and maintain the required availability by monitoring the network for node failures.

7.1 Replication using node availability distribution

Some of the initial work for data replication was conducted to improve data availability based on the replica management system TotalRecall[9]. The replication algorithm was based on Chord. The goal for their work was to automate data availability management. In TotalRecall, replication of files is performed based on the availability of the nodes. Availability of the nodes was centrally monitored. The replica creation node pings nodes to monitor the states of the nodes. Using the received values, a replication factor is determined. A random selection of nodes takes place for the calculated number of replicas. I evaluated the TotalRecall protocol using the replication equation. Let $\mu_H$ be the mean host availability. To calculate the required replicas $c$ using:

$$A = 1 - (1 - \mu_H)^c \quad (7.1)$$

where $c$ is solved to be,

$$c = \frac{\log(1 - A)}{\log(1 - \mu_H)} \quad (7.2)$$
TotalRecall provides availability prediction, redundancy management based on erasure code, and dynamic repair. It performs eager repairs. One of the main drawbacks has been the extra overhead per file. In addition, the heterogeneity of the nodes in the system is not considered and different nodes would have different bandwidth requirements.

In the replication approach, presented by Kim [37], in which data is replicated to nodes have the least overlapping time, the host node estimates the departure time of its neighbour nodes using their join time and MTTF. The MTTF is calculated based on historical records by the host node. The host estimates its own departure time using the join time and MTTF as well. Calculating the difference in the two departure times gives the overlapping time. The nodes having a departure time that is greater than the host node departure time are considered for replication. The host continues to calculate the lifetime overlaps with its neighbours until the target availability is reached. This approach does not consider the heterogeneity of the nodes as well. Although this approach considers nodes that have less overlapping time, but it does not consider the time required to transfer the data. Some overlapping time is required to transfer data to create a new replica.

A basic model for replica placement with respect to availability is presented in [67][10]. In both of these studies, the authors vary the number of replicas based on the availability factor of the nodes in the network. The availability factor is the failure rate of the nodes in the network. The number of replicas to be placed varies with the probability of node up-time.[10], provides analysis on both replication and erasure coding, and uses the availability factor to further analyse the bandwidth consumption for maintaining the availability. In their approach it is assumed that all of the nodes in the network exhibit the same probability of uptime. However, this is not the case in reality. Using the algorithm, the number of replicas required are calculated and data is replicated to a random set of replica nodes without considering individual node availability.

Xinet al[44] presented a comprehensive analysis for replication approaches based on eager maintenance, deterministic lazy maintenance[9], randomized lazy repair [23], and proactive maintenance[77]. They suggest that different replication strategies are applicable dynamics of the network. The evaluation monitored bandwidth usage, availability of objects, and probability distribution of the number of fragments. Live traces of Skype and synthetic data were used for analysis to vary the parameters of the systems. The evaluation was performed for erasure coding. The result of the analysis suggested that a randomized lazy strategy provides a good overall result. Erasure coding has been considered to be usable in less churned networks, and in cases where relatively large and rarely accessed objects are required as an application. The
replica overhead and the search failure has is higher using TotalRecall replication approach compared to my work.

A data replication problem is proven to be NP-hard using game-theoretic analysis [72]. The replication problem is mapped to the problem of optimizing resource utilization given the number of peers and capacity to hold data of each node. In this work, it is shown that optimizing replication in a central approach is NP-hard. Analysis was performed on two P2P availability phenomena, using the uncertain probabilistic model and model based on diurnal patterns in time slots. Their performed replications based on cliques of peers, where each other’s data is replicated. They raised a load-balancing problem, similar to the problem raised by my study, whereby highly available peers tend to replicate data between themselves ignoring the low available peers. The availability of data for less available peers can be improved by utilizing diurnal patterns of peer availability. This is possible when a system has participants from different time zones. An optimal replica placement algorithm allows peers to create replication agreements, but forces them to have at least one worse-off peer in the group.

A rigorous analysis of machine availability in the line of reliability theory is presented in [60] where three networks (workstations at CS-UC (Santa, Barbara), Condor at Univ of Wisconsin, and Internet hosts survey [46]) were used for the analysis. The analysis was performed using four probability distributions (Weibull, hyper-exponential, exponential, and pareto) of the network for modelling. The data set used from the distributed system fit more closely with the Wiebull and hyper-exponential distributions. The data had a poor fit with exponential and pareto distributions. Using the Wiebull distribution, the authors showed that it is possible to model the aging effect of nodes by calibrating the shape parameter of the distribution. In contrast to exponential and pareto distributions, Wiebull and hyper-exponential distributions use previous parameter estimations and maintain a historical information that improves the accuracy of prediction of availability of the nodes. Similar to the disadvantages of using the Markov model, this approach does not capture the possible change in user behaviour in the network.

Blond et. al[11] presented a availability aware approach to handle the problem of partner matching for applications like task scheduling, where tasks are scheduled on all the online nodes, or replicating to the nodes to decrease the need of replicating more data. An epidemic protocol T-man is used to update the set of best peers, which match the metrics required to be achieved for presence matching and disconnection matching. The approach presented here for calculating the presence time is called the average availability in reliability theory. Where the on-line time of the nodes is divided overall online time of the nodes. We have employed
a very similar approach to calculate the overlapping and non-overlapping time for the nodes. Through analysis of the network they show that some node exhibit similar patterns during the week. Thus simple predictor helps to predict future availability, handling the changing node patterns has not been tested in their analysis. The time series analysis approach handles seasonal behavior, and adjusts to the changing node patterns by updating the autocorrelation and noise parameters.

Replicating data to reduce retrieval time is presented by Zhang et. al[89] The proposed approach provides availability base replica placement takes into consideration the transmission delay due to the geographically distributed nature of the nodes in the network. A clustering algorithm is run in parallel to the replication algorithm. The replication algorithm maintains a number of replicas depending on the probability distribution pattern of the node online time. The clustering algorithm creates cluster based on the data transmission delay time. The nodes meeting the threshold for the data transmission delay are considered to be part of the cluster. Replicas are created for each cluster to reduce the data retrieval time between the requester and the replica node. New data replicas are only created in clusters where there is no data replica available.

Reputation maintenance and determination approaches are becoming increasingly important with the increase in threats from malicious nodes in the network. A P2P based reputation approach is presented in PeerTrust[87], and a replication approach using the reputation approach is presented by Naseera et. al[55], applied to data grid. PeerTrust developed approach to evaluate trustworthiness of the nodes in the network in a decentralized way. The approach takes into account the feedback for the transaction, the feedback from others, the transaction context feedback as to how good the feedback is for critical task compared to non-critical tasks and including incentives for feedbacks. The proposed replication approached, includes how likely the resource fulfills QoS commitments. The commitments fulfillment is determined from the past and current experience. The replica placement determines nodes which are reliable and offer minimum average access-cost (time taken for sending the replica to the requester). Their approach takes into consideration the storage capacity of the node, guaranteed QoS and minimum access-cost.

7.2 Markov model-based approaches

The Markov model is a mathematical model that helps to model the dynamic behaviour of the system based on state-space, where state-space is a graph, and the edges between the graph
have a certain probability of transition between the nodes. With a given probability, the state of the system is changed from one state to another. As shown in Figure 7.1, the idea of using a Markov model is to maintain replica states with a transition probability determined from the analysis of failure and repair rate probabilities. Each state would represent the number of replicas available in the network, given the failure rate $\mu$ and the repair rate $\lambda$ of nodes in the network. State D is the dead state, and at this point there is no replica of the data available in the network. Data are lost from the network when the dead state is reached. If the repair rate is lower than the failure rate, then large number of replicas are required.

![Markov model for replication](image)

To improve data survival in a P2P network, a data replication approach using a Markov chain model-based [84] analysis of MTTF replication was proposed. The study showed that simple replication of data showed better availability compared to erasure codes. Erasure codes require high availability peers, and simple replication performs better with fewer high availability peers. Erasure codes increase the maintenance cost with an increasing number of communication messages. Markov models for each node has three states: connected, temporarily unavailable, and permanently unavailable. The connection duration is given by a continuous random variable. The transition probability follows the rate of failure and recovery of nodes. The Markov model helps to determine the number of replicas to be created based on the expected number of replicas at a particular time, based on the probability that the nodes reaches a connected state after reaching failure or being disconnected. This helps to determine the overall data availability in the network.

The Markov chain model [65] has been used to determine the appropriate number of replicas required based on the dynamics of the system. A set of states based on the number replicas are generated for Markov chain. The transition rate between the states are determined based on the departure rate and arrival rate of the nodes in the network. The proposed approach calculates the number of replicas required by the P2P network based on the state in the Markov chain. Each state represents the available number of replicas. The number of replicas are determined by optimizing the system parameters to maximize a lifetime. An absorbing state is defined where the number of replicas falls to 0, from which the system cannot recover, as there are no more replicas available in the network. The rate of replica creation is determined
based on the repair rate ratio, which is the ratio of the rate of repair to the rate of departure. This determines how aggressively replicas need to be created.

The work conducted in both approaches does not take into consideration the network cost requirements. The bandwidth and storage requirements are also not taken into consideration.

A more accurate Markov model [62] for predicting durability in a P2P network was postulated to show that the model generated earlier had a constant or linear number of replicas. A accurate model has been presented from the analysis of a network that suggested that the parameters values grow sublinearly with the number of replicas. They show that linear expression results in the probability of data loss to be underestimated, with constant expression, resulting in a significant overestimation. Similar to the previous Markov model-based approaches, the states of the model represent the number of replicas available. The transition probability is dependent on not only the MTBF but also on node bandwidth and storage capacity.

Given the state transition probability for data, the availability modelling of the network is required as well to determine the correct nodes for replicating data. The Markov model allows for a determination of the MTTF of the data, but does not suggest the nodes where data can be replicated. Therefore, availability modelling needs to take into consideration the node availability in the network.

7.3 Summary

The behavioural analysis-based replication approaches presented in this chapter evaluate the availability of nodes or data by analysing of the historical data logs. The dynamic replication occurs through analysis of the failure rate of the data in the dynamic network. Markov models, distribution analysis of online and offline times, and reliability-based approaches presented here capture behaviour of the nodes from past historical data. A replication approach requires a feedback system to update parameters in order to adapt to the behaviour of nodes. The current approaches lack the ability to adopt to the changing node behaviours. In my approach to replication, I have used a parameter adaption approach, where any changes in availability leads to an update of the modelling parameters.

The approaches do not present the replication cost. The replica approaches also do not evaluate the time and bandwidth required to transfer data and maintain the required avail-
ability. Therefore Storage available at each node needs to be taken into consideration when replicating data.
Chapter 8

Application of prediction models in computing domains

The main research work for the prediction model presented here has been in the financial sector [40][61]. Some inspiration has been taken from this field to use these ideas in computing domains for various applications. Some of the relevant work that has inspired me to use these prediction models are presented here.

8.1 Time series analysis

The design of a failure detector requires a statistical analysis of messages received within a time frame called timeout. This timeout ensures a termination property of the protocol. Modelling communication delay [59] uses time series to model the possible delay in receiving the acknowledgement by monitoring nodes in a distributed system. The timeout time varies based on the time when the message is expected. This requires a dynamic timeout prediction to predict the correct status of the message, which is based on the principle of a pull-based failure detector, where a message is sent after a regular interval, and expects a response in the given time. The round trip time is modelled as a time series based on the time at which the message is sent. The algorithm presented takes into consideration stable conditions where the message response is received in time, and unstable conditions where the message response is either delayed or lost.
Time series analysis is used to predict I/O request times [83]. A combination of ARIMA and the Markov model is used to model temporal and spatial behaviour. The temporal model determines when to fetch data and the spatial model determines what to fetch. This study has used ARIMA time series to model the patterns of the I/O times for applications that have high I/O requirements with limited storage devices. By representing each I/O time to a storage device as a time series, the ARIMA model predicts complex time patterns that exhibit stationary, non-stationary, and seasonal behaviour. Time series analysis requires a model created based on the existing data. The I/O prefeching requires online modelling of time series. The ARIMA model parameter was updated based on the online version of parameter estimation and differentiation to create a stationary series that is used for parameter estimation. The online model determines the I/O request inter-arrival times, and predicts the time at which to prefetch blocks of data in order to overcome I/O latency.

Time series analysis has been used in predicting the access patterns of files [45]. Given a VoD application for a P2P network, an ARIMA model was used to predict the possible usage of files that provide scheduling based on the request and popularity of the files. This prediction helps with sharing the document in the network by adjusting the number of replicas based on the capacity of the peers. Time series analysis improves the load balancing between the peers. The ARIMA model predicts the popularity of data for the next 60 min. This probability is combined with the capacity and bandwidth constraints of the peers to adjust the number of replicas.

### 8.2 Monte Carlo simulation

Agent-based systems [21] have also used Monte Carlo simulation for evaluation of the reliability of mobile agents. A reliable random walk routing strategy has been proposed, which uses a reliable path for the mobile agents communication. In the distributed system the agents are given a task to travel as tokens, and these agents travel between nodes and traverse a random path within the timeout time $t$. The mobile agent should complete the random walk within the given timeout time. The reliability of the routing task is defined by the probability that the agent completes the task. The reliability of the system $S$ is a probability that the mobile agents complete the tasks during the given time period. Based on the given probability distribution of success, a Monte Carlo simulation is performed using the uniform distribution for each random walk. Using the result from the Monte Carlo simulation, the reliability state variable is evaluated. If the state variable is 1, then all of the mobile agents are operational. If the state
variable is 0, then all of the agent failed the tasks. A value between 0 and 1 indicates that some
of the agents are operational.

Monte Carlo simulation has been used extensively to monitor the reliability of networks
[14][35]. These studies inspired me to utilize a Monte Carlo simulation in a P2P system
reliability analysis. The purpose of using a Monte Carlo simulation was to more accurately
simulate the dynamic environment using the component failure distributions, which addresses
link failure analysis using a Monte Carlo simulation. Based on varying network connections
resources, simulation was performed to determined the overall reliability of the system.

8.3 Summary

The prediction models and their application presented here are most relevant for reliability
analyses. My replication requirements are very similar. The replication approach presented
in this thesis was hypothesized based on these applications and reliability analysis theory.
The time series analysis and Monte Carlo simulation provide a feedback analysis approach
to frequently analyse the changing node behaviour. Through my analysis, I have found that
one of the draw backs of the Monte Carlo simulation is that it requires a higher number of
historical records to accurately predict future behaviour. Because the model depends on the
generation of a probability distribution of the node online and offline times for analysis, a long
historical record is required to generate the probability distributions.
Part III

Design and analysis of availability based replication
In this section I present the overall design of the replication system. Figure 8.1 shows the overall system flow. The overall design flow incorporates the prediction models with the availability analysis to filter out the nodes for replication.

![Diagram](image)

**Figure 8.1:** The overall process of data replication.

The replica calculation starts with the gathering of lifetime information. Both prediction approaches presented in Chapter 5 require some lifetime history of the node before parameters of the model can be generated. Each node in the network generates a historical record of the time duration it has been online and offline. The pre-predication modelling step involves analysis of the historical data. The time series analysis approach performs some parameter determining steps to identify a correct model. When using a Monte Carlo simulation, similar steps are needed to determine the probability distribution of the data. Each node predicts future values after analysis of the historical online and offline data. The times series analysis approach uses historical data as time series and determines the future online and offline
times. In a Monte Carlo simulation, simulations are run to predict future values based on the distribution of the data resulting from the fitting of data to a distribution. The replica generating node uses the predicted values to evaluate required availability of data by evaluating possible lifetime overlaps and applying network constraints (available bandwidth and storage requirements). When a node meets the network constraints requirements, it is selected for data replication. As part of the replica calculation, popularity of the data is also considered. This is important to avoid overhead of replicating data that is not of high demand. A running example is demonstrated with the data analysis and behaviour modelling phase. Using the model generated, the final step of the model is to predict the future online and offline times of the node.

**Assumptions for Algorithm Design**

The following assumptions were taken into consideration during the design of the algorithm:

1. Peer Trust worthiness: There is no reputation system in the replication algorithm. The data gathering and sharing between nodes assumes honestly shared availability information.

2. Peer capacity: Peers in the network have varying network bandwidth and storage capacity. The replication algorithm takes into consideration the varying bandwidth and storage capacity.

3. Peer behaviour bootstrapping: The time series analysis and Monte Carlo simulation requires modelling of online/offline time of the peers. Bootstrapping time is required for these analysis approaches in order to have sufficient data to analyze historical logs to make predictions.

4. Peer Behaviour adaption: The replication algorithm takes time to adapt to the rapid changes of the peer behaviour and update parameters of the availability analysis models.

5. Replication responsibility: The owner of the data is responsible for the replication of data and the maintenance of the required availability.
Chapter 9

Data gathering and data analysis

In this chapter I will analyze the process of data gathering and analysis of the data gathered. I will outline the requirements of historical data and modelling in this chapter as presented in[56][57].

9.1 Data gathering

Data gathering is the first required step for the overall design of behavioural analysis. Sufficient historical data are required to develop a behavioural model. There data should be sufficient for determining the trends in a series. The sample size is sufficient when the normalized mean square error is between 0-1. The suggested sample size for an ARIMA time series analysis by Box Jenkin is 50 observations. The two behavioural analysis, approaches require varied historical data to predict acceptable level of node behaviour. For time series analysis the data set size depends on the randomness of the data. The higher the level of randomness, the larger the size of data set required for time series analysis. Data that have less variation requires a smaller sample size. Through the analysis it was found that after 4 days of data, the ARIMA time series model is able predict behaviour with the required level of normalized mean square error. The time required to achieve a similar level of accuracy for a Monte Carlo simulation is one week.

Each node is responsible for recording the time for which it has remained connected to the network and the time it has remained disconnected. At startup, when no historical information of the availability is available, a node randomly selects 10 nodes from its neighbour table for
replication. From this initial setup, nodes share behaviour prediction information with the nodes interested in replication of data.

Through the analysis of the historical data it can be determined if a node exhibits any seasonal/weekly similar behavior. Time series analysis is able adjust parameters of autoregressive and moving average components based on the model established from the analysis of ACF and PACF. This adjustment helps to accurately predict future values. Any node showing seasonal behavior requires the historical data to be differentiated to make the series stationary.

To run the example for analysis, I consider a node starting as a non-stationary process. The following are the consecutive times in minutes when the node remains online:

\[ 40, 60, 105, 100, 75, 90, 105, 65, 105, 65, 125, 190, 150 \]

Similarly a historical record of the nodes offline time durations is retrieved from the nodes joining and departing time from the network. This historical record is kept by each node for analyzing the behavioural changes.

### 9.2 Data analysis

This section analyses how the nodes in the network perform the pre-modelling and pre-prediction steps. The nodes analyze their historical online and offline times. Due to the dynamic node behaviour in the network, an iterative/recursive data analysis approach is required, which updates its parameters with the changing behaviour of the nodes.

#### 9.2.1 Time series modelling

As presented in Section 5.1, time series analysis predicts future behaviour based on the assumption that an entity to be analysed will mimic past behaviour. Based on this assumption I consider modelling the nodes in the P2P network as individual entities that exhibit independent behaviour, with some correlation to their previous up times and down times. As discussed before a node can follow a seasonal, stationary or non-stationary behaviour. Runtime data analysis was required for the P2P network data. Based on a previous study [83], I have utilized the described approach for runtime data analysis.
I will applying an ARIMA time series model and follow the steps identified in the previous section. The historical data gathering is a continuous process, as part of the replication algorithm. This data gathering process helps to update the model based on the latest values and minimize any error due to the change in the node behaviour. I use an autocorrelation function to automatically identify an ARIMA model. A Least square algorithm is used to estimate the parameters of the model. Statistical tests can be used to test the model, such as Chi-squared goodness of fit test, but with the extra overhead of this calculation, I instead chose using prediction errors based on the difference between the predicted and actual values. I used the normalized mean square error (NMSE) for the prediction evaluation. The normalized mean square error is given by:

\[
NMSE = \frac{1}{\sigma^2} \frac{1}{M} \sum_{t=1}^{M} (X_t - \hat{X}_t)^2 
\]

where \(\sigma^2\) is the variance of the time series over a given time duration, \(M\) is the total number of predicted values, and \(X_t\) and \(\hat{X}_t\) are the observed and predicted values respectively. In this case, the time series is the time during which a node remains connected to the network, and the second time series is the time the node remained disconnected from the network. For a perfect predictor, the NMSE has a value of 0, and an NMSE value greater than 1 is considered to be a bad result. For a trivial predictor, which statistically predicts the mean of the time series, has \(NMSE = 1\).

As part of the design of the time series model for node behaviour prediction we will go through the steps with some examples from live traces used in the analysis of data node availability in the network. Figure 9.1, shows the steps of the ARIMA model needed to evaluate the future behaviour. It starts with the model identification and performs a series differentiation if required. A parameter estimation is performed for the moving average and auto-regressive part of ARIMA model, followed by the prediction of the future values. In this chapter I will only run time series modelling until parameter identification. The future value predication phase as shown in Figure 8.1 and elaborated in Chapter 10. Consider a system where nodes leave and rejoin the network. As shown in Figure 9.2, let a node joining the network at time \(t\), which stays connected until time \(t + \delta t\). The time duration for which the node remains connected to the network is \(c(t)\). Given this time duration, the set \(C(t)\) of connected times, is a historic details for node connectivity in the network. Similarly, to model the time a node remains disconnected from the network, let \(d(t)\) be the time where the node remains disconnected from the network. This is calculated using the time when the node left the network \(t + \delta t\), and the
time when the node rejoins the network at $t_1 + \delta t$. Let the set of node disconnection durations be $D(t)$.

**Figure 9.1:** ARIMA model process.

**Figure 9.2:** Node lifetime.
Formally $C(t)$ can be defined as:

$$C(t) = \bigcup_{t=1}^{n} c(t)$$  \hspace{1cm} (9.3)

and $D(t)$ is given as

$$D(t) = \bigcup_{t=1}^{n} d(t)$$  \hspace{1cm} (9.4)

Where $i = 1, 2, ..., n$ is time instance index.

$$c(t_i) = t_{i+1} - t_i$$

Where $t_{i+1}$ is the node’s departure time from the network, and $t_i$ is the time when the node joins the network.

$$d(t_i) = t_{i+1} - t(i)$$

In this case $t_{i+1}$ is the node’s joining time to the network, and $t_i$ is the time the node departs from the network.

Each set serves as a series of data for time series analysis.

**Time series model identification**

As discussed in a previous section, time series analysis starts with model identification. Parameter identification is needed of auto-regression $p$, moving average $q$, and lag time $d$ for the time series ARIMA model. Autocorrelation function (ACF) and partial autocorrelation functions (PACF) are used to determine parameter values of the ARIMA model. ACF and PACF calculate any correlation between observation for a given time series.

The autocorrelation function measures linear dependencies between observation pairs, given $y(t)$ and $y(t+k)$, separated by time period $k=1, 2, ...$. Autocorrelation at lag $k$, for random variable $Y_t$ and $Y_{t+k}$ is defined as a correlation between these two variables, which is known as expectation of the random variables deviations from the mean, normalized by the variance and given as:
Data gathering and data analysis

\[ ACF(k) = \frac{E(Y_t - \mu)(Y_{t+k} - \mu)}{\sigma^2} \] (9.5)

This autocorrelation function is a value generated for a stationary process, where the mean and variance remain constant.

Autocorrelation function does not take into account an observation in between the observed time period. For that purpose, partial autocorrelation is used, which considers observations between the lag as well. To determine the effect of \( y(t) \) and all of its intermediate values on \( y(t+k) \), let this effect be denoted as \( A_{kk} \). In the equation coefficient, \( A_{kk} \) is the contribution of \( y(t) \) to the total effect, given as:

\[
y(t + k) = A_{k1}y(t + k - 1) + A_{k2}y(t + k - 2) + ... + A_{kk-1}y(t + 1) + A_{kk}y(t) + e(t) .
\] (9.6)

Regression analysis of coefficient \( A_{kk} \), can be very computational extensive, due to an evaluation for every lag until \( k \). Durbin [24] proposed a recursive method by using the autocorrelation and using the regression coefficient from the previous lag.

In the following section I will discuss the expected behaviour of the patterns of data based on ACF and PACF, in order to determine the appropriate model. These patterns are for stationary series, but my node is modelled as a non-stationary process, which is converted to stationary using the Box and Jenkin approach to use differencing.

**Shapes of correlation functions**

Patterns of the ACF and PACF analysis help the ARIMA model determine the parameters \( p \) and \( q \) of the autoregressive AR\( (p) \), moving average MA\( (q) \) or ARMA\( (p, q) \) process. Because the pattern exhibited by the two functions has sinusoidal or exponential decay, or has spikes with an abrupt cutoff. Box and Jenkin found that the most appropriate values for \( p \) and \( q \) are usually 0, 1, and 2. Table 9.1 summarizes the characteristics of the plots.
Table 9.1: ACF and PACF patterns

<table>
<thead>
<tr>
<th>Process</th>
<th>ACF</th>
<th>PACF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(p)</td>
<td>exponential or sinusoidal decay to zero</td>
<td>cuts off to zero after lag p</td>
</tr>
<tr>
<td>MA(q)</td>
<td>cuts off to zero after lag q</td>
<td>exponential or sinusoidal decay to zero</td>
</tr>
<tr>
<td>ARMA(p,q)</td>
<td>exponential or sinusoidal decay to zero</td>
<td>exponential or sinusoidal decay to zero</td>
</tr>
</tbody>
</table>

Model identification methodology

The ARIMA model follows four steps to identify the appropriate model:

- ACF of the series determines if it is a non-stationary series. A slow decay shows that the series requires differencing. Continuous differencing is performed until the series decays quickly. The number of differencing $d$ is usually identified to be no greater than 2.

- The model is then determined based on spikes in the PACF or decay in the ACF plots. Given the characteristics of the patterns, the value of $p$ and $q$ is selected.

I will run through a model identification methodology using live traces of a node inter-arrival time.

Time series differencing

The ARIMA model requires a non-stationary series to be converted to a stationary series for parameter estimation of autoregressive and moving average equations. There are several techniques used to transform non-stationary series to a stationary series. Differencing is the most common approach used for it. The aim of differencing is to stabilize the mean of the non-stationary series. In differencing, consecutive values of the series are subtracted to generate a transformed series. The following is an example of one level differencing.

In some cases complex transformation will be required to stabilize a series mean. More than one differencing will be required in that case. This would handle the seasonal behaviour of the series as well. Where an interval is determined, in which the cyclic behaviour is exhibited by the series and the starting of the next cycle value is subtracted from the starting of the previous cycle value, the corresponding values within the cycle are subtracted with their corresponding values in the previous cycle.
As part of the overall architecture, after the forecasting on n-ahead values has been performed, which is accomplished through integration in order to recover the values for the non-stationary series.

**An example of running a model structure identification**

As part of the first phase of the ARIMA model, the values of p, d, and q have to be determined. These values will be calculated by determining the correlation between observations in the given time series. Given the nodes online times, I will determine if the node is stationary. An autocorrelation function of the series will tell if the series is stationary. Based on the online time analysis, it can be seen that the mean and variance of the node changes over time as shown in Figure 9.3a. The autocorrelation plot in Figure 9.3b shows that there is slow decay to 0. This suggests that series is non-stationary. The node does not exhibit any seasonal behaviour. In seasonal behaviour, the data in a time series graph would have shown repetitive patterns.

![Node Online Time Series](image-a.png)

**Figure 9.3:** Non-Differentiated Node Online Data Time Series.

![Node Online Time Series ACF](image-b.png)

---

<table>
<thead>
<tr>
<th>40</th>
<th>60</th>
<th>105</th>
<th>100</th>
<th>75</th>
<th>90</th>
<th>105</th>
<th>65</th>
<th>105</th>
<th>65</th>
<th>125</th>
<th>190</th>
<th>150</th>
</tr>
</thead>
</table>

![Node online time series-1st differentiation procedure](image-c.png)

**Figure 9.4:** Node online time series-1st differentiation procedure.
Convert to stationary process

To convert a non-stationary series to a stationary series, differentiation is performed. The differen-
tiation is defined as the difference between the pair of values. The differencing procedure is shown in Figure 9.4. Each time series value is subtracted from the next value in the series. Figure 9.5 shows that the mean and variance do not change over time after differentiating the series once.

Figure 9.5: Differentiated Node Online Time Series.

Figure 9.6: Differentiated Node Online Data Time Series.
Model identification for stationary series

After converting a non-stationary process to a stationary process, the model identification process can be applied to the ARMA series. To determine the parameters $p$ and $q$ of the ARMA model ACF and PACF are applied. The parameter values are determined based on the parameter identification rules defined in the previous chapter. As shown in Figure 9.6, the ACF and PACF exponentially decay to zero, with a spike at lag 1. This suggest an ARMA(1,1) model. The next step is to determine the parameters for ARMA(1,1).

Parameter estimation

In this section I will present the parameter estimation technique used for the ARMA(p,q) model. I also discuss the dependencies between observations. The technique used for parameter estimation is the least squares algorithm. I will use the same ARMA(p,d) equation given in Equation 5.3. The model consists of two parts, autoregressive AR(p) and moving average MA(q). Both require coefficient determination. The AR(p) requires coefficient determination for $a_0, a_1 \ldots a_p$, and MA(q) has coefficient determination for $b_1, b_2, \ldots b_q$. The ARMA model also includes noise term $e(t)$ of AR(p), which is the current noise term and previous noise terms $e(t-1), e(t-2) \ldots e(t-q)$ of MA(q). These noise terms are considered to be Gaussian, which is based on the central limit theorem. Because as the number of samples is sufficiently large, the sample mean will be approximately normal. The ARMA model helps to determine how far back the values should be considered and how correlated they are with the current value. For example, considering an ARMA(2, 1) model of a stationary series, the online times of a node will be given as:

$$y(t) = 0.9y(t - 1) - 0.2y(t - 2) + e(t) + 0.03e(t - 1)$$  \hspace{1cm} (9.7)

Using 2 terms from the AR(p), online times, $y(t-1)$ and $y(t-2)$ are highly correlated with the current online time. The coefficient of these values quantifies the correlation. The value at t-1 90% correlated with the value at time t and is 20% negatively correlated with the value at t-2. Earlier values are not considered in this model. The noise coefficient for the last time t-1 has a 3% positive influence on the current node online time. The current noise coefficient for $e(t)$ is considered to be 0. In this example the mean is considered to be 0. I will continue to look into how these coefficients can be determined. The value of p and q of the autoregressive and moving average model have been determined based on the details given in section 9.2.1.
Least square parameter estimation is the most commonly used parameter estimation approach for an autoregressive model. The estimation process for AR parameters follows these steps:

I Calculate the priori prediction error $e(t)$ at $p$ steps $t$, the difference of the observed value and predicted values is calculated. This is the prediction error known as the noise term:

$$e(t) = y(t) - a_0 - a_1 y(t - 1) - a_2 y(t - 2) - ... - a_p y(t - p)$$  \hspace{1cm} (9.8)

II Calculate the prediction error for $p+1$: this results in a system of equations with $p+1$ unknown parameter ($a_0, a_1, a_2 ... a_p$), to be estimated:

$$\Delta(t) = \varphi(t) - \psi(t - 1)\gamma(t - 1)$$ \hspace{1cm} (9.9)

- $\varphi(t)$ is a vector with current observation $y(t)$ and previous $p$ observations.
- $\psi(t - 1)$ is a previous observation matrix. Each row consists of a constant 1 associated with $a_0$, and previous $p$ observations, $y(t-1), y(t-2),...,y(t-p)$.
- $\gamma(t - 1)$ is a parameter vector for all the estimated parameters $a_0, a_1, ... a_p$.
- $\Delta(t)$ is the prediction error vector, which is the difference between the current observation vector $\varphi(t)$ and $\psi(t - 1)\gamma(t - 1)$

III Least Square parameter estimate: To calculate the parameter estimate, minimize the sum of the square of the prediction error vector $\Delta(t)$ based on the parameters $\gamma(t - 1)$. Minimizing the squared error given in Equation 9.9, the following equation is obtained:

$$\min_{\gamma} \sum_{i=t-p+1}^{t} [e(i)]^2 = \min_{\gamma} \Delta'(t)X\Delta(t)$$ \hspace{1cm} (9.10)

$$= \min_{\gamma} [\varphi(t) - \psi(t - 1)\gamma(t - 1)]' \times [\varphi(t) - \psi(t - 1)\gamma(t - 1)]$$ \hspace{1cm} (9.11)

Partial differentiation of this equation is then performed with respect to each parameter of $\psi$, and each partial derivative is set to zero. The equation is solved and the resulting system of equations for unknown parameters yields:

$$\gamma(t - 1) = \psi'(t - 1)\psi(t - 1)^{-1} \times \psi'(t - 1)\varphi(t)$$ \hspace{1cm} (9.12)
Next, the parameters [6] for MA(q) are determined. There parameters are part of the noise component e(t-1), e(t-2), etc of the ARMA model. This noise represents the overall previous behaviour of the data. Before the estimation can begin, posterior error has to be determined. This is the error for the current value $y(t)$ and previous predicted values.

$$e(t) = y(t) - a_0 - a_1 y(t-1) - ... - q_p y(t-p) - b_1 e(t-1) ... - b_q e(t-q)$$

past predicted values can be written as $\phi'(t-1)\gamma(t)$, and the final equation for posterior error is given as:

$$e(t) = y(t) - \phi'(t-1)\gamma(t) \quad (9.13)$$

where $\phi'(t-1)$ is the observation vector at time $t-1$. The vector contains past $p$ values with past $q$ noise estimates, in addition to a constant 1 representing the parameter $a_0$.

$$\phi'(t-1) = [1 \ y(t-1) \ ... \ y(t-p) \ e(t-1) \ ... \ e(t-q)]$$

The following steps are used to determine these constants:

1. Initialization: Parameter vector is initialized to null, $\gamma(0) = 0$. Inverted covariance matrix $R^{-1}$, is set to an identity matrix. This matrix reflects how much the series values vary with respect to each other.

2. Estimate Parameters: parameters for $\psi(t)$ are determined using the priori prediction error and gain factor, which are the previous gains given as:

$$GainFactor = \frac{R^{-1}(t-2)\varphi(t-1)}{1 + \varphi^T(t-1)R^{-1}(t-2)\varphi(t-1)} \quad (9.14)$$

The denominator in the above equation represents the correction by which the previous gain has to be adjusted. As the algorithm converges, it approaches 1, thereby reducing the gap between the previous and current gains. Parameters are estimated by recursively using the previous parameters, priori prediction error, and gain factor:

$$\phi(t) = \phi(t-1) + (GainFactor * prioripredictionerror) \quad (9.15)$$
3. Post estimation phase: the posteriori prediction error is calculated using Equation 9.13. The observation vector \( \phi(t) \), is updated using the autoregressive past and current observation values. The error calculated is also shifted to the moving average part. The inverted covariance vector is updated from its previous values \( R^{-1}(t-2) \) to include the gain factor given as:

\[
R^{-1}(t-1) = R^{-1}(t-2) - \text{Gain Factor} \cdot \varphi^T(t-1)R^{-1}(t-2)
\] (9.16)

**Parameter identification for stationary series**

Consider the following example for ARMA(2, 1):

\[
y(t) = 0.8y(t - 1) - 0.1y(t - 2) + e(t) + 0.03e(t - 1)
\] (9.17)

This equation shows how much historical data would be needed to evaluate the current observation. This equation contains two autoregressive terms and one moving average. The AR part of the equation considers the recent two online times. It shows the previous online time is 80% positively correlated to the recent observation, and the one before that is 10% negatively correlated to the most recent observation. The equation does not consider any earlier time events, and discards them. The noise term associated with the moving average has 3% influence on the current online time. The noise term \( e(t) \) has an expected value of zero based on the Gaussian distribution assumption. Using the same example to show the effect of the \( a_0 \) on the ARMA equation, I introduce a non-zero mean \( \mu \). In the previous example mean was set to zero. In this case \( \mu = 5 \), and the equation with the non-zero mean is reevaluated:

\[
y(t) - \mu = 0.8[y(t - 1) - \mu] - 0.1[y(t - 2) - \mu] + e(t) + 0.03e(t - 1)
\]
\[
y(t) = \mu(1 - 0.8 + 0.1) + 0.8y(t - 1) - 0.1y(t - 2) + e(t) + 0.03e(t - 1)
\]
\[
y(t) = 1.5 + 0.8y(t - 1) - 0.1y(t - 2) + e(t) + 0.03e(t - 1)
\]

The constant term 1.5 summarizes the effect of correlation on the mean. It is the value over which the series fluctuates.

With the parameter of ARMA determined, I will continue with the calculation for \( ARMA(1, 1) \) coefficients using the stationary series:

\[
20, 45, -5, -25, 15, 15, -40, 40 - 40, 60, 65, -40
\]
With this series I will use the extended least square (which includes a parameter calculation for MA(q) to determine the parameters for the ARMA model. The parameters will be determined in three phases:

- Phase I: In this phase the covariance matrix is determined using the initial values of the observation matrix \( \varphi(t - 1) \). Only one autoregressive part the matrix consists of the first online time. The moving average part, representing the noise terms, is initialized with zero. The first entry is the constant 1 associated with \( a_0 \). The observation matrix is initialized as:

\[
\varphi(t - 1) = \begin{pmatrix}
1 \\
20 \\
0
\end{pmatrix}
\]

The inverted covariance identity matrix is initialized with very large values. The size of the matrix is determined based on the number of terms to be used and \( a_0 \). In my example it will be a symmetric matrix with a dimension of 3. The matrix is given as:

\[
\begin{pmatrix}
10^6 & 0 & 0 \\
0 & 10^6 & 0 \\
0 & 0 & 10^6
\end{pmatrix}
\]

- Phase II: This phase is the first parameter estimation phase where the second online time is used. Priori prediction error is calculated, which is the difference between the new online time 45 and the predicted value. The predicted value is the product of the transposed observation vector and the previously estimated vector \( \psi(t - 1) \) values, which are zero. The priori prediction error is given as:

\[
Priori\ prediction\ error = 45 - \begin{pmatrix} 1 & 20 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = 45
\]

Next, the gain factor is calculated using the observation vector \( \varphi(t - 1) \) and the inverted covariance matrix \( R^{-1}(t - 2) \), as given in Equation 9.14.
Data gathering and data analysis

\[ \text{Gain Factor} = \frac{\begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix} \begin{pmatrix} 1 \\ 20 \\ 0 \end{pmatrix}}{1 + \begin{pmatrix} 1 & 20 & 0 \end{pmatrix} \begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix} \begin{pmatrix} 1 \\ 20 \\ 0 \end{pmatrix}} = \begin{pmatrix} 0.0025 \\ 0.0499 \\ 0 \end{pmatrix} \]

Finally the parameters are estimated for the current observation \( \phi(t) \) using Equation 9.15:

\[ \phi(t) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.0025 \\ 0.0499 \\ 0 \end{pmatrix} \times 45 = \begin{pmatrix} 0.1122 \\ 2.2444 \\ 0 \end{pmatrix} \]

- Phase III: The final phase of the parameter estimation is to prepare for the next parameter estimation.

First the posterior prediction error is determined. The posterior prediction error is the difference between the current online time and the predicted value, which a product of transposing the observation vector \( \phi(t-1) \) and the new parameter vector \( \phi(t) \).

\[ \text{Posteriori Prediction error} = 45 - \begin{pmatrix} 1 & 20 & 0 \end{pmatrix} \begin{pmatrix} 0.1122 \\ 2.2444 \\ 0 \end{pmatrix} = 1.1 \times 10^{-7} \]

The inverted covariance matrix is updated on every iteration as shown in Equation 9.16. The previous covariance matrix and the observation vector are used to re-evaluate the matrix. A recursive least squares equation is used to determine this in order to avoid
matrix inversion:

\[
\begin{pmatrix}
997500 & -49900 & 0 \\
-49900 & 2500 & 0 \\
0 & 0 & 10^6
\end{pmatrix}
= \begin{pmatrix}
10^6 & 0 & 0 \\
0 & 10^6 & 0 \\
0 & 0 & 10^6
\end{pmatrix}
- \begin{pmatrix}
0.0025 \\
0.0499 \\
0
\end{pmatrix}
\begin{pmatrix}
1 & 20 & 0 \\
0 & 10^6 & 0 \\
0 & 0 & 10^6
\end{pmatrix}
\]

The final step is to use the current online time in the autoregressive part of the observation vector \( \varphi(t - 1) \). The posterior prediction error is moved to the moving average part for the next estimation:

\[
\varphi(t - 1) = \begin{pmatrix}
1 \\
45 \\
1.1 \times 10^{-7}
\end{pmatrix}
\]

For the rest of the series, phase 2 and 3 are repeated by recursively updating the parameters and using the covariance matrix, gain factors and previously calculated parameters.

### 9.2.2 Monte Carlo modelling

This section elaborates a Monte Carlo simulation model for a P2P network. Each node \( N_i \) in the network exhibits its own behaviour. The time that the node remains connected to and disconnected from the network follows a probability distribution. After evaluating this distribution, the nodes can determine the future probabilities of being online or offline for a given time period. I will use the Monte Carlo simulation for that purpose, as it is an iterative process, where a node based on its previous probability evaluates the future behaviour based on random generated value the.

Let \( G \) be the network consisting of independent nodes \( V \). Consider \( G_s \) as a subset of \( G \) representing a set of nodes sharing a set of common data. Each node \( V_i \) is represented as a random variable \( X_i \). In this network, each node \( V_i \) is not represented as an independently and identically distributed (i.i.d) random variable. Each node as a random variable can have its own probability distribution for the time that the node remains connected to the network or is disconnected. Based on the availability theory, the data are considered available if at least one of the nodes holding the data remains connected to the network. The probability that a data remains available in the network for time \( \delta t \) is the reliability of the data. I will estimate the
reliability of the data by predicting the reliability of the nodes in the network. The reliability of $G_s$, is the probability that $G_s$ has some node connected in to it a give period of time $t$. The reliability is considered to be 1 if all ofthe nodes of the network are available. Reliability of the subnetwork $G_s$ is zero when no node is available and connected to the graph, Based on the state variable, which will be defined in the reliability modelling in chapter 11. The state function of sub-network $G_s$, $\Phi(S)$ is the mean of the random state variables of nodes, where $\Phi(S)$ is 1 if all of the nodes are online for the sub-network $G_s$, or 0 if all of the nodes are down. This value will be between 0 and 1 if only a portion of the nodes are available.

The reliability definition for the sub-network is given by the following equation:

$$R_s = P(G_s \text{ is available}|Y)$$

(9.18)

This is the conditional probability that a sub-network $G_s$ is connected, given $Y$ as the random state of the nodes in the network. To optimizes this for a replica set of nodes, I will use a Monte Carlo simulation to simulate node behaviour. Based on the previous online and offline times of the nodes, I determine the distribution that these follow. Parameters are determined for the fitted distribution.

**Monte Carlo data analysis running example**

The first step in the Monte Carlo analysis is to fit the distribution of the online and offline times. Through analysis of the node online and offline times as shown in Figure 9.7, it was determined that they exhibit behaviour that follows Weibull and Lognormal distributions, respectively.
Figure 9.7: Distribution fitting for Node online and offline time.
Chapter 10

Behaviour prediction

In this section I present the prediction evaluation using the time series and Monte Carlo simulation. The prediction evaluation step from Figure 8.1, uses the parameter estimation and data distribution fitting from the data analysis step. The prediction evaluation is done for the next 1 week, where each time unit is for 6 hr, the x-axis in the figure 10.1 and figure 10.2 is scaled to present 6 hr duration per unit, and 5 in the x-axis is 30 hrs.

10.1 Time series modelling of N-step prediction

The main objective of estimation and differentiation of the time series of node online and offline times in Section 9.2.1 was to predict future values based on a previous analysis. For a one step ahead forecast, starting at time $t$, the next observation $t+1$ is conditioned on all previous time series values up to time $t$, based on the ARMA model. The ARMA model for online time can be given as:

$$y(t+1) = a_0 + a_1 y(t) + \ldots + a_p y(t - p + 1) + e(t+1) + b_1 e(t) + \ldots + b_q e(t - q + 1)$$

(10.1)

All of the quantities in the model, except the noise $e(t+1)$, are known at time $t$. Gaussian noise is assumed to be normally distributed, and $N(0, \sigma^2)$, $e(t+1)$ can be substituted with the expected value, which is zero at time $t$. One step ahead prediction is a linear combination of past online times and past noise terms weighted by estimated parameters $a_i$ and $b_i$. 
Based on one-step ahead prediction, future values can be predicted by incrementing the time steps to n time steps. From the first Equation 10.1, n-ahead values can be predicted as follows:

\[ y(t + 1) = a_0 + a_1 y(t) + \ldots + a_p y(t - p + 1) + 0 + b_1 e(t) + \ldots + b_q e(t - q + 1) \]
\[ y(t + 2) = a_0 + a_1 y(t + 1) + \ldots + a_p y(t - p + 2) + 0 + b_1 e(t) + \ldots + b_q e(t - q + 2) \]
\[ y(t + 3) = a_0 + a_1 y(t + 2) + a_2 y(t + 1) + \ldots + a_p y(t - p + 3) + 0 + b_1 e(t) + \ldots + b_q e(t - q + 2) \]
\[ \ldots \]
\[ y(t + n) = a_0 + a_1 y(t + n) + \ldots + a_p y(t - p + n) + 0 + b_1 e(t) + \ldots + b_q e(t - q + n) \]

To forecast the autoregressive part of the equation, previously forecasted values \( y(t+j-1) \) and \( y(t+j-2) \) are used as expected values of future online/offline times. The moving average part of the equation can be computed by using future Gaussian noise terms \( e(t+j), e(t+j+1) \ldots \) replaced with their expected mean of zero.

When using non-stationary online/offline time series, the predicted values have to be transformed back to the real value, similar to how a series had to be converted to a stationary series using differentiation.

### 10.1.1 Forecasting future online time with an ARIMA model

As part of forecasting the future values, the differencing performed to convert to stationary series requires the that predicted value for the stationary series must be re-integrated back to the non-stationary series. In this example, the predicted value was 4.2277. Adding this value to the previous value gives the forecasted value of 154.2277. A complete analysis was performed for this node to predict future values and then compared with the rest of the node online times. The normalized mean square error for this was 0.14. The figure 10.1, shows the online time prediction using the ARIMA model compared to the actual values received from the traces. It can be seen that the time series analysis is able to predict the future trends with only slight variation from the actual values. In this example it can be seen with the parameter...
estimated for the autoregressive part resulted in the predicted values to be slightly higher than the actual values for most of the observations. Lowering the estimated values increased the normalized mean square error.

Figure 10.1: ARIMA node online time forecast. X-axis is time in hrs calculated with per unit time of 6 hrs.

10.2 Monte Carlo modelling future prediction

The future prediction using a Monte Carlo simulation approach uses the distribution fitting and parameter estimation from Section 9.2.2. Each future prediction runs k times to generate random samples based on the distribution fitted:

\[ Y = f((X_1, X_2, ...X_n)_k), k = 1, 2, ..., N \]  

I am attempting to determine the most likely value for the node online or offline time for or any given Y, thereby obtaining a histogram for the number of times the particular number had been returned. Let \( R_l \), be the lowest reliability of the node, and \( R_h \) be the highest reliability of the node. The simulation run tells us the probability for the nodes to remain online or offline for a particular time. Let \( p_i \) be the threshold probability expected. Any number having an equal or higher value than \( p_i \) is considered to be the expected behaviour of the node. Therefore evaluating the mean for \( Y \), \( \mu_y = \frac{1}{n}(Y_1 + Y_2 + ..Y_N) \), which is the estimator of \( E(Y) \). The
variance of the estimator is:

\[ Var(Y) = \frac{Var(Y)}{N} \]  

(10.3)

The variance of the estimator determines the accuracy of the estimator. The lower the variance, the higher the accuracy of the estimator. Based on the mean value, this is considered the online or offline time prediction for the node. The simulation continues for the next \( t \) times, to determine the possible behaviour for the next \( t \) of the node.

### 10.2.1 Forecasting future online time with a Monte Carlo simulation

In this example I will forecast the next values for the same series used for prediction in the time series analysis. Using the parameter determined after fitting the distribution, random samples are generated. After running 10000 random generations using the fitted parameters, the forecasted value was 182.3825, and the logged online time of the node was 175. The normalized mean square error value was 0.60, which is higher than the time series analysis. The error comparison with actual values is shown in Figure 10.2. Although there appears to be a higher difference in values from the actual values, this approach is still able to capture the future trends as well.

![Monte Carlo node online time forecast. X-axis is time in hrs calculated with per unit time of 6 hrs.](image-url)
Chapter 11

Modelling reliability

As part of the overall replication dataflow as shown in Figure 8.1, reliability modelling is required when calculating the availability of data. In this section I will model a P2P network based on the reliability analysis model. The intention of modelling of the system is to provide maximum reliability and availability. As described in [74], a distributed system never fails completely but instead continues to operate in a degraded fashion. Thus, this model should be able to handle the dynamics of the network, where each node can independently join and leave the network. As shown in [88], overall availability takes into consideration the down times of the nodes. MTTF is defined as the mean time a node is connected to the network. The Mean down time (MDT)/ Mean time to repair (MTTR) is the mean time a node remains disconnected from the network. A node that has a high MTTF does not guarantee high availability. When selecting nodes with high availability (MTTF), an intersection of the time online would occur, and this would not help the overall availability. Selecting a node that is available when another node is not available, increases the overall availability. In this model I make sure that the MTTR duration of a node is modelled and covered by another node for that duration.

A P2P system is considered to be a system composed of parallel components, where each component is a node. All nodes are considered equal and have no hierarchy. The system fails only if all the nodes depart. To create replicas, each node considers a subset of nodes that, when combined, represent a system itself. Node data are considered unavailable when all of the nodes in the subset fail. The nodes in the network have a local knowledge based on their routing table and replication list.
Table 11.1: Reliability system model notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Total nodes in a system</td>
</tr>
<tr>
<td>$A_{avg}$</td>
<td>Average availability of a node</td>
</tr>
<tr>
<td>MTTF</td>
<td>Mean time to failure, mean time when a node remains connected to the network</td>
</tr>
<tr>
<td>MDT/MTTR</td>
<td>Mean down time/Mean time to repair, mean time when node remains disconnected from the network</td>
</tr>
<tr>
<td>$O$</td>
<td>Object to be replicated</td>
</tr>
<tr>
<td>$\Phi(X)$</td>
<td>Structure function</td>
</tr>
<tr>
<td>$R$</td>
<td>Reliability of the system</td>
</tr>
<tr>
<td>$R_k(t)$</td>
<td>Reliability of the replica set</td>
</tr>
</tbody>
</table>

I consider replication to $k$ nodes, where $k$ varies based on the availability of the nodes in the network. Let $O$ be the object to be replicated on $k$ out of $N$ nodes. Even if the replicas are placed on highly available nodes, it does not guarantee prevention from simultaneous failure. When nodes share similar availability duration and have an intersecting time duration, simultaneous node failures would lead to an availability of 0, for the given time duration. Let $F(x)$ be the overall failure probability distribution of the system. Failure of each node $X_i(t)$ at time $t$ is independent of all other nodes. The structure function $\Phi(X)$, which defines the state of the overall system, can be:

$$
\Phi(X) = \begin{cases} 
1 & \text{at least one node connected to the network} \\
0 & \text{otherwise} 
\end{cases}
$$

The structure function for a system of an overall P2P network can be modelled as:

$$
\Phi(X) = 1 - (1 - X_1)(1 - X_2)\ldots(1 - X_N) = 1 - \prod_{i=0}^{N}(1 - X_i)
$$  \hspace{1cm} (11.1)

Using this overall structure function, system reliability is the expectation of the overall system structure function for $N$ nodes, and is defined as follows:
\[ R = E[\Phi(X(t))] \]
\[ = 1 - \prod_{i=1}^{N}(1 - E[X_i(t)]) \]

as

\[ E[X_i(t)] = R_i(t) \]
\[ R = 1 - \prod_{i=1}^{N}(1 - R_i(t)) \]
\[ for \ i = 1, 2, ..., N \]

Availability is not only the time when a node is up, but also considers the down time as well, as in [7]. Average availability is calculated as:

\[ A_{avg} = \frac{MTTF}{MTTF + MDT} \] (11.2)

Using these lifetime availabilities, analysis is performed to identify the nodes that are available while the data owner node is unavailable. The overlapping times between the owner node and candidate nodes are calculated based on the previous node behaviour.

My algorithm prefers nodes with high availability, but does not ignore low availability nodes. The operational time of the system with the combination of nodes with high and low availability could be much higher than using only high availability nodes.
Chapter 12

Algorithm

In this section I elaborate upon the replication algorithm[56][57], which utilizes the availability prediction, using the models given in Chapter 10 and reliability model defined in Chapter 11. As shown in [86], replication shows better data availability than erasure-code in an environment that has low node availability. Erasure-code has proven to be very costly in terms of maintaining the number of m-blocks of data, because nodes that can be used to gather blocks are not connected to the network due to low availability. Replication is a special case of erasure-code where the number of m-blocks are 1. Similar to an erasure-code, it is replicated to k-locations. I evaluated availability parameters based on the network probability distribution parameters taken from [65], [28], and [60]. From that evaluation, I varied the churn rate to measure my data availability factor.

The main objective of replication is to improve data availability and fault tolerance. A node replicates data to nodes that are highly available and complement each other with their availability, (i.e. when one node is unavailable the other node would be available). The replication algorithm follows the steps shown in Figure 12.1.

I present epidemic algorithms for replica placement based on two phases. As in the epidemic algorithms, information is disseminated from a node to multiple nodes in the same way information required to calculate the required availability is propagated between the nodes to the source node. The two phases are used to retrieve information for the availability calculation, and the second phase is to place the replicas. The two phases are shown in the Figure 12.1.
12.1 Replication cost model

Given the sorted list of node availability, the number of replicas to be created is based on the overall availability and maximum lifetime availability generated from the best set of nodes. The size of data to be replicated varies based on the transfer time that the two nodes have before one becomes unavailable. It has been observed that queries follow a Zipf distribution \( 1/i^\alpha \), where \( \alpha \) is between 0.7 and 1, based on the work in [79]. The probability that a request arrives for i’th most popular data of \( P(i) \), where \( i = 1, 2, ..., N \), and \( N \) is the total data available at a node.

\[
P(i) = \frac{\Omega}{i^\alpha} \quad (12.1)
\]

where

\[
\Omega = \left( \sum_{i=1}^{N} \frac{1}{i^\alpha} \right)^{-1} \quad (12.2)
\]

Let total data be \( I_N \) and size of each file be \( S_i \). The transfer time \( T \) between two nodes is the time between which both the nodes are connected to the network be \( T \) seconds. Given \( B \), the bandwidth of the node, and \( r \) the number of replicas, the total data that can be transferred during that time is \( (B/r) \times T \) bytes. This is the size constraint for the data to be transferred. Let \( Q\% \) be the requirement to satisfy the queries. The total data to be replicated calculated based on the percentage of queries answered with the given data. Let the total data \( I_t \) to replicate, have a percentage of queries satisfied greater or equal to the required query answering percentage \((Q)\%\). The total size of the data to be replicated is \( S_{It} \).
Algorithm 101

\[ S_{It} = \sum_{i=1}^{L_t} I_i \ast S_i \]  

(12.3)

Total size \( S_{It} \) should be less than the size of data, given the bandwidth and time when both the nodes are running \( T \).

\[ S_{It} < = (B) \times T \text{ bytes} \]  

(12.4)

If this constraint is not met, the search for a replica continues.

Given the availability requirement, let \( r \) be the total replicas, and therefore the total size of the replicated data \( R_{It} \) is:

\[ R_{It} = r \ast S_{It} \text{ bytes.} \]  

(12.5)

12.2 Replication algorithm

In the first step, the push phase of the network is used to place new nodes, and in the second step the pull phase occurs where the replica placed node requests the original data holder for the updates.

A node responsible for replication has push updates to replica nodes with \((O, O_v, t_e)\), where \( O \) is the updated data sent and \( O_v \) is the version of the data. Finally, \( t_e \) is the time when no node is online to maintain the replica. Every node performs the availability analysis and selects nodes upon receiving the update, based on \( t_e \), which is available during the time given in \( t_e \). When a node comes online again, nodes that have not received updates use the pull algorithm to become synchronized.

I consider the highly dynamic nature of the nodes with a mean online time between 20 and 40%. The owner node \( N_i \) initiates the replication algorithm by retrieving tuple < time since the last join, the next failures, and up times until time \( \tau \), based on the time series analysis forecast, average availability > from its neighbours using ping responses. Each candidate node applies the time series analysis or Monte Carlo simulation to forecast its next behaviour.
upon receiving a ping. The next up times and down times until time $\tau$ are predicted and sent in the response. Each node saves $p$ past failures and reconnection times for analysis.

The algorithm for the replica management is given in algorithm 1. The replication algorithm follows 3 steps: aggregation phase, analysis phase and replication phase. Lines 3-7 are the aggregation phase (Figure 12.2), lines 10-23 are the analysis phase (Figure 12.3) and lines 25-28 are the replication phase where data are replicated to the nodes selected for replication in the analysis phase (Figure 12.4). Instantiation of the algorithm Lines 1-2, starts with the initialization of replicas set with a random set of K nodes from the routing table and setting the required acceptable availability to 99.99 percent. The replication algorithm is initiated when the overall availability falls below the threshold $reqAvail$ or data popularity changes, using the $pingNode$ operation to retrieve updated data availability and data popularity, given in the conditional statement at line 4. When the conditions for updates have been met, the algorithm (lines 5-7) estimates the new data replication size $dataSize$ by calling $totalDataRep$ operation. It sets the initial availability pattern to be the source node life pattern and sorts the routing table based on node availability in descending order. Lines 10-23 contain the core algorithm where data availability is calculated and the replica set is selected. Line 12, the operation $generateLifePat$, is used to generate life patterns for the node. A Monte Carlo simulation or ARIMA model for the failures analysis are generated to calculate the availability and predict future failure. At line 12, the transfer time is calculated by the ANDing operation, which generates the transferring time for data between the nodes. The nodes not meeting the time constraint requirement of data transfer are discarded. The subset of nodes meeting the required availability is selected for replication. At line 14, the node performs the ORing operation of the lives, to determine the availability of the two nodes during time $\tau$ and totals the availability in order to calculate the current availability. The process continues until the required availability is not reached. The node transfers data to the node in the replica list from lines 25-28.

A source node required to replicate data requests availability information from the neighbours. Using the algorithm given in Algorithm 2. The $pingNode$ operation evaluates the overall popularity of data shown in line 10-13 and determines if a replication update is required based on the changed data popularity. The function $generateLifePat$ is used to determine the future node departure from the network time and node network joining time. From line 16-22, an online and offline prediction evaluation is performed and life patterns are generated. Starting at time $t = 0$, a time $t1$ is taken from the uptime given from the ping response until the first time to failure/departure $t + \delta t_f$. The simulation clock is now set to $t1$. The next $d1$ is given
<table>
<thead>
<tr>
<th>Algorithm Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Size of the replica set.</td>
</tr>
<tr>
<td>replicaList</td>
<td>Set of nodes where data has to be replicated</td>
</tr>
<tr>
<td>reqAvail</td>
<td>Required Availability, set at the start of the network.</td>
</tr>
<tr>
<td>avail(i)</td>
<td>life pattern of node i in the replica set. This availability contains the overall availability provided between the host and node from routing table.</td>
</tr>
<tr>
<td>Node</td>
<td>Node from the routing table.</td>
</tr>
<tr>
<td>Node.Life</td>
<td>Life patterns of node.</td>
</tr>
<tr>
<td>popularityUpdate</td>
<td>Flag to monitor if the data distribution popularity has been changed.</td>
</tr>
<tr>
<td>timeToDeparture</td>
<td>future time prediction of leaving the network.</td>
</tr>
<tr>
<td>timeToJoin</td>
<td>future time prediction of joining the network.</td>
</tr>
<tr>
<td>C</td>
<td>Sum of inverse of ranking of all content.</td>
</tr>
<tr>
<td>P(i)</td>
<td>Probability that a request arrives for i’th popular data</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>exponent for Zipf distribution</td>
</tr>
<tr>
<td>Query_Popularity</td>
<td>Required popularity of queries to resolved</td>
</tr>
</tbody>
</table>
Algorithm 1 Replica Maintenance Algorithm

1: replicaList = routingTable(K) initialize with K random nodes from routing table
2: reqAvail = 99.99
3: if sessionTime % τ == 0 then
4:   if pingNodes() < reqAvail or popularityUpdate then
5:     data = totalDataRep()
6:     dataSize = calculateDataSize(data)
7:     avail(0) = this.Life;
8:     sortRoutingTable(routingTable) \{ sort routing table based on the node availability \}
9:     i=1
10:   while i ≤ routingTable.size() & currentOverAllAvail < reqAvail do
11:     Node = routingTable(i)
12:     Node.Life = generateLifePat(Node)
13:     transferTime(i) = AND(Node.Life, this.Life)
14:     if transferTime(i)*BandWidth ≥ dataSize then
15:        avail(i+1) = OR(Node.Life, avail(0))
16:        currentOverAllAvail = SUM(avail)
17:        insert Node[i] to replicaList
18:        if currentOverAllAvail ≥ reqAvail then
19:           break;
20:   end if
21: end while
22: i++
23: end while
24: i=0
25: while i < replicaList.size do
26:   transferData(replicaList[i], data)
27:   i++
28: end while
29: end if
30: end if
from a ping response till the next time the node joins. The clock now moves to $t+\delta t_f + \delta t_r$. A similar process continues for $ti, di$ till a lifetime, reaches duration $\tau$. Using the generated lifetime the host node calculates the availability of each node. The lifetime patterns are generated from the predicted values, as shown at lines 19-20. The host node uses the lifetime patterns to determine the transfer time and overall availability.

Algorithm 2 Ping, data popularity and lifetime generation Algorithm

```plaintext
1: function pingNodes()
2:   i=0
3: while $i \leq$ routingTable.size() do
4:     ping(Node(i))
5:     Node.currentDataPopularity(i) = popularityList; receive data popularity probability list
6:     currentAvail=Node(i).currentAvail;
7:   i++
8: end while
9:
10: if fit_data(Node.currentDataPopularity) $\neq$ Node.InitPopularity then
11:   N.popularityUpdate = true;
12: end if
13: return currentAvail
14:
15: function generateLifePat(Node) i=0
16: while $i \leq \tau$ do
17:     timeToDeparture = retrieveTimeAnalysis(Node, 'fail') or retrieveMonteCarlo(Node, 'fail')
18:     timeToJoin = retrieveTimeAnalysis(Node, 'rejoin') or retrieveMonteCarlo(Node, 'rejoin');
19:     Node.Life(i:timeToDeparture) = 1;
20:     Node.Life(i+timeToDeparture:timeToJoin) = 0;
21:     i = i+timeToDeparture+timeToJoin
22: end while
23: return Node.Life
```

In Algorithm 3, function totalDataRep selects the top k% of data replication for which Q% of query were received. Each node sends a ping message to the replica list after time $t$ the data are replicated. Data popularity and availability evaluation tuple are then retrieved. For each retrieval of the data popularity from replica nodes, the host evaluates the Zipf parameter. The popularity of data at the node is evaluated in lines 3-11. From line 13 to 18, the data to be replicated are selected based on the probabilities of queries satisfied.
Figure 12.2: Aggregation Phase

- Evaluate availability till specified duration (Apply Time Series Analysis or Monte Carlo Simulation)
- Evaluate popularity of data

Figure 12.3: Analysis Phase.

- Evaluate the overall availability for the duration for all neighbors using reliability analysis
- Evaluate popularity of data
- Evaluate replication cost
Algorithm 3 Data replication cost Algorithm

1: function totalDataRep()
2: i=0
3: while $i \leq \text{Node}_\text{Data}()$ do
4:   $C = C + 1/i^\alpha$
5:   i++
6: end while
7: i=0
8: while $i \leq \text{Node}_\text{Data}()$ do
9:   $P(i) = C/i^\alpha$
10:  i++
11: end while
12: data=0
13: while $i \leq \text{Node}_\text{Data}()$ do
14:   if sum(P(1:data)) $\geq$ Query_Popularity then
15:     break;
16: end if
17: data++
18: end while
19: return data

Figure 12.4: Replication Phase.
Part IV

Evaluation
In this section I present my evaluations performed using live traces from the Kad P2P network as presented in[56][57]. The node connectivity comparison was made with networks from PlanetLab\(^1\), BitTorrent\(^2\)[32], OverNet\(^3\)[7] and Kad\(^4\)[80]. Table 12.1 provides an overview for the evaluation performed.

<table>
<thead>
<tr>
<th>Network</th>
<th>Network Size</th>
<th>Mean Uptime (hr)</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanetLab</td>
<td>699</td>
<td>272</td>
<td>19 months</td>
</tr>
<tr>
<td>BitTorrent</td>
<td>5000</td>
<td>19</td>
<td>5 months</td>
</tr>
<tr>
<td>OverNet</td>
<td>3000</td>
<td>1.2</td>
<td>7 days</td>
</tr>
<tr>
<td>Kad</td>
<td>50000</td>
<td>13</td>
<td>14 days</td>
</tr>
</tbody>
</table>

The table shows that PlanetLab has a different characteristic, with a very large mean uptime, which has also been shown in[65]. In [60][20], networks have a very high mean uptime similar to PlanetLab. The data set taken from their evaluation were Long-Muir-Golding (LMG), UCSB, CS instructional Lab (CSIL), and Condor. The nodes in LMG showed a mean uptime of 352.2 hours, CSIL showed a mean uptime of 127.7 hours and Condor had a mean uptime of 1.5 hours. As a result, model presented in those studies is not feasible for most of the P2P networks, and only Condor had the closest mean uptime to be used for analyses. Evaluation performed on PlanetLab and a network with similar characteristics does not hold with the node distribution of the P2P networks. The analyses traces were taken, from four different regions of the world in the Kad network for week. The nodes of the network exhibit diurnal patterns, and while one region show high node activity, the other showed low activity. The analyses of my replication algorithm was performed on a Kad network. Out of the network of 50000 nodes, nodes from two countries were taken, that totalled 1000 nodes, on which analysis was performed on the 2 week traces. The bucket size was set to 20. The Kad network simulation was performed using the event-based P2P simulator peersim [33]. The routing table was refreshed every hour.

\(^1\)http://pdos.csail.mit.edu/strib/plapp/  
\(^2\)http://mikel.tlm.unavarra.es/mikel/bt_pam2004/  
\(^3\)http://sysnet.ucsd.edu/recall/download.html  
\(^4\)http://www.eurecom.fr/btroup/kadtraces/
Chapter 13

Time series analysis

13.1 Evaluation setup

The parameters used for time series analysis per node for forecasting their future behaviour were ARIMA (2, 1, 2). The availability required was set to 0.99. The time duration $\tau$ for generating the life patterns was set to 6 hours, which is the time it takes to transfer highly requested data. During the analysis, 40-45% nodes remained connected to the network, for a total load of 100 files per node, with each file being 5 MB in size. The bandwidth of the nodes was randomly taken from a range of 5-10 Mbps. The Zipf constant $\alpha$ and data popularity were constant for analysing this algorithm. The $\alpha$ was set at 0.8 and the query data popularity requirement was 90%.

I compared my algorithm with current approaches using the availability [9] and lifetime [37] of the nodes. My approach stabilizes in the first two days in terms of per node average replica.

13.2 Required replicas

To determine the required replicas, I tested the mean replicas created by the nodes in the network for the given availability requirement. The replication algorithm was allowed to run for a period of time, during which data had been collected to determine the number of replicas created in the network. As shown in Fig 13.1, the number of replicas required for my algorithm
is markedly lower than other approaches. The mean number of replicas using my algorithm is approximately 5; for TotalRecall it was 15 using Lifetime aware replica placement gives a mean of 13. In my algorithm, given the additional information required for determining replica nodes, the reduced number of replica nodes required reduces the overall traffic load. Therefore, the overall storage requirement is less for my approach.

### 13.3 Reliability analysis

To test data survivability in the network, the reliability of the nodes in the replica set were tested, (i.e the set of nodes chosen to be replica placement nodes) is tested. This analysis shows how the replica placement strategy improves data survivability.
Figure 13.2: Comparison of the reliability of replica nodes TotalRecall, Lifetime aware replication, and my algorithm (replications using lifetime prediction).

The analysis was performed to determine the reliability of the nodes for all of the replica placement algorithms, as shown in Figure 13.2. The reliability of the set of nodes chosen for replication in my algorithm was higher than other approaches for the duration of time $\tau$, even when using fewer replica nodes. This shows that the replication algorithm presented here has a better replica placements algorithm.

### 13.4 Replica set availability

The replica set availability analysis checks the dependence of the algorithms on highly available nodes. The replication algorithm presented here includes the availability parameter, and the nodes with higher availability are likely to be selected for replication. The analysis of the replica set availability shows which algorithms rely heavily on highly available nodes to improve availability, and increase the traffic overhead on them.
Figure 13.3 presents the average availability of the replica set using my algorithm. The mean availability of nodes using my algorithm was less than the TotalRecall and Lifetime aware replica placement algorithms. The replica set in the algorithm presented, relies on the nodes that complement the presence of each other in the network. This provides the higher reliability without dependence on the highly available nodes. A host node makes sure that the replica nodes are available when the host node is unavailable. The number of replicas increases, until the required availability is not reached.

13.5 Lookup failure

To determine the data unavailability, each was queried for data every 15 min in order to determine data unavailability. This approach to evaluation determines the replica placement and availability of the data in the network.
The results are shown in Fig 13.4. The failures rates for nodes using my algorithm showed approximately 0.008 mean failures. The failure rates for TotalRecall and Lifetime aware replications increased to 0.25 and 0.06, respectively. I did not include the number of hops required to retrieve the data because I was only interested in determining the the data survivability in the network. This analysis shows that the replication approach presented here selects a replica set that maintains the lowest level of data failure.

**Table 13.1: Message overhead**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>My algorithm</td>
<td>19</td>
</tr>
<tr>
<td>Lifetime aware</td>
<td>31</td>
</tr>
<tr>
<td>TotalRecall</td>
<td>37</td>
</tr>
</tbody>
</table>
13.6 Message overhead

An evaluation was performed to capture the average number of messages per node (Table 13.1). The number of messages using my approach was 19, even with an overhead in the size of the message. This reduces the network traffic compared to the Lifetime aware and TotalRecall replications in maintaining the replicas, which required 31 and 37 messages, respectively.
Chapter 14

Monte Carlo simulation analysis

14.1 Evaluation setup

The analysis of the Monte Carlo simulation requires analysis of node availability in the network. Through availability analysis of nodes using Matlab, the MTTF and MTTR are taken to fit to the Weibull and Lognormal distributions, respectively. Nodes using the Weibull distribution had parameters $\mu$ from 3.4 to 6 and $\sigma$ from 2.7 to 3. For the Lognormal distribution, parameter $a$ was from 62 to 900 and parameter $b$ was between 1 to 3. Each node selects these parameter values randomly for each distribution. I start with a brief introduction to Total Recall, to which my algorithm is compared. The Zipf constant $\alpha$ and data popularity are constant, and in the analysis of my algorithm, $\alpha$ is set at 0.8 and the query data popularity requirement is 90%.

Two repair policies were evaluated eager and lazy repair. In lazy repair, the threshold is set to 5 replicas. This value is the average replicas required, which was determined based on the evaluation of my algorithm. When the number of replicas falls below the threshold, the replica management starts. The availability information storage had been varied from a 2 hours to 1 day of node network connectivity information.
14.2 Required replicas

In the eager repair policy in my approach, the number of replicas decreased as the availability storage information increased. The mean replicas created during the analysis with varied availability information decreased from 12 to 6. When using the lazy replica policy, the mean number of replicas decreased to a range between 9 and 4, with the varied availability information. When using replica management of TotalRecall, the eager repair showed similar behaviour, but required twice as many replicas compared to my approach, starting with means of 28 to 20. When using lazy repair, this fell to a range of 23 to 16 replicas. Using 1 day of connectivity information, as shown in Figure 14.1, eager repair results shows that the mean replicas required by TotalRecall is 18, where as the mean replicas required in my approach is 8. Changing the repair policy to lazy repair, decreases the replicas required in both the approaches. Figure 14.2, shows the mean replicas required for TotalRecall decrease to 17. The decrease in replicas was because the eager repair compares the availability required, and as soon as it drops to the required availability, the replicas are increased. However, in the lazy approach, it waits until the threshold is reached, at which point replica management does not start.

14.3 Lookup failure

The failure ratio is the, number of results returned with respect to the number of replicas made. When evaluating my approach for failure, I found that the increase in availability information storage had an inverse effect on failures, with a slight improvement in the failure in lookups with low availability storage. The mean failure increased from 8% to 15% with an increase in availability information of the nodes for eager repair. When using the lazy repair, similar results were obtained, but they had higher mean failure. The mean number of failures for lookups was 37% which increased to 43% with an increase in availability storage. When using TotalRecall replica management, the failure ratio for eager repair showed same results as my approach, but for lazy repair the failure increased from 53% to 60%. The analysis for eager repair is shown in Figure 14.3. The failure ratio is very similar for TotalRecall and my approach, with a mean failure of 9%. Using lazy repair, the failure increases as the nodes to which the data had been replicated have departed, and the replica list is not updated until the threshold has been reached (Figure 14.4). The mean failure ratio for TotalRecall was 55% compared to 40% in my approach.
Figure 14.1: The CDF of the number of replicas made in TotalRecall as well as Availability and life pattern-based replication approaches using eager repair.
Monte Carlo simulation analysis

Figure 14.2: The CDF of the number of replicas made in TotalRecall as well as Availability and life pattern-based replication approaches, using lazy repair.
Figure 14.3: The CDF of the failure ratio to lookups in Total Recall as well as Availability and life pattern-based replication approaches using eager repair.
Figure 14.4: The CDF of the failure ratio to lookups in Total Recall as well as Availability and life pattern-based replication approaches using eager repair.
Part V

Conclusion
Chapter 15

Summary and future work

The P2P network helps share data between users and provides a vision of a single shared resource to be used by users. Each user in the network acts as a server to provide resources and data usable to the network. Research on P2P networks concentrate on optimizing the usage of these resources. However several challenges arise while working with P2P networks, and based on the analysis of P2P networks I have attempted to address the following:

- Modelling user behaviour.
- Utilizing user availability.
- Reducing replication overhead.
- Reducing data search failure.

15.1 Summary

In the preliminary section of this thesis, I presented a background of the field, including the main motivation for the thesis and an analysis of P2P networks, I also presented and detailed the replication approaches used in distributed computing. Some work towards fault management and analysis were presented as well, and the approaches to be used for user behaviour analysis were addressed in the prediction models.

Model user Behaviour: Users in the network exhibit certain behaviours in a P2P network. The ability to capture that behaviour and utilize it based on statistical analysis is useful for
searching and storing data. In Chapter 4, the reliability analysis introduction covered some of the details that are currently used to model users. There I also show how the theory of reliability analysis is useful for the P2P network domain. As part of my hypothesis, a better user availability prediction can help improve data availability in the network. In Chapter 5, the description of prediction models provides some detail of the techniques applied to P2P networks in my work.

**Contribution:** The core focus of the thesis was to address the issue of modelling user behaviour. In Chapter 11, I determine how the P2P network can be modelled by providing a description of reliability analysis. I present a P2P network modelled as a connection between user and their modelling, using reliability analysis theory, how the availability of each node affects the reliability of the network, and how the overall data availability is affected by the node availability. By analysing P2P networks, I concluded that having an accurate prediction approach to user behaviour can improve the overall availability of data in the network. The predication model presented in Chapter 10, which is based on the details in the background section, uses user availability from the past patterns to determine possible future availability of the user. The prediction approaches used here have been successfully utilized in financial domains to predict market behaviour, which is highly dependent on user trends.

**Utilizing user availability/reducing replication overhead:** The heterogeneous nature of a P2P network poses a substantial challenge to optimizing approaches that address resource sharing and replication issues. An algorithm is required that can use this heterogeneity and integrates the prediction model.

**Contribution:** Given the model for predicting user behaviour, I develop an algorithm in Chapter 12 that uses this model in a P2P network to share availability information and replicates data to an appropriate number of nodes in order to maintain a high level of availability. The algorithm utilizes the heterogeneous resources in the network, given the bandwidth and data popularity constraints. The algorithm routes data that meets the popularity requirement and transferable in the time frame when both the user from whom and to whom the data is been transferred are available. The transfer time is calculated based on the bandwidth available, and given the possible user availability prediction, the algorithm calculates if the user is suitable to transfer the data within the time frame. This would reduce the unnecessary transfer of data to nodes with low availability, as observed in the evaluation. The number of replicas required based on my evaluation were less than other approaches. The reduced number of replicas required and the transfer load reduction, also decreases the number of overall messages required to transfer and collect availability information from nodes.
Reducing data search failure: Lookup failure of data in a network is partially due to the failure to maintain a high level of data availability. If a high level of availability is provided for the data, then the lookup loss can be reduced.

Contribution: In Chapter 12, I describe an algorithm that reduces the lookup failure when given availability information. From the evaluation performed in Chapter 13 and 14, I found that a reduction in the number of replicas did not reduce the availability of the data. The search evaluation performed showed that the number of data lookup failures was very low compared to other approaches. Based on the evaluation, the average availability of the replica set of my algorithm was much lower than the other approaches, but it maintain a higher level of data availability. This also shows that compared to other approaches, my algorithm distributes the load evenly, without ignoring less available nodes.

15.1.1 Comparison of the time series analysis and Monte Carlo simulation

Based on the experience and knowledge gained from the analysis of these two approaches, I can conclude with the following comparison:

Modelling Flexibility One of the main reasons to use time series analysis using an ARIMA model was to capture the changing behaviour of the users. The ARIMA model is flexible with the changing modelling requirements. A Monte Carlo simulation relies heavily on the predetermined distribution fitting. A Monte Carlo simulation was originally developed to study properties of equilibrium system. However it is not universally accepted that this method can also be used to simulate a system that is not in equilibrium. The Monte Carlo simulation can model a complex system with higher dimensions.

Model Complexity The Monte Carlo simulation does not require as many parameter estimations compared to an ARIMA model. It only requires fitting the distribution and gathering the parameter values from the fitted data. The ARIMA model requires a differencing parameter, AR and MA parameters, and coefficient determination for the AR and MA parameters. In addition a good ARIMA model requires more experience for analysis of data, compared to the Monte Carlo simulation.

Prediction Accuracy The ARIMA model showed better accuracy for less historical data. However the increased number of runs for the Monte Carlo simulation improves the
accuracy of the results, and running the simulation 10000 or more runs showed better results.

Error Handling The accuracy of the ARIMA model prediction is due to the noise factor added in model. A feedback loop in the model adjusts the error in the prediction. Compared to the Monte Carlo simulation, a large number of runs is required without taking prediction errors into account.

System Modelling The ARIMA model captures some of the characteristics of the data by showing correlation functions and error feedback. The Monte Carlo simulation masks the underlying data complexity. The data can only be explained based on the distribution fitting.

Model Coverage Both methods can be used for stochastic (involving probability) and deterministic (without probability).

15.2 Future work

The major focus of this thesis was addressing replication in a P2P network; however, future work could analyse the following points: but the future work tends towards few direction which can be as follows:

- Trust-based approach to replication.
- Social network application.

In future work, I could explore possible applications of these findings by assessing the usage of the approaches addressed in this thesis. A core research question for P2P network research involves trust-based systema where replication information is shared between the nodes.

Trust-based approach to replication: Trust-based replication refers to the level of trust each node in the network has on its neighbour and how trustworthy the availability information is that is shared by the neighbour. An approach such as the Byzantine fault detection [30] would be needed to capture the requirement where a user can share false information in order to gain access to maximum resources and destabilize the network. Since the approach presented in this thesis is highly dependent on the availability information shared by the node, it would be very important to understand the level of trust offered by that node. This would result in extra
overhead for gathering data about the level of trust that neighbours have for another neighbour, but the cost of failure to predict the behaviour would be higher. The data loss from inaccurate prediction could also be very high.

*Social network application: As shown in this thesis, a P2P network resembles a social network. One possible usage of my approach would be the avoidance of having a central resource to share data, where each user in the network could maintain a copy of data for another user in order to improve the overall data availability. A social network is basically a sharing of information that requires very similar algorithms for improving information visibility as those used for a P2P network. The re-use of information in those networks and the maintenance of a certain level of availability of this information is a possible research direction for my approach in social networks.*
Bibliography


and Security of Distributed Systems (SSS ’09), pages 472–484.


# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Chord Routing Identifier Space</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Chord Routing Structure</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>DKS level 1 and 2 intervals</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>DKS level 3 intervals</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Routing table of node 0</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Lookup for Id 20, from node 0</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Identifier space for Kademlia</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>Routing in Kademlia</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>P-Grid Network</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>Monte Carlo simulation process</td>
<td>42</td>
</tr>
<tr>
<td>6.1</td>
<td>Chord with replication factor of 1 and 2</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>Symmetric Replication with factor 2</td>
<td>52</td>
</tr>
<tr>
<td>6.3</td>
<td>Kademlia with factor 3</td>
<td>53</td>
</tr>
<tr>
<td>6.4</td>
<td>P-Grid Replication</td>
<td>54</td>
</tr>
<tr>
<td>6.5</td>
<td>Behive Replication</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>Markov model for replication</td>
<td>61</td>
</tr>
</tbody>
</table>
8.1 The overall process of data replication. 71
9.1 ARIMA model process. 76
9.2 Node lifetime. 76
9.3 Non-Differentiated Node Online Data Time Series. 80
9.4 Node online time series-1st differentiation procedure. 80
9.5 Differentiated Node Online Time Series. 81
9.6 Differentiated Node Online Data Time Series. 81
9.7 Distribution fitting for Node online and offline time. 90
10.1 ARIMA node online time forecast. X-axis is time in hrs calculated with per unit time of 6 hrs. 93
10.2 Monte Carlo node online time forecast. X-axis is time in hrs calculated with per unit time of 6 hrs. 94
12.1 Algorithm Cycle. 100
12.2 Aggregation Phase 106
12.3 Analysis Phase. 106
12.4 Replication Phase. 107
13.1 Comparison of the number of replica nodes required by TotalRecall, Lifetime aware, and my algorithm (replications using lifetime prediction). 114
13.2 Comparison of the reliability of replica nodes TotalRecall, Lifetime aware replication, and my algorithm (replications using lifetime prediction). 115
13.3 Comparison of the average availability of replica nodes set in TotalRecall, Lifetime aware replication, and my algorithm (replications using lifetime prediction). 116
13.4 Comparison of replica failures in TotalRecall, Lifetime aware replication, and my algorithm (replications using lifetime prediction) . . . . . . . . . . . . . . . . 117

14.1 The CDF of the number of replicas made in TotalRecall as well as Availability and life pattern-based replication approaches using eager repair . . . . . . . . . . . . . . 121

14.2 The CDF of the number of replicas made in TotalRecall as well as Availability and life pattern-based replication approaches, using lazy repair . . . . . . . . 122

14.3 The CDF of the failure ratio to lookups in Total Recall as well as Availability and life pattern-based replication approaches using eager repair . . . . . . . 123

14.4 The CDF of the failure ratio to lookups in Total Recall as well as Availability and life pattern-based replication approaches using eager repair . . . . . . . 124
List of Tables

9.1 ACF and PACF patterns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

11.1 Reliability system model notations . . . . . . . . . . . . . . . . . . . . . . 96

12.1 Evaluation overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111

13.1 Message overhead . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117