IMPROVED LOAD-BALANCING FOR A CHORD-BASED PEER-TO-PEER STORAGE SYSTEM IN A CLUSTER ENVIRONMENT

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2015

By
Fu Chen
School of Computer Science
Contents

Glossary 10
Abstract 17
Declaration 19
Copyright 20
Acknowledgements 21

1 Introduction 22
  1.1 Overview ........................................... 22
  1.2 Chord ............................................. 24
    1.2.1 Chord Ring .................................... 25
    1.2.2 Maintaining Chord ............................. 27
    1.2.3 Key-value Pair Operations ..................... 27
  1.3 Chord-based Storage System ......................... 28
    1.3.1 File Structure ................................ 28
    1.3.2 File Operations ............................... 30
  1.4 Research Hypothesis ................................ 31
  1.5 Contribution ..................................... 31
  1.6 Thesis Structure ................................ 33

2 Load-balancing in Distributed Hash Tables 35
  2.1 Introduction .................................... 35
  2.2 Natural Load-Balancing in
    Distributed Hash Tables ............................ 36
    2.2.1 Consistent Hashing ........................... 36
    2.2.2 Plaxton Mesh ................................. 42
List of Tables

2.1 Comparison of the four DHTs. ............................................. 48
3.1 Average MND of all points (squares) in two-dimensional space. .... 73
3.2 Average MND of all points (cubes) in three-dimensional space. .... 74
3.3 Average MND of all points (vectors) in four-dimensional space. .... 74
3.4 Average ranking correlation coefficient of all points (squares) in two- dimensional space. ....................................................... 75
3.5 Average ranking correlation coefficient of all points (cubes) in three- dimensional space. ..................................................... 75
3.6 Average ranking correlation coefficient of all points (vectors) in four- dimensional space. .................................................... 75
3.7 Possible values of dimension $n$ and order $b$, if $d \times b = 160$. .... 78
3.8 The 11 machines’ Chord ID, RTTs to Landmark 1 and Landmark 2. ... 84
4.1 Minimum size of available storage space of file system containing /tmp directory and the number of machines in the five experiment laboratories. ................................................................. 105
4.2 Information on DHash’s three dependent packages. .................. 106
5.1 Average CPU usage rate and average memory usage in experiments with different numbers of virtual nodes per EM. ................... 124
5.2 Experimental duration (in seconds) from start to FWTP and from FWTP to UTP in experiments with 1, 2 and 5 Virtual Nodes (VNs) per EM. . 134
5.3 Reliable system storage utilisation $(U_{system,UTP})$ and system storage capacity $(L_{capacity})$ for experiments with 1, 2, 5 virtual nodes (VNs) per EM. Experiments with 2 and 5 virtual nodes are provided a comparison between the storage system with P_VNode and that without P_VNode. 136
5.4 Experiment insert instructions and insertion successful rate ($R_{insert}$), average block insertion latency ($Ave.LI$) and average hops ($Ave.H$) in different experiment phases. After ‘Insertion 7’, there is a ‘no operations’ period until the end of the experiment.

5.5 Experiment retrieval instructions, block retrieval successful rate ($R_{retrieve}$), average block retrieval latency ($Ave.LR$) and average hops ($Ave.H$) in different experiment phases. After ‘Retrieval 7’, there are no file operations until the end of the experiment.

5.6 The number of hotspots detected by the hotspots detection approach in different experimental phases. For each experiment, the timeout value (TO) is set to be 30 seconds, 60 seconds, 90 seconds or 120 seconds.
# List of Figures

1.1 A Chord Ring with capacity of $2^6$, 9 nodes and 11 key-value pairs. ... 26
1.2 File structure in a Chord-based storage system. ......................... 29

2.1 A Chord Ring with 12 virtual nodes in 6 machines. ................. 39
2.2 A CAN space of 2-dimensional Cartesian coordinate space on a 2-torus 
   (d = 2). ................................................................. 40
2.3 A CAN space of 2-dimensional Cartesian coordinate space on a 2-torus 
   with 16 nodes and 15 key-value pairs (including KV1, KV2 and KV3). 41
2.4 Routing from node 348AE to node E4601 in a Plaxton mesh and the 
   neighbour map of node E4601. ........................................ 43

3.1 Simultaneous inbound network traffic to a machine by key-value pair 
   insertion operations. ...................................................... 67
3.2 Simultaneous outbound network traffic from a machine by key-value 
   pair retrieval operations. .................................................. 68
3.3 Peano Curve, Reflected Binary Gray-code (RBG) Curve and Hilbert 
   Curve of order 2 pass through 2-dimensional squares. ............... 71
3.4 Procedure of producing Hilbert curves of order 1, 2, and 3 [46]. .... 72
3.5 Three parts of a Chord node’s Hilbert table. ........................... 80
3.6 The 11 machines on the Chord Ring with their Chord IDs. ........... 83
3.7 2-dimensional order 3 Hilbert curve with the 11 machines in the same 
   2-dimensional square space. .............................................. 85
3.8 The Hilbert key-value pairs of the 11 machines on the Chord Ring and 
   the main part of the nodes’ Hilbert tables. ............................ 86
3.9 Hilbert tables of the 11 machines on the Chord Ring. .................. 87

5.1 Maximum, minimum and average CPU usage rate for different intervals between file insertions. ........................................ 127
5.2 Maximum, minimum and average memory usage for different intervals between file insertions. ........................................... 128
5.3 Proportion of different RPC messages in the entire network sending traffic in the INS scenario. ..................................... 129
5.4 Proportion of different RPC messages in the entire network sending traffic in the RNS scenario. ................................. 130
5.5 $D_{\text{utilisation}}$ represents the uniformity of the storage utilisations across all EMs. This figure compares the Chord-based storage system with P_VNode and the one without P_VNode. ......................... 132
5.6 The zoomed in figure on the range between 65 and 70 of $D_{\text{utilisation}}$ demonstrates the changes of $D_{\text{utilisation}}$ when the Chord-based storage system includes the P_VNode load-balancing approach. .......... 133
5.7 Durations from experiment start to the time points FWTP and UTP in experiments with 1, 2 and 5 virtual nodes (VN) per EM. The right bars aside of experiments with 2 and 5 virtual nodes per EM are those for the experiment with the P_VNode load-balancing approach. .... 135
5.8 Average block insertion successful rate ($R_{\text{insert}}$). .............. 140
5.9 Average latency of block insertions ($Ave_{\text{LI}}$). .................... 141
5.10 Average hops of block insertions ($Ave_{\text{H}}$). ....................... 142
5.11 State diagram of hotspot detection approach. ...................... 143
# List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Overloaded machine: $EM_o$</td>
<td>91</td>
</tr>
<tr>
<td>3.2</td>
<td>Rendezvous node: $N_r$</td>
<td>92</td>
</tr>
<tr>
<td>3.3</td>
<td>Target machine: $EM_t$</td>
<td>92</td>
</tr>
</tbody>
</table>
Glossary

**All Warning Time Point** (AWTP) is the time when all machines in the Chord-based system storage system reach X% storage utilisation. 65

**Balance** means that objects are evenly distributed across all cache machines from a specific client view in consistent hashing. 37

**Bootstrap machine** is the machine in a Chord-based storage system which is used by other machines to join the same Chord-based storage system. 27

**CAN space** is a conceptual space that is divided for the nodes in CAN. It is a d-dimensional Cartesian coordinate space on a d-torus. Figure 2.2 shows a CAN space with d=2. 39

**Chord ID** is the identifier for a node or a key-value pair on the Chord Ring. 25

**Chord Ring** is a conceptual circular structure on which Chord nodes are located, which is the ID space of Chord IDs. 25

**Chord storage space** is the part of local storage space used by a Chord-based storage system to store key-value pairs in the machine. 62

**Consistent hashing** is a distributed data structure to implement the functionality of a conventional hash function in a distributed environment. 36

**CPU running time** is the amount of time for which a CPU was used for processing instructions of a specific computer program. 111

**CPU usage rate** ($U_{CPU}$) is the percentage of the total CPU running time for a specific computer program from the elapsed time duration. 111

**Current received bytes of data** ($NR_t$) is the amount of data in bytes that has been received by a machine before the time point $t$. 112
Current sent bytes of data \((NS_t)\) is the amount of data in bytes that has been sent out of a machine before the time point \(t\). 112

Data block is a continuous segment of data in a file. 28

Data Block ID is the identifier of a file data block, which is produced by a hash function from the content of the file data block. 29

Data ID is the identifier of a key-value pair in a DHT. 25

Data message is an RPC message containing a key-value pair and is produced by the two file operations. 113

DHT Distributed Hash Table. 22

DHT-based storage system denotes a distributed storage system that is implemented by a DHT. 28

EM Experimental Machine. 101

Experimental file content is the randomly produced data for the experimental files. 109

Experimental file size is the size of the experimental files. 109

File block is a segment of a file to be stored in the Chord-based storage system. 29

File insertion is the process of storing a file into a storage system. 30

File insertion capacity is the acceptable file insertion frequency which does not produce file insertion failure in the Chord-based storage system. 138

File insertion phase is an experimental phase in which experimental files are inserted into the Chord-based storage system. 118

File operation includes file insertion operation and file retrieval operation. 30

File retrieval is the process of searching for and fetching a specific file from a storage system. 30

File retrieval capacity is the acceptable file retrieval frequency which does not produce file retrieval failure in the Chord-based storage system. 139
File retrieval phase is an experimental phase in which EMs retrieve experimental files from the Chord-based storage system. 118

Finger table records the successors of the Chord ID that are $2^k$ distant from the current node’s Node ID clock-wise on the Chord Ring, where $0 \leq k \leq m - 1$ and $m = \log(\text{Chord Ring capacity})$. 27

First Warning Time Point (FWTP) is the time when the first machine in the Chord-based storage system reaches X% storage utilisation. 64

Global consistent algorithm is a predefined global method for the Plaxton mesh to store an object in a node in the system when there are no nodes with the same Plaxton ID as the object to be published. 43

Hilbert ID is an identifier of a machine produced by the RTTs from the machine to landmark machines using a Hilbert curve. 70

Hilbert key-value pair is a key-value pair with Hilbert ID as its key and one machine’s network information as its value. 77

Hilbert table is a table to keep the Hilbert key-value pairs in the node’s responsible zone on the Chord Ring. 79

Hop is the number of machines by which a message is forwarded on its way from its source to its destination. 115

Hotspot is a single machine with persisting long-time high network transfer rate that prevents it from fulfilling further requests. 56

Indirect block contains the Data Block IDs of a list of data blocks. 28

Indirect Block ID is the identifier of a file indirect block, which is produced by a hash function from the content of the file indirect block. 29

Inode block contains the Indirect Block IDs of a list of indirect blocks. 28

Inode Block ID is the identifier of a file inode block, which is produced by a hash function from the content of the file inode block. 29

INS Insertion Normal Scenario for an experiment. 120
Interval is the whole range of a one-dimensional indexing that is labelled by a series of consecutive integers. 71

IOS Insertion Overload Scenario for an experiment. 121

Key-value pair is the basic storage unit of a hash table. Key is the label of the key-value pair. Value is the actual data to be stored. This is also a basic storage unit of DHTs. In some cases, there is only the value without a corresponding key. 22

Key-value pair insertion is a primitive operation to store a key-value pair into a DHT. 28

Key-value pair operation contains key-value pair insertion operation and key-value pair retrieval operation. 30

Key-value pair retrieval is a primitive operation to retrieval the value of a key-value pair with specific key. 28

LAN Local Area Network. 102

Landmark machine is a machine that is chosen to produce a machine’s network coordinate which is a vector of RTTs from all landmark machines to the machine in the Chord-based storage system. 54

Landmark vector is a vector with $n$ elements, each of which is the network proximity scalar to one of $n$ landmark machines in a landmark approach. 76

Load is the maximum number of possible objects which may be stored in a cache machine from all client’s views in consistent hashing. 37

Machine is a practical computer machine that is connected to a network in a system. 25

Maintaining message is one of the essential messages transferred in Chord to ensure the Chord DHT works properly. 113

Memory usage ($U_m$) is the amount of memory space that is utilised by a specific computer program for its running. 111

MND Maximum Neighbour Distance. 73
**Monitoring resolution** is the recording frequency of the four monitoring programs in the experiment monitoring system. 108

**Monotonicity** means that object migration from an old cache machine to a newly joined machine only happens in order to evenly distribute objects after joining of the new cache machine in consistent hashing. 37

**Network boost traffic** is a temporary situation that is caused by a number of file operations happening to come through the same machine in the Chord-based storage system at the same time. 144

**Network distance** is a variable indicating the proximity between two machines in the network according to some network proximity scalar. 54

**Network proximity scalar** is a primitive measurable scalar that reflects two machines’ network proximity property, such as Round Trip Time or IP routing hops. 54

**Network saturation** is a long term situation caused by too frequent file operations in a busy Chord-based storage system. 144

**Network structure** is the organisation of the computer network connecting all participating machines in a Chord-based storage system. 66

**Network utilisation** is an individual machine’s network usage that is produced by the network operations from the participating machine in a Chord-based storage system. 66

**Node** is a conceptual representation of a practical machine or virtual node in a DHT’s ID space. 25

**Node ID** is an identifier for a node in a DHT. 25

**NS** Normal Scenario for an experiment. 120

**Object ID** is the identifier of an object that is stored in a DHT. 48

**Operational capacity** is the acceptable file operation frequency which cannot produce operation failure in the Chord-based storage system. 138

**Owner machine** is the machine that should store the key-value pair according to the key-value pair insertion operation in Chord. 95
**P2P** Peer-to-Peer. 22

**P_VNode** is the proximity-aware virtual node storage load-balancing approach for the Chord-based storage system. 88

**Plaxton mesh** is a distributed data structure to implement message routing and object location in a static distributed environment. 42

**Predecessor** is the node whose Chord ID is immediately close to the given Chord ID counter-clockwise. 27

**Proximity neighbour selection** means that each machine detects and records some machines that are ‘near’ from it, according to their proximity information. 53

**Proximity routing** means that messages are routed to their destinations across the system with respect to machines’ proximity information at each routing hop. 53

**Publishing machine** of a key-value pair is the machine from which the key-value pair is inserted into a DHT. 28

**Receiving network transfer rate** \((R_{recv})\) is the amount of data that is received by a machine within a unit of time. 112

**Reliable system storage utilisation** is the system storage utilisation at UTP. 117

**Rendezvous node** is the node storing the Hilbert key-value pair of some machines. 79

**Replication machine** is the machine that stores a replica of a key-value pair for other machines in the Chord-based storage system. 95

**Replication table** is a data structure that records the locations of all replicas of the key-value pairs in a machine. 95

**Responsible zone** of a node \(N\) is a set of Chord IDs between the Chord ID of node \(N\) inclusive and the Chord ID of \(N\)’s predecessor exclusive on the Chord Ring. 27

**RNG** Random Number Generator. 109

**RNS** Retrieval Normal Scenario for an experiment. 120

**Root node** is a node to record the storage location of an object. 43
**ROS** Retrieval Overload Scenario for an experiment. 121

**RPC** Remote Procedure Call. 105

**RTT** Round Trip Time. 76

**Sending network transfer rate** ($R_{send}$) is the amount of data that is sent out of a machine within a unit of time. 112

**SFC** Space-Filling Curve. 70

**Spread** is the number of cache machines required to store an object, when the object is inserted into the cache layer by a certain number of clients from their different views in consistent hashing. 37

**Stable phase** is the beginning phase in all experiments and ensures that the Chord-based storage system is in a stable state before each experiment commences. 118

**Successor** is the node whose Chord ID is immediately close to the given Chord ID clockwise. 27

**Surrogate routing** is a routing method when there are no nodes with the same ID as the object to be published in Tapestry. 44

**System storage capacity** is the total used local storage space of machines at UTP. 117

**System storage space** is the part of local storage space used by the operating system and other applications in a single machine. 62

**System storage utilisation** ($U_{\text{system},t}$) is the overall storage utilisation of the Chord-based storage system at time $t$. 116

**System structure construction** means that a system structure is constructed with respect to each joining machine’s proximity information. 53

**Unreliable Time Point** (UTP) is the time when the first machine in the Chord-based storage system reaches 100% storage utilisation. 64

**Virtual node** is one of multiple conceptual representations of a practical machine in a DHT’s ID space. There may be more than one virtual node in a machine. 38
Abstract

IMPROVED LOAD-BALANCING FOR A CHORD-BASED
PEER-TO-PEER STORAGE SYSTEM IN A CLUSTER ENVIRONMENT
Fu Chen
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2015

The thesis investigates deployment of a Peer-to-Peer storage system in a cluster environment, in which machines have good and persist network connection, in order to provide the functionality of a data centre. For various reasons, the implementation is based on the Peer-to-Peer system known as Chord. Chord naturally provides storage load-balancing, especially if its virtual node scheme is used, but this needs to be improved if Chord is used to implement a storage system. A novel, threshold-based storage load-balancing scheme is proposed. Each machine in the system contributes a fixed amount of disk storage space to the Peer-to-Peer storage system. The system commences operation in the normal Chord manner except that two distinct sets of tables are initialised, one to maintain the usual Chord Ring, and one to maintain proximity information about the machines in the system. As files are inserted, the collective storage space gradually fills up. When any machine reaches the threshold for usage of its contributed space, the system behaviour is modified. Attempts are made, repeatedly if necessary, to migrate virtual nodes from heavily loaded machines to less-heavily loaded machines elsewhere in the system. The proximity information is used so as to minimise the costs of this migration. The nature of the proximity information is complex, and a Space-Filling Curve is utilised to reduce the complexity. For reasons of effectiveness, demonstrated by an evaluation against other kinds of Space-Filling Curve, the Hilbert curve is specifically chosen. The performance of the
resulting implementation is evaluated in a practical experimental environment which consists of five teaching laboratories in the author’s school. Under the specific conditions of the experiments, the new system achieves significantly better distribution of storage utilisation across the participating machines and also defers the onset of unreliable behaviour in the system. In one experiment, the amount of the total storage space available that is actually utilised by the system increased from \( \sim 43\% \) to \( \sim 62\% \) using the proposed mechanism. The parameters used in the experiments have been chosen somewhat arbitrarily, so it is possible that even better results might be feasible.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copy-right works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgements

In the long duration of my PhD research, my supervisors, Professor John Gurd and Doctor Nick Filer, have provided me with great assistance.

Under the supervision from John, I found the direction of my research and passed the interview at the end of the first year. He also made a great deal of effort to review this thesis from the perspectives of both the academic view and English writing. My research includes a large number of experiments which require over 200 practical machines in the several student laboratories in the School of Computer Science in the University of Manchester. Nick helped me to ask for permissions to change the setting of machines and to setup the experimental environment. During regular meetings with John and Nick, a large number of various inspiration were obtained from the talking with them.

Furthermore, when I suffered from severe tension headache due to the sleeping disorder, high pressure and depression, Nick and John have given me useful advises and solutions in order to let me recover quickly. This is the main reason why I was able to continue my research and then to complete it with this thesis.

During the whole period of my research, I have also received valuable advises and suggestions from my friends: Zixu Song, Wuxiang Ge, and Zuo Liu.

I appreciate my parents and my wife. My parents supported all my study life, because of which, I have never worried about the life in Manchester. My wife accompanied me aside when I was making the last correction on the thesis.

The completion of my research and this thesis cannot be separated from the assistance that is provided by my supervisors, my family and my friends.
Chapter 1

Introduction

1.1 Overview

The last decades have witnessed massive deployments of Peer-to-Peer (P2P) systems on the Internet. From the first P2P system – Napster [4] was available to the public in 1999 – P2P systems have increasingly caught attention from the research community. Napster provided a file sharing service to its users who register the list of their files in Napster’s central server, through which all users are able to query and to locate the files they want. The recorded list of music files in the central server forced Napster to be closed in 2001, because the system potentially facilitated users’ infringement of the copyrighted music, which should have been noticed by Napster by checking the file list in the central server [11]. However, Napster’s failure also stimulated a development of the P2P system to its second generation. The initial versions of Freenet [7] and Gnutella [17] are representatives of the second generation of P2P systems, in which the central server is removed. The system has no direct way to obtain knowledge of the files shared among users. Therefore, the second generation P2P systems not only solved the legal problem, they also provided high scalability due to the lack of a central server. Some systems also provided more features, such as anonymity [7]. File sharing service is the main application field for the first and second generation P2P systems. When the P2P system evolved into its third generation, it found other application fields. Using a Distributed Hash Table (DHT) [50, 24, 42, 29, 23] is the distinguishing feature of the third generation of P2P systems. A DHT provides conventional hash-table-like functions to other components as an overlay structure in a distributed system. A conventional hash table provides a put operation and a get operation in order to insert a key-value pair into the hash table and to retrieve the value according to the given
key from the hash table, respectively [19]. All related operations and storage space involved in both these two functions of the conventional hash table are in a single machine. A DHT distributes the storage space over multiple machines and provides the corresponding two operations, namely key-value pair insertion and key-value pair retrieval. All the research in this thesis is based on a DHT called Chord [50] that will be described in more detail in Section 1.2.

The essence of the P2P system is the two roles of each machine, the server role and the client role. The conventional client-server model gives each machine a fixed role, either a server or a client, which determines that the number of servers is fixed in a conventional distributed system. In a P2P system, a participating machine is able to upload files to other users while it is downloading other files. All machines in a P2P system have files that other users want. In this way, a P2P system theoretically has the same number of servers as the number of clients, which increases the capability to fulfil requests. Due to the lack of a central component, a DHT has high scalability and the structure of the machines is self organised automatically.

Due to each machine’s dual roles, the conventional server machine can be replaced by clients themselves in a P2P system. A conventional central server usually has faster CPU computing power, bigger memory and storage space and broader bandwidth of network connections than that of the client machines. The philosophy of P2P is that of contribution. Each client machine contributes its CPU computing power, its storage space and its network connections to the whole P2P system, which collectively utilises all these contributed computing resources. Although there is no powerful central server, a P2P system is still able to provide the same quality services. As for DHT, a large database is not necessary to store all objects for participating machines, because each participating machine contributes its storage space to the whole system. The whole system’s network bandwidth is now equal to the total network bandwidth of all machines, because the whole system’s storage space is accessible from all machines. DHT also has a natural load-balancing property which can evenly distribute stored objects across all participating machines. Furthermore, because DHT naturally implements the storage functions of a hash table, it is feasible to construct a DHT-based distributed storage system. Accordingly, there are several storage systems [33, 9] that are based on a DHT.

Current DHTs and P2P applications are mostly designed in the environment of the Internet where the network connections are variable and unpredictable. Thus all DHTs provide the capability to deal with a dynamic environment with machines joining and
disappearing frequently. Because DHT has other useful features, such as scalability, autonomy and load-balancing, the research presented in this thesis puts DHT into a more stable environment, computer clusters, in which all machines have persistent network connections. In order to construct a DHT-based storage system in a cluster environment, an important issue is its load-balancing. A global file sharing P2P system does not necessarily take the load-balancing into account, because the disappearance of some files in a highly dynamic environment is normal. Nonetheless, disappearance or unavailable files label a storage system unreliable. Therefore, monitoring the loads of machines in a DHT-based storage system is important for the whole system’s performance.

Before the discussion of the load-balancing of DHT-based storage systems in Chapter 2, it is worthwhile to introduce Chord and the Chord-based storage system in more detail.

1.2 Chord

As a DHT, Chord implements the most basic key-value pair insertion and key-value pair retrieval operations, that follow the conventional hash table. For example, a person Wang and her telephone number 13019801111 are going to be stored in a telephone book that is implemented using a conventional hash table. Wang is the key; and 13019801111 is the value associated with the key Wang. A conventional hash table uses a hash function to convert the key Wang to a hash number, which is then used as the index of a memory space in a group of memory spaces possessed by the hash table. In this way, the key-value pair — Wang-13019801111 is inserted in the hash table (telephone book) according to the produced index. This is the put operation of a conventional hash table. Correspondingly, given the key Wang, the associated value 13019801111 can be retrieved from the hash table (telephone book) according to the hashed index of the key Wang. This is the get operation of a conventional hash table. Chord is a distributed version of a conventional hash table; it distributes the storage space of the conventional hash table to a group of machines that are connected by a network. Corresponding to the put and get operations of a conventional hash table, Chord also provides two primitive operations, namely key-value pair insertion and key-value pair retrieval. The difference is that the hash number of a key-value pair does not label a memory space in the DHT. Instead, the hash number points to the machine where the key-value should be stored. In order to achieve these functions
of DHT, Chord organises all participating machines into the Chord Ring according to each machine’s identifier (ID).

### 1.2.1 Chord Ring

Chord has a circular identifier space called the Chord Ring, which contains all possible Chord IDs. Both participating machines and stored key-value pairs are labelled on the Chord Ring according to their identifiers that are called Chord IDs. Each participating machine has a corresponding node on the Chord Ring. In this thesis, machine refers to a practical computer and node refers to a logical representation of a machine on the Chord Ring. In each machine, there may be multiple nodes in some cases, such as the situation described in Section 2.3.1. The Chord ID of a node is called the Node ID, which can be produced in two ways:

1. A Node ID can be produced randomly using a random number generator. Different random generators produce different node distributions on the Chord Ring. Usually, a uniform random generator is used in order to produce an even distribution of nodes on the Chord Ring.

2. A cryptographic hash function is another method to produce a Node ID from a unique property of a participating machine, such as the machine’s IP address, port number or MAC address.

The original Chord paper [50] employs the machine’s IP address to produce a Node ID using the SHA-1 hash function [34]. For example, suppose a machine in Chord has IP address 130.192.1.232. Then the machine’s Node ID is 647046cf1a986527218a1f1feafcc920668735 from the hash function: $f_{SHA-1}(130.192.1.232)$ where $f_{SHA-1}(str)$ means the SHA-1 hash number of the string str.

There is also a point on the Chord Ring for each key-value pair according to its Chord ID. A key-value pair’s Chord ID is called the Data ID, which can be produced by the following two methods:

1. The Data ID of a key-value pair can be produced from the hash function applied to the key: $(f_{SHA-1}(key))$.

2. The Data ID of a key-value pair can alternatively be produced from the hash function applied to the value of $f_{SHA-1}(value)$. This is used if there is no key for the data to be stored in the DHT.
In the experiments presented in Section 4.6, the second method is used to produce Data IDs. In this way, Data ID and the value in the key-value pair construct a new key-value pair to be inserted into Chord. In the rest of this thesis, without contrary indication, the key-value pair denotes the Data ID and the value pair.

Using different hash functions in Chord, there are different Chord Rings. In the original Chord paper, SHA-1 is the hash function to produce both Node ID and Data ID, therefore, the ID space is: \[ ID_{\text{Chord,SHA-1}} = \{id | id \in [0, 2^{160}) \}, \] which represents the Chord Ring by concatenating the 0 and \( 2^{160} \). The number of possible Chord IDs is: \[ |ID_{\text{Chord,SHA-1}}| = 2^{160}. \] As an example, Figure 1.1 shows a simplified Chord Ring with a smaller ID space: \[ ID_{\text{Chord}} = \{id | id \in [0, 2^6) \}. \] In Figure 1.1, the solid squares represent the stored key-value pairs with their Data ID labelled inside of the Chord Ring; and the smaller circles denote the nodes with their Node ID labelled outside of the Chord Ring. Therefore, there are 9 nodes and 11 key-value pairs in this Chord Ring.

![Figure 1.1: A Chord Ring with capacity of \( 2^6 \), 9 nodes and 11 key-value pairs.](image)
1.2.2 Maintaining Chord

Given a Chord ID, the closest node whose Node ID is smaller than the given Chord ID is called the *predecessor* of the Chord ID. Correspondingly, the closest node whose Node ID is bigger than a given Chord ID is called the *successor* of the Chord ID. On the Chord Ring, both nodes and stored key-value pairs have their successor and predecessor according to their Node ID and Data ID. In Figure 1.1, the node \( Node_{32} \)’s predecessor is \( Node_{15} \); and its successor is \( Node_{41} \). The key-value pair \( (K - V)_{35} \)’s predecessor is \( Node_{32} \); and its successor is \( Node_{41} \). The Chord Ring is a logical structure for Chord; the practical linkages exist between nodes in the form of pointers to their successors and predecessors.

When a machine \( M_J \) joins a Chord Ring, it needs to know another machine that is already in the system. That machine is called \( M_J \)’s *bootstrap machine*, which notifies \( M_J \)’s predecessor and successor in the storage system according to \( M_J \)’s Node ID. Consequently, the *join* function determines \( M_J \)’s location on the Chord Ring using its Node ID, and notifies \( M_J \)’s successor and predecessor with its joining. Furthermore, in order to achieve a more efficient message routing, Chord introduces a *finger table* in each node, which starts to be populated with appropriate entries after the machine’s joining. Chord is designed for a highly dynamic environment, so each node’s successor, predecessor and the entries in its finger table are updated periodically. Because message routing in Chord is irrelevant to the storage load-balancing research in this thesis, the finger table will not be described in detail in this thesis. Details of the node joining and departure, message routing and the Chord Ring maintaining can be found in Chord’s original paper [50].

1.2.3 Key-value Pair Operations

Each node has a *responsible zone*, which is defined to be a series of contiguous Chord IDs on the Chord Ring. A node \( N \)’s responsible zone \( (RZ_N) \) is a set of Chord IDs between the Node ID of \( N \)’s predecessor exclusive \( (NID_{N,Pred}) \) and the Node ID of \( N \) inclusive \( (NID_N) \) with the consideration of Chord’s wrap around ID space with the modulus of \( 2^{160} \) using SHA-1:

\[
RZ_N = \begin{cases} 
\{id \mid id \in (NID_{N,Pred}, NID_N]\} & \text{if } NID_{N,Pred} \leq NID_N \\
\{id \mid id \in ((NID_{N,Pred}, 2^{160}) \cup [0, NID_N])\} & \text{if } NID_{N,Pred} > NID_N
\end{cases}
\]

(1.1)

The responsible zones of all nodes evidently cover the whole Chord Ring.
CHAPTER 1. INTRODUCTION

When a key-value pair \((K - V)\) is inserted into Chord from a participating machine \(M\), according to the corresponding Data ID \(id_{K - V}\), the key-value pair \(K - V\) has a position on the Chord Ring. Thereafter, the key-value pair is stored in the machine whose node’s responsible zone contains \(id_{K - V}\). The task of machine \(M\) is to find the successor of Data ID \(id_{K - V}\), where the key-value pair \(K - V\) will be stored. The machine \(M\) is known as \(K - V\)’s publishing machine. For example, when \((K - V)_{25}\) is inserted into the Chord Ring in Figure 1.1, its Data ID is 25 whose successor is \(Node_{32}\). Then \((K - V)_{25}\) is stored in \(Node_{32}\) because 25 is in the responsible zone of \(Node_{32}: (15, 32)\). As mentioned before, this is the process called key-value pair insertion.

In order to retrieve a key-value pair from Chord, the Data ID of the key-value pair must be known firstly. Then, following the same process of finding its Data ID’s successor, the Node storing the key-value pair is located. The key-value pair retrieval is accomplished by transferring from the node storing the key-value pair.

Chord functions as a storage system because it implements the two most primitive key-value pair operations: key-value pair insertion and key-value pair retrieval. However, if Chord is used to implement a storage system, there are more considerations that need to be taken into account, such as the operations for large files. These are described next.

1.3 Chord-based Storage System

In Dabek’s work [9], a DHT-based storage system is implemented using Chord. The main features of this system are described below, thereby forming a Chord-based storage system. In the rest of this thesis, storage systems implemented by any DHTs are called DHT-based storage systems.

1.3.1 File Structure

The files stored in the Chord-based storage system are sliced into a number of data blocks with predefined fixed block size. The structure of files is shown in Figure 1.2 which also illustrates three types of blocks in the storage system, namely inode block, indirect block and data block. Each file is broken into a number of data blocks which each have the same fixed block size. In the file structure, no blocks exceed the fixed block size. Therefore all file data blocks except the last block have the same size which is equal to the fixed block size. The last file data block may have a smaller
block size than others. For each data block, its Chord ID is called *Data Block ID*, which is produced from the data block’s content using the hash function SHA-1. An indirect block is filled with Data Block IDs. If the size of an indirect block reaches the fixed block size, a new indirect block is created to accept the rest of the Data Block IDs. This process repeats until all Data Block IDs are filled into indirect blocks. An indirect block’s Chord ID is called the *Indirect Block ID*, which is produced from the content of each indirect block using SHA-1. According to Figure 1.2, Indirect Block IDs are then put into the inode block of the file. For a file, there is only one inode block which contains the file name, the file size and a list of Indirect Block IDs. The inode block also has a Chord ID which is called the *Inode Block ID*. The fixed block size also limits the size of inode block and indirect block. Therefore, the maximum theoretical file size (*Max_Size_file*) that can be handled by the Chord-based storage system with the file structure in Figure 1.2 is:

\[
Max_{\text{Size}}_{\text{file}} = \left( \frac{\text{Fix}_\text{Size}_\text{block} - \text{Size}_{\text{file name}} - \text{Size}_{\text{file size}}}{\text{Size}_{\text{Chord ID}}} \right) \times \left( \frac{\text{Fix}_\text{Size}_\text{block}}{\text{Size}_{\text{Chord ID}}} \right) \times \text{Fix}_\text{Size}_\text{block}
\]

(1.2)

where *Fix_Size_block* is the fixed block size, *Size_file_name* and *Size_file_size* are the size of file name string and size of file size number, respectively, and *Size_Chord_ID* is the size of a Chord ID. All sizes are measured in bytes. All file blocks have the same fixed block size, which is set to 1 MB (Megabyte) in the Chord-based storage system which is used.
in the experiments of Section 4.4. At this point, in the Chord-based storage system with $\text{Fix\_Size\_block} = 1\text{MB}$, the maximum acceptable file size is $\approx 2.55\text{ PB}$ (Petabytes), given that $\text{Size\_file\_name} + \text{Size\_file\_size} \leq 260\text{ bytes}$ and $\text{Size\_Chord\_ID} = 20\text{ bytes}$ for Chord IDs that are produced by SHA-1.

1.3.2 File Operations

The Chord-based storage system provides two file operations: file insertion and file retrieval. When a file is inserted into the Chord-based storage system, it is firstly divided into data blocks which, then, are organised into the structure in Figure 1.2 including an inode block, a group of indirect blocks and the divided data blocks. Through a machine in the system, each file block is treated as a key-value pair which is inserted into the system using Chord’s key-value pair insertion operation. The file insertion operation is accomplished by successful key-value pair insertions of all file blocks. At the end, the file’s Inode Block ID is returned by the file insertion operation.

Before retrieving a file from the Chord-based storage system, the file’s Inode Block ID must be known firstly. Using the key-value pair retrieval operation in Chord, the file’s inode block can be retrieved from the storage system. Thereafter, the inode block contains all Indirect Block IDs, which, in turn, are used to retrieve all indirect blocks. All of a file’s Data Block IDs can be obtained from the indirect blocks. Finally, the file can be reconstructed by orderly concatenating all of the data blocks that are retrieved according to their Data Block IDs. In the Chord-based storage system, each file operation involves a number of key-value pair operations. If and only if all the involved key-value pair operations are successfully completed, the corresponding file operation can be labelled as a success. A predefined global timeout value is associated with each key-value pair operation in a Chord-based storage system, in order to prevent a long time waiting for a disappeared key-value pair operation request.

In the Chord-based storage system, there is no file deletion operation; it is instead substituted by a file expiration mechanism. Each stored key-value pair is given an expiry time, after which the key-value pair will be deleted in its stored machine. If a key-value pair’s publishing machine asks for an extension for the key-value pair, the stored machine resets the expiry time. The upper layer applications using the Chord-based storage system can also use a republish mechanism to keep wanted key-value pairs in the system.
1.4 Research Hypothesis

According to the description of the Chord-based storage system in Section 1.2 and Section 1.3, when a DHT, such as Chord, is used in a storage system, key-value pairs are stored in all participating machines using a hash function. Because of the DHT’s natural load-balancing, that will be introduced in Section 2.2, Chord has the capability to evenly distribute stored key-value pairs across all machines in the system without consideration of the key-value pair’s size. The experimental result in Section 5.3.1 indicates that a certain degree of balanced storage load can be achieved by Chord’s natural load-balancing. However, another experimental result in Section 5.3.3 shows that only 11.94% of storage space contributed by all machines in the system can be harnessed by the storage system, if there are no extra load-balancing approaches. This storage load issue motivates the research on an efficient storage load-balancing mechanism to make the Chord-based storage system more usable.

Conventional load-balancing approaches for large-scale systems, such as web systems, are not suitable for a DHT, because there are clearly defined servers and clients, between which the conventional load-balancers are situated [16]. The conventional load-balancers have the information of all servers to which a request can be allocated. However, in a DHT, there is no global view to allow allocation of a request to a proper machine. This is also the most challenging part of designing a load-balancing approach for DHTs.

This thesis concentrates on the improvement of load-balancing for a Chord-based storage system in a cluster environment. The hypothesis is that the Chord-based storage system can be deployed in a cluster environment, which causes load-balancing problems, and the proposed additional $P_{VNode}$ load-balancing approaches can improve the storage load-balancing.

1.5 Contribution

In the following, the main contributions of this research are listed:

A Survey of Load-balancing Approaches for DHTs

This research has investigated the load-balancing of four DHTs, namely Chord, CAN, Tapestry and Pastry, which have been used to implement a DHT-based storage system. Their natural load-balancing properties are extracted from an analysis and comparison of these four DHTs. Due to the inadequacy of their
natural load-balancing properties, there are a number of previously published extra load-balancing approaches that are categorised into three techniques: virtual node approach, network proximity and replication. These three load-balancing techniques for DHTs are discussed in Section 2.3. They have also inspired the design of the proposed \( P_{VNode} \) storage load-balancing approach described in Section 3.4.

**DHT-based Storage System in a Cluster Environment**

This research is looking for a possibility to deploy the DHT-based storage system into a cluster environment, in which machines have good and persistent network connections, in order to make the DHT-based storage system available to be used by applications such as those used in a data centre. Although DHTs are proposed in P2P systems to deal with the dynamic environment with frequent machine joining and leaving, DHTs are naturally appropriate to be used to implement a distributed storage system. On the one hand, a DHT naturally defines two primitive data operations, namely key-value pair insertion and key-value pair retrieval. On the other hand, DHT’s natural load-balancing is able to evenly distribute stored key-value pairs across all machines in the system. Therefore, it is promising to put a DHT-based storage system in a cluster environment in order to provide data centre services. The research in this thesis chooses Chord to implement the storage system.

**Proximity-aware Storage Load-balancing Approach**

A storage load-balancing approach, \( P_{VNode} \) is proposed by this research in Section 3.4, in order to improve the storage load-balancing of the Chord-based storage system. The \( P_{VNode} \) scheme takes the size of each stored key-value pair into consideration and provides a more uniform distribution of the storage utilisation across all participating machines than that provided by Chord’s natural load-balancing or the original virtual node load-balancing approach. Three important time points are proposed in this research to determine the storage status of the system. Due to the imbalanced storage utilisation of the original Chord-based storage system, before it becomes unreliable, a storage system can only harness 11.94\% of the whole storage space of the system, which is contributed to by all machine’s local Chord storage space. The \( P_{VNode} \) scheme increases the accessible percentage of the whole storage space to 61.82\% in the experiments conducted during the work reported here.
Hilbert Curve Approach

This research employs an abstraction called the Hilbert curve to reduce the cost of the proposed load-balancing approach. The Hilbert curve approach takes network proximity information into consideration and provides an appropriate criterion to the P_VNode approach’s virtual node migration operation. In order to accomplish this task, a Hilbert ID, a Hilbert table and the Hilbert table’s maintaining operations have been introduced in each node of the Chord-based storage system.

Experimental Methodology

The research in this thesis employs experiments in a practical environment as the evaluation method. Most existing DHTs and their load-balancing approaches are evaluated by analytical proof or by simulation. In this research, the Chord-based storage system is deployed in a cluster environment that consists of five teaching laboratories in the author’s school. Using this environment, Chord’s natural load-balancing, the original virtual node load-balancing approach and the proposed P_VNode load-balancing approach are compared. In the same experimental environment, besides the storage load-balancing, other aspects of the original Chord-based storage system, such as the CPU running time, the memory usage, the storage system’s operational capacity and the hotspot detection approach can be evaluated. In order to achieve these tasks, an experimental platform is developed for this research to perform the experiments as described in Section 4.4.3.

1.6 Thesis Structure

The remainder of the thesis is structured as follows.

Chapter 2

This chapter introduces three more DHTs, CAN, Tapestry and Pastry, besides Chord that has been described in Section 1.2, under the background of implementing a distributed storage system using the DHTs. The natural load-balancing properties of these four DHTs are surveyed and analysed. Because of limitation in their natural load-balancing, a number of extra load-balancing approaches have been proposed to improve each DHT’s natural load-balancing. All these extra load-balancing approaches are categorised into three techniques: virtual node approach, network proximity and replication. The research in the
remainder of the thesis combines these techniques to improve the storage load-balancing of the Chord-based storage system.

Chapter 3
This chapter analyses the requirements of the storage space usage and network utilisation of the storage load-balancing for a Chord-based storage system. The Hilbert curve approach is constructed to cluster the machines in the Chord-based storage system according to their network proximity information. Based on the requirement analysis and the Hilbert curve approach, a proximity-aware virtual node load-balancing approach, P_VNode, is proposed in Section 3.4. At the end of this chapter, a suggestion for a method for reducing hotspots in the system is provided.

Chapter 4
This chapter introduces the experimental evaluation method of this research to evaluate the Chord-based storage system in a practical environment. An experiment platform is developed to control the whole experimental environment that consists of five teaching laboratories and to instruct the experiment’s running. The experiment objectives, the experimental network environment and the setting of experiment machines are presented at the beginning which is then followed by the measurement metrics and the experimental scenarios.

Chapter 5
This chapter presents and discusses the experimental results of the Chord-based storage system in three parts. The first part examines the experimental results on the original Chord-based storage system. The second part experimental results concentrate on the improvements of the storage load-balancing when the proposed new storage load-balancing approach, P_VNode, is embedded in the Chord-based storage system. The last part experimental results investigate the network issues of the Chord-based storage system.

Chapter 6
All findings of this thesis are concluded in this chapter together with a critique of the research. Due to the limited time of the project, there are still some further works that would be worthwhile to investigate based on the work in this thesis. These future works are presented at the end of this chapter.
Chapter 2

Load-balancing in Distributed Hash Tables

2.1 Introduction

In a Chord-based storage system, as described in Section 1.2, key-value pairs are stored in participating machines according to their Data IDs that are produced by the cryptographic hash function, SHA-1. From the perspective of the load-balancing, therefore, the distribution of key-value pairs across all machines is affected by both the distribution of Data IDs of the key-value pairs and the distribution of Node IDs of Chord nodes on the Chord Ring, according to the description of key-value pair operations in Section 1.2.3. When the Node IDs are also produced by the SHA-1 hash function, the distribution of key-value pairs in all machines relies on the values produced by SHA-1. Stoica et al. [50] claim that Chord has the capability to allocate key-value pairs to machines in a uniform random pattern. This is the natural load-balancing provided by Chord. In order to investigate the load-balancing of a storage system based on a DHT, the research in this thesis starts from the natural load-balancing property provided by a number of DHTs: Chord, CAN, Tapestry and Pastry.

A survey and analysis of these four DHTs’ natural load-balancing is presented in Section 2.2, which also contains an introduction to the basic operations of the three latter DHTs in order to explain their natural load-balancing properties. The detailed introduction to Chord has already been provided in Section 1.2.

Due to limitations of the natural load-balancing, extra work has been undertaken
CHAPTER 2. LOAD-BALANCING IN DISTRIBUTED HASH TABLES

36
to improve the natural load-balancing property provided by such DHTs. These extra load-balancing approaches are classified into three techniques: virtual node approach, network proximity and replication, which are described in Section 2.3. The load-balancing approach proposed in Chapter 3 is inspired by these three design directions.

2.2 Natural Load-Balancing in Distributed Hash Tables

This section compares four DHTs: Chord, CAN, Tapestry and Pastry. Chord and CAN are based on consistent hashing [24], Tapestry and Pastry are based on the Plaxton mesh [38]. Most more recent DHTs [23, 49] are based on these four DHT designs. In the comparison, the natural load-balancing of the number of objects stored in nodes is analysed based on using these DHTs in a distributed storage system.

2.2.1 Consistent Hashing

The consistent hashing approach [24] was originally used to address the hotspot problem of the client-server model systems in large scale networks such as the World Wide Web system in the Internet. Between a number of servers and a much larger number of its clients, there is usually a caching layer to relieve the burden of swamped servers due to the large number of client requests. The consistent hashing approach is used to organise the machines in this caching layer to fulfill the requests from a large number of clients on behalf of servers. For example, in web service systems in the Internet, web pages are cached in cache machines of a middle layer between web servers and web clients in order to relieve the tremendous access pressure on the web servers from a large number of web page requests. The middle caching layer employs consistent hashing to target web pages from the independent cache machines in a distributed environment.

Consistent hashing is inspired by the conventional hash function by treating each cache machine as a bucket in a conventional hash function. Therefore, a client request is able to be targeted at a specific cache machine using a conventional hash function. However, cache machines are in a dynamic distributed environment where the changing (joining or leaving) of cache machines is allowed. Although changing of a bucket is easy for conventional hash functions to rearrange the stored objects in buckets within
a local environment, it would be too expensive to perform a similar rearrangement for new cache machines joining, or existing cache machines leaving, in a distributed environment. Due to the dynamism of the environment of the cache layer, clients may have different views of cache machines. The view, here, denotes a subset of all cache machines.

Accordingly, in order to adapt to the dynamic distributed environment, consistent hashing is designed to fulfil the following four properties: monotonicity, balance, load and spread.

- **Monotonicity** means that object migration from an old cache machine to a newly joined machine only happens in order to evenly distribute objects after joining of the new cache machine. Object migration from an old cache machine to another old cache machine is treated as unnecessary.

- The balance property follows the main property of a conventional hash function, namely even distribution of objects across all cache machines from a specific client view.

- The load is the maximum number of possible objects which may be stored in a cache machine $M_{i}$ from all client’s views. The load of each cache machine should be kept small for a good consistent hashing design, in order to prevent swamped cache machines.

- The spread is the number of cache machines required to store an object $O_{i}$, when $O_{i}$ is inserted into the cache layer by a certain number of clients from their different views. The spread should be small for a good consistent hashing design.

According to these four properties, monotonicity, balance, load and spread, consistent hashing makes maximum effort to preserve the uniform distribution of a conventional hash function for the cache layer in a dynamic distributed environment. When the number of cache machines changes, the stored objects in the remaining cache machines are roughly evenly distributed by object migration between a minimal number of cache machines. Therefore, the load-balancing achieved by consistent hashing is at all times close to a uniform distribution of the number of objects across cache machines.

Because of the adaption to the dynamic distributed environment, consistent hashing is employed by a number of designs for DHTs. In the following, two of these, Chord
and CAN are presented, together with their load-balancing properties.

### 2.2.1.1 Chord

Section 1.2 has already introduced Chord, the Chord Ring, key-value pair operations in Chord and the Chord-based storage system [9]. Therefore, this section concentrates on the natural load-balancing provided by Chord as a DHT. According to the key-value pair insertion operation, described in Section 1.2.3, the distribution of the key-value pairs in Chord nodes is heavily affected by two factors: the distribution of Node IDs and the distribution of Data IDs. The Chord nodes are allocated on the Chord Ring following their Node IDs that are produced from the cryptographic hash function SHA-1 (Section 1.2.1). Each key-value pair to be inserted into Chord has a Data ID that is also produced from the hash function SHA-1. Therefore, in order to make key-value pairs evenly distributed across all nodes, SHA-1 should keep its produced Chord IDs uniformly distributed on the Chord Ring. Because Chord mainly follows the consistent hashing approach, Rao et al. [41] show that the number of key-value pairs in all nodes has a $O(\log N)$ imbalance factor which means that the node holding the largest number has $O(\log N)$ times as many key-value pairs as the node holding the smallest number.

Besides the natural load-balancing inherited from consistent hashing, another simple load-balancing mechanism is provided in Chord, namely the virtual node approach. Each machine of the original Chord system has only one node on the Chord Ring. If the virtual node approach is employed, each machine in a system has multiple nodes, each of which is called a virtual node. On joining the Chord system, a machine is required to initialise a fixed number of virtual nodes for which the machine will be responsible. Each virtual node is responsible for a part of the Chord Ring in the manner described in Section 1.2.3. Figure 2.1 shows a system in which there are two virtual nodes in each machine in a small sized Chord Ring as described in Section 1.2.1. The Node ID of each virtual node in a machine is produced by applying the cryptographic hash function, SHA-1, on the virtual node’s identification information that is constructed by the machine’s IP address and its suffix, the virtual node’s index number in the machine. Therefore, even though the two virtual nodes are in the same machine, there is no relationship between their locations on the Chord Ring, as demonstrated by the machine-virtual node table on the lower right-hand side of Figure 2.1.

This load-balancing approach is based on the fact that both nodes and key-value pairs are allocated on the Chord Ring using a uniform Chord ID producer, such as a uniform random number producer. In this case, if there are more virtual nodes on the
CHAPTER 2. LOAD-BALANCING IN DISTRIBUTED HASH TABLES

Chord Ring, the size of the responsible zones of the nodes become closer to the average zone \( (2^m/N, \text{where } 2^m \text{ is the capacity of the Chord Ring and } N \text{ is the total number of nodes in the system}) \). As a result, a more uniform distribution of nodes on the Chord Ring is provided by the virtual node approach. This is a basic idea for improving load-balancing of DHTs based on consistent hashing. More detail of the usage of the virtual node load-balancing approach and its derivatives is given in Section 2.3.1.

2.2.1.2 Content-Addressable Network (CAN)

CAN is short for Content-Addressable Network [42], which is another DHT implementation based on consistent hashing. CAN provides three functions: key-value pair insertion, key-value pair lookup and key-value pair deletion.

Chord has the circular Chord Ring, on which all participating machines are allocated. Following a similar idea, CAN uses a d-dimensional Cartesian coordinate space on a d-torus as the logical structure on which to locate the participating machines. The d-dimensional Cartesian coordinate space on a d-torus is called the CAN space. Figure 2.2 illustrates the CAN space for a 2-dimensional Cartesian coordinate space on a
2-torus. Due to the 2-torus, edge C1C2 and edge C4C3 are colocated; at the same time, edge C1C4 and C2C3 are colocated, as shown by the dashed links.

![Diagram](image.png)

Figure 2.2: A CAN space of 2-dimensional Cartesian coordinate space on a 2-torus (d = 2).

In Chord, both nodes and key-value pairs are allocated a Chord ID which labels a position on the Chord Ring. CAN allocates each node and each key-value pair a point in the CAN space, so the point’s coordinate vector corresponds to the Chord ID in Chord. A node’s point is chosen randomly using a uniform random generator that ensures that the nodes are evenly distributed over the whole CAN space. According to the node’s point, the node is allocated a d-dimensional responsible zone in the CAN space. Figure 2.3 demonstrates the joining procedure for a new node N16 in the CAN space of Figure 2.1. The whole CAN space is divided into a number of node’s responsible zones which are labelled by the node’s name at the lower right of the zone in Figure 2.3. Point 16 is the random point of the new node N16, which falls into the responsible zone of node N15. Thereafter, half of node N15’s responsible zone is given to the newly joined node N16. At the same time, the “neighbours” of node N15 are shared to node N16, both of which adjust their neighbour tables. After N16’s joining, N15 has its new “neighbours”: N2, N6, N9 and N16. N16’s neighbour table includes N2, N6, N8 and N15, all of which are notified by the joining of N16, and then modify their neighbour tables. In this way, nodes are joined into the system with their allocated responsible zones in the CAN space. Due to the uniform random generator that is employed, the responsible zone of N1 has 4 times more possibilities to be allocated points than the responsible zone of N16 according to their areas in this 2-dimensional
example.

Figure 2.3: A CAN space of 2-dimensional Cartesian coordinate space on a 2-torus with 16 nodes and 15 key-value pairs (including KV1, KV2 and KV3).

Following a process similar to that of producing the Data ID for a key-value pair in Chord, each key-value pair is assigned a point by applying a uniform hash function on its key. Then the key-value pairs are stored by the node whose responsible zone contains their corresponding points. In Figure 2.3, the points of key-value pairs are presented by small circles. Therefore, key-value pairs: KV1, KV2 and KV3 should be stored in node N6.

The distribution of key-value pairs across all nodes still relies on the uniform random generator for nodes and the uniform hash function for key-value pairs. The uniformity of the distribution of key-value pairs in all nodes is ensured by the uniformity of node’s allocation in the CAN space and the uniformity of the distribution of key-value pairs’ points. This is the natural load-balancing that CAN is able to achieve. The number of key-value pairs in each node is proportional to the area of the node’s responsible zone; for example, the zone of N1 has 4 times more possibilities to be allocated the points of key-value pairs than zone of N16. In the worst case, if the smallest zone was always halved for N newly joined nodes, the largest responsible zone would be \((\log_2 N)\) times bigger than the smallest responsible zone. Therefore, the node with the most key-value pairs could have at most \((\log_2 N)\) more key-value pairs than the node with the least key-value pairs. Thus, the imbalance factor for CAN is \(O(\log_2 N)\).

A virtual node approach can be added to CAN in exactly the same way as for Chord.
2.2.2 Plaxton Mesh

The Plaxton mesh is another inspiration to implement DHTs, besides the consistent hashing approach. The Plaxton mesh is a data structure proposed by Plaxton et al. [38] in order to route messages and locate objects in a static distributed environment without consideration of node joining and leaving. A machine in a Plaxton mesh is called a node, and an object is any type of data stored in the nodes. The Plaxton mesh provides two operations: message routing and object locating.

Both node and object are allocated an ID which is called the Plaxton ID. A node’s Plaxton ID is specified by its Node ID, and an object’s Plaxton ID is called the Object ID, for convenience. The method of producing a Plaxton ID is similar to that of producing a Chord ID, as described in Section 1.2.1. However, in DHTs based on the Plaxton mesh, a Plaxton ID normally comes with a base $b$; e.g. one of the Plaxton IDs produced by SHA-1 (160-bit) hash function with $b = 4$ is 647046CF1A986527218A1F1B1FEAFCC920668735 which consists of 40 Hex digits, each of which can be represented by a 4-bit binary number. Therefore, both Node ID and Object ID are located in the same ID space. The message routes to its destination according to the destination’s Node ID and the Node IDs of the nodes on the routing path. Figure 2.4 shows the routing process from node with ID 348AE to node with ID E4601 in a Plaxton mesh having a 20-bit ID space with $b = 4$. For each message routing forward, the Plaxton ID has one more identical 4-bit suffix towards the destination Node ID, incrementally. In Figure 2.4, this sequence is 348AE $\Rightarrow$ 288B1 $\Rightarrow$ 08801 $\Rightarrow$ A9601 $\Rightarrow$ 74601 $\Rightarrow$ E4601. In order to achieve this routing, each node maintains a local neighbour map that records the nodes that share different numbers of suffixes with this node. The arrows in Figure 2.4 point to the entry nodes in each node’s neighbour map. The shaded nodes in the lower right of Figure 2.4 represent different level entries in node E4601’s neighbour map. The entries in level $l$ of E4601’s neighbour map share maximally $l - 1$ suffixes with node E4601. On the $l$-th digit, there are $2^b - 1$ different digits apart from E4601’s $l$-th digit. Therefore, the first level of E4601’s neighbour map has nodes with Node ID ending with 0 and from 2 to F, and the second level’s Node IDs ending from 11 to F1, and so on. In the Plaxton mesh, there may be multiple nodes which are qualified to be placed in the same location of a node’s neighbour map. For example, in Figure 2.4, there may be another node 24E42 that can also be placed at the location of node B5432 in node E4601’s neighbour map. For a neighbour map entry in level $l$, the possible choosing ID space is $2^{b_N - b \times l}$, where $b_N$ is the total number of bits in the Plaxton ID (in the case of the Plaxton mesh using SHA-1, $b_N = 160$). As
$l$ increases, the choosing ID space becomes smaller. In other words, there are more probabilities for a neighbour map entry to choose a node from multiple qualified nodes in lower levels than in higher levels. Eventually, some entries in higher levels of a node’s neighbour map cannot find appropriate nodes from the system. At this point, in each level of a node’s neighbour map, the maximum number of entries is $2^b - 1$. The maximum size of a node’s neighbour map is therefore $l \times (2^b - 1)$.

Figure 2.4: Routing from node 348AE to node E4601 in a Plaxton mesh and the neighbour map of node E4601.

When a node publishes a stored object to other nodes in a Plaxton mesh, the object is allocated an Object ID using an ID producing method similar to that of Chord’s key-value pair as described in Section 1.2.1. Therefore, an object’s ID and the object itself actually construct a key-value pair in the Plaxton mesh. The node routes a publishing message targeting the Object ID of the object to be published. Usually, there is not a node with the same Plaxton ID as the object, so the Plaxton mesh provides an extra global consistent algorithm to choose a node for the object’s publishing based on a global view of all nodes in the mesh. This node is called the object’s root node which keeps the information of the object and the information of the node actually storing
the object. The locating of an object follows the same routing process which targets the object’s Plaxton ID. When the locating message reaches the object’s root node, the node information for the target object can be retrieved.

Following the message routing and object locating functions provided by a Plaxton mesh, two DHTs have been developed, namely Tapestry and Pastry, which are described next.

2.2.2.1 Tapestry

Based on Plaxton mesh based message routing and object locating, Tapestry [58] is proposed with improved properties which tackle some limitations in a Plaxton mesh.

Firstly, as a DHT, Tapestry is designed to be deployed in a dynamic distributed environment with nodes joining and leaving without any global information for any nodes in the system. The Plaxton mesh assumes its deploying environment is static with a fixed number of nodes. Tapestry defines the node’s joining approach which gives Tapestry scalability in a dynamic environment. A gateway node $N_g$ that has already joined Tapestry is required by a new node $N_j$ wanting to join into Tapestry. Through $N_g$, the neighbour map of $N_j$ is populated by the entries of the neighbour map of the node at each hop which is in the routing path towards $N_j$’s Node ID. At the same time, the presence of $N_j$ in the system is filled in the neighbour map of the relevant nodes. After $N_j$ joins Tapestry, its neighbour map will be updated further when routing messages pass through or arrive at $N_j$.

Secondly, the global view based consistent root node choosing algorithm for object publishing in a Plaxton mesh is replaced by the surrogate routing approach in Tapestry. When Tapestry is confronted with the usual object publishing situation in which there is no node that has the same Plaxton ID as the Object ID of the object to publish, the surrogate routing approach defines a consistent alternative operation, such as choose the next lower or higher entry at the same level in the neighbour map. For example, according to Figure 2.4, if an object A5601’s publishing message reaches node E4601 after 3 hops from another node, at the fourth level of E4601’s neighbour map, there is no entry with 5 at its fourth suffix. If the surrogate routing was defined to find the next higher entry from the neighbour map, the node B9601 would be chosen as the fourth hop of A5601’s publishing message. Until a node $N_s$ (e.g. the node B9601) with empty entry at the corresponding level of its neighbour map (e.g. the fifth level of B9601’s neighbour map) is reached, the node $N_s$ that is called the surrogate node of object A5601 replaces the role of the root node in the Plaxton mesh to store object
In Tapestry, an object’s storage information is cached by all nodes along the routing path of its publishing message. However, because Tapestry is deployed in a dynamic environment, the adding of new nodes may change the surrogate nodes which record objects’ storage information. Two methods are proposed to solve this problem in [58]. The first solution is that the node where the surrogate routing begins has responsibility to take action on the change of a surrogate routing. In the object A5601’s publishing example, node E4601 would have responsibility to notify the owner node of A5601 to republish the object A5601, if a newly joined node 36601 has been filled in the fifth level of node E4601’s neighbour map. Then node 36601 is the new surrogate node for object A5601. Node E4601 also has to delete all stale entries of object A5601’s previous publishing. Another solution is to assign each object’s storage information an expiration time, after which the object’s storage information is removed and the object is then republished by its owner node.

Finally, Tapestry has an unsatisfactory natural load-balancing according to the description of the object publishing operation provided by Tapestry based on the Plaxton mesh. The object’s storage information that is published in Tapestry is a soft-state that contains the object’s location information (its owner node) instead of the object itself. This is satisfactory for a file sharing system, in which all nodes share the files in their local storage to other nodes in the system. However, for a distributed storage system, there is no mechanism to distribute files across all nodes. When a file is inserted into the system, Tapestry does not define any operation to choose a node to store this file. If Object ID and the object itself construct a key-value pair which replaces the soft-state (a pointer to the object’s owner node to be stored in the surrogate node), Tapestry implements a key-value pair storage system similar to Chord. By using this method, Tapestry is able to be employed to implement a DHT-based storage system. Because the Node IDs and Object IDs are both uniformly distributed in the ID space of Tapestry, the imbalance factor should be the same as Chord, i.e. $O(\log N)$. Nevertheless, the introduction of object replicas in the nodes along the path of the object’s publishing message makes some nodes along the path of the object’s publishing message more possibilities to store more objects, which is affected by the storage system users’ behaviour. If a node $N_o$ is the place from where more object publishings are initiated than from other nodes, the nodes with similar Node IDs in Tapestry’s ID space would store the replicas of these larger number of published objects. Therefore, when Tapestry is employed in a storage system, it is not only vulnerable to the presence of malicious object insertion operations from a single node, but its storage load-balancing is also affected by the location of users’
insertion operations. In order to approach an $O(\log N)$ imbalance factor, the locations from which users insert objects have to be uniformly distributed in Tapestry’s ID space, which overly restricts the users.

2.2.2.2 Pastry

Pastry [45] is another DHT based on the Plaxton mesh’s routing mechanism. In order to be deployed in a dynamic distributed environment, like Tapestry, Pastry also defines node join and node leaving operations. However, there are some differences between Tapestry and Pastry. In the paper describing Pastry [45], the message routing is based on prefix matching of IDs, which is replaced by suffix matching of IDs in this section in order to be consistent with the earlier descriptions of Tapestry and the Plaxton mesh.

Besides a neighbour map that is inherited from the Plaxton mesh for message routing, two extra tables, a neighbourhood set and a leaf set are maintained in each node in Pastry. For a clearer presentation, the neighbour map is called the routing table in Pastry. The neighbourhood set records the network information (such as IP address) of the closest nodes to the current node according to some proximity metric. The leaf set contains the network information of nodes with numerically closest Node ID to the current node’s ID in the ID space of Pastry. Global system parameters $M$ and $L$ denote the capacity of the neighbourhood set and the leaf set, respectively. Because Pastry has a circular ID space, the distance between two Node IDs is evaluated based on the distance modulo the size of that ID space. For instance, if SHA-1 is employed to produce IDs for Pastry, the distance between node 0 and node $FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF$ is 1 instead of $2^{160} - 1$. The introduction of the neighbourhood set means that Pastry takes proximity information into consideration for an efficient message routing. More details of the proximity information for load-balancing approaches of DHT will be discussed in Section 2.3.2. Since the two extra tables have to be maintained in each node, more network traffic is required for maintaining messages for the neighbourhood set and the leaf set.

In order to publish an object avoiding the situation of no matching node with the same ID as the object’s ID, as described on Page 43, Pastry uses the leaf set to replace the surrogate routing in Tapestry. When the basic Plaxton routing for object $O_t$ is stopped by empty entries at a specific level of the routing table at a node $N_s$, the nodes in $N_s$’s leaf set are searched for the node with a numerically closer ID that shares a suffix with $O_t$’s ID at least as long as $N_s$’s ID. In this way, $O_t$’s publishing message is
routed to the node \( N_o \) that has the numerically closest ID to \( O_t \)’s ID in Pastry’s circular ID space. Therefore, the object’s storage information is stored in \( N_o \) which is called the storage node for \( O_t \). Pastry implements a different replication mechanism from that of Tapestry, which replicates objects in the nodes along their publishing message’s routing path. Pastry replicates object \( O_t \) to \( k \) number of nodes in node \( N_o \)’s leaf set, given that \( k \leq L \). So the replicas of \( O_t \) are stored by nodes ‘locally’ close to \( O_t \)’s storage node \( N_o \) in Pastry’s ID space.

Since Pastry has a circular ID space and uses a local replication approach, Pastry has a better natural storage load-balancing than that of Tapestry which is heavily affected by users’ publishing operations. The soft-state is also stored when objects are published in Pastry. If soft-state was discarded for deploying Pastry in a storage system, Pastry would have a similar storage load-balancing property to that of Tapestry, because they are both constructed based on the message routing mechanism of a Plaxton mesh. However, because the distributions of both node and object are uniform, and objects are associated with nodes according to closeness in the circular ID space, Pastry should have a similar imbalance factor, \( O(\log N) \), of the number of stored objects in nodes to that of Chord which also has a circular ID space. Furthermore, the replicas placed in the ‘local’ nodes of an object’s storage node ensure that the load-balancing of the number of objects in nodes would not be affected by users’ operations, even if there were malicious object publishing operations from the same node. In the design, both Tapestry and Pastry provide replication mechanisms which can be employed to effect network load-balancing. More details of replication will be presented in Section 2.3.3.

### 2.2.3 Comparison

This section has introduced four DHTs that are based on consistent hashing and the Plaxton mesh. Because this thesis focuses on the storage load-balancing of a DHT-based storage system, the storage load-balancing of these four DHTs have been discussed during the presentation. For comparison, Table 2.1 lists these four DHTs’ system structure, routing hops and imbalance factor. In Table 2.1, \( N \) is the number of nodes in the system, and \( b \) for Tapestry and Pastry is the base that is defined in the Plaxton mesh. The imbalance factor of Tapestry is presented as ‘Not Available’, because its imbalance factor is heavily affected by the location of users’ insertion operations.

All these four DHTs’ object storage pattern can be specified by three functions:

\[
f_N : N_s \rightarrow ID_N \mid ID_N \in ID_{space}
\]  

(2.1)
DHT | System Structure | Routing Hops | Imbalance Factor
---|---|---|---
Chord | Chord Ring | $O(\log_2 N)$ | $O(\log N)$
CAN | d-dimensional Cartesian coordinate space on a d-torus | $O(N^{1/4})$ | $O(\log_2 N)$
Tapestry | Plaxton mesh with plain ID space | $O(\log_b N)$ | N/A
Pastry | Plaxton mesh with circular ID space | $O(\log_{2b} N)$ | $O(\log N)$

Table 2.1: Comparison of the four DHTs.

\[ f_O : O_s \rightarrow ID_O | ID_O \in ID_{space} \]  \hspace{1cm} (2.2)
\[ f_{\text{store}} : (ID_N, ID_O) \rightarrow \{\text{to\_store}, \text{un\_store}\} | ID_N \in ID_{space} \text{ and } ID_O \in ID_{space} \]  \hspace{1cm} (2.3)

DHTs employ various specific functions $f_N$ and $f_O$ which convert a node’s information and an object’s information to node’s ID and object’s ID, respectively. In this section, the data stored in DHTs, such as key-value pairs in Chord, are all generally called objects for convenience. $N_s$ is the unique information of the node’s settled machine, including the machine’s IP address, Hardware address, Hardware identifier or any combinations of these. If a uniform random number generator is employed as $f_N$, the set $N_s$ is empty. $O_s$ is object’s information which contains the object’s semantic information, such as a key-value pair’s key or the name of an object. $ID_N$ and $ID_O$ are outputs of $f_N$ and $f_O$, the Node ID and the Object ID in the DHT’s ID space, $ID_{space}$. In the four DHTs’ description, cryptographic hash function, uniform hash function and uniform random number generator are mentioned all around. Although these are different functions, actually, what these DHTs require is only one, a function that is able to output uniformly distributed values in the ID space of a DHT. This is the requirement of the function $f_N$ and $f_O$. As for the third function $f_{\text{store}}$, this is defined by DHTs according to their system structure. Chord defines that an object is stored in the object’s successor node according to the description in Section 1.2.3. Therefore if the object $ID_O$’s successor is the node $ID_N$, function $f_{\text{store}}$ has output value $\text{to\_store}$, otherwise the output value $\text{un\_store}$ is obtained. CAN’s $f_{\text{store}}$ is defined by the existence of the point of the object $ID_O$ in node $ID_N$’s responsible zone. The function $f_{\text{store}}$ of Tapestry and Pastry is defined by the numerically closest distance from the object $ID_O$ to the node $ID_N$. Due to the similarity of the first two functions ($f_N$ and $f_O$) in these four DHTs, a similar imbalance factor for the number of objects in nodes.
is obtained. However, the third function $f_{store}$ heavily depends on the system structure of each DHT.

According to Table 2.1, although these four DHTs have distinct ID spaces, Chord, CAN and Pastry have similar wrap around ID spaces, which means that there are no boundaries in these three DHT’s ID spaces. Tapestry, however, is constructed on a Plaxton mesh with a one-dimensional plain ID space which has two boundaries at the beginning and at the end. These two boundaries make the two nodes on each side of the boundary have different opportunities to be used to store objects. Because of Tapestry’s ID space, it is hard for Tapestry to reach the $O(\log N)$ imbalance factor. Therefore, ID space is actually the central issue affecting these DHT’s natural load-balancing of the number of objects in nodes.

Although these four DHTs have similar natural load-balancing, Chord is chosen in the research of this thesis because of its simplicity and provable correctness. Compared with Tapestry and Pastry, which are based on the Plaxton mesh, Chord and CAN based on consistent hashing are more appropriate to be deployed in a storage system, because the design of Tapestry and Pastry suggests a soft-state publishing mechanism to share objects stored in all nodes in their systems. Tapestry and Pastry are designed for a file or information sharing system where distributing objects to all nodes is separated from these two systems. The users of Tapestry and Pastry are supposed to join in them with objects to share, so these two DHTs provide capability to share objects to other users in the same system. In contrast, Chord and CAN provide both object insertion operation and object retrieval operation in their designs.

Moreover, even the soft-state is discarded; as described on Page 45, Tapestry still has a vulnerable load-balancing which is affected by the location of user’s object publishing. As for Pastry, three tables to be maintained in each node make Pastry more complex than Chord which has one table and two points for each node. According to the experimental results for Chord presented in Section 5.2.2, maintaining messages produce a small part of the network traffic during the system’s running. Due to the availability of the Chord-based storage system, Chord is chosen as the base for this research.

Three of these DHTs, Chord, CAN and Pastry have similar imbalance factors, which indicates that the node with the largest number of objects has up to $\log N$ times more objects than the node with the least number of objects, with high probability. This natural load-balancing of the number of objects in nodes is ensured by the first
two uniform functions: $f_X$ and $f_O$. An experiment performed on the original Chord-based storage system is described in Section 5.3.1. The result of this experiment’s beginning indicates that a certain uniformity of node’s storage utilisation is provided by Chord’s natural load-balancing. Therefore, due to the natural load-balancing provided by DHTs such as Chord, CAN and Pastry, some DHTs naturally have capability to evenly distribute objects across all nodes in the system, which is the motivation to use a DHT to implement a distributed storage system. The natural load-balancing property of a DHT has inspired the research in this thesis to look for more possibilities to implement a DHT-based storage system by further improving the storage system’s load-balancing.

Notwithstanding that natural load-balancing is provided by some DHTs, it is still not enough for a storage system. According to consistent hashing’s design purpose that is to construct a cache layer to solve the hotspot server problem in web systems, what it counts is the number of hits on each cache machine from a large number of web object requests. So uniform distribution of the number of cached web objects in cache machines is important, but this does not imply that the used storage space in cache machines is also uniformly distributed. Furthermore, the natural load-balancing of DHTs only provides an acceptable method to distribute objects to nodes. There is no operation defined to address the storage overloaded nodes after a storage overload has already happened. Therefore, an extra load-balancing approach is required when a DHT-based storage system is constructed. Possible approaches for this are discussed in the following section.

### 2.3 Extra Load-balancing Approaches

Inspired by the natural load-balancing property that is shared by the four DHTs discussed in Section 2.2, it is promising to implement a distributed storage system using a DHT. However, the natural load-balancing is not sufficient for a storage system. Firstly, natural load-balancing only balances the number of objects across all nodes in the system, without any consideration of variance in the size of objects. In a storage system, a crucial aspect is the distribution of each machine’s used storage space that is contributed to by the different sized objects stored in it. Secondly, relying on natural load-balancing is a static method to achieve storage load-balancing in a DHT-based storage system. Even though a more evenly distributed set of objects can be ensured by natural load-balancing, there is still no guarantee that no storage overloads will arise.
A more dynamic storage load-balancing approach is required to deal with the situation where a storage overload happens. Finally, in a distributed environment, a DHT-based storage system requires more considerations, such as the network issue, beyond the storage load-balancing that natural load-balancing concentrates on.

Accordingly, in order to implement a DHT-based storage system with improved load-balancing based on any DHT’s natural load-balancing, this section investigates the existing load-balancing approaches proposed for DHTs, which are categorised into three groups: virtual node approach (Section 2.3.1), network proximity (Section 2.3.2) and replication (Section 2.3.3). These approaches have been briefly encountered through Section 2.2.

2.3.1 Virtual Node Approach

Virtual nodes were first mentioned on Page 38 in the context of Chord. The paper on Chord [50] introduces the virtual node load-balancing approach which is based on a theorem that is proven in the paper on consistent hashing [24]. A part of the theorem states that “each node in Chord is responsible for at most \((1 + \epsilon)K/N\) identifiers on the Chord Ring and \(\epsilon = O(\log N)\)” [50]. If there are \(\Omega(\log N)\) number of virtual nodes in each machine, \(\epsilon\) can be reduced to an arbitrarily small value. This is the theoretical basis of applying virtual nodes to Chord in order to achieve a more uniform distribution of nodes’ responsible zones on the Chord Ring. Each virtual node stores the objects with their Object IDs that fall into the node’s responsible zone, according to the description in Section 1.2.1. This theoretical basis ensures a more uniform distribution of the number of stored objects in all virtual nodes, because objects’ Object IDs are produced by the uniform hash function, SHA-1, which means that a larger responsible zone has more probability that Object IDs will fall into it. Although the virtual node load-balancing approach is proposed for Chord, it can be applied on all other DHTs whose object storage pattern can be summarised by the three functions, Function 2.1, Function 2.2 and Function 2.3 introduced in Section 2.2.

Following the natural load-balancing of DHTs, the virtual node approach is still based on the theorem that ensures more uniform distribution of the number of objects in all virtual nodes. Although the imbalance factor can be reduced by the virtual node approach, it cannot ensure that all machines have exactly the same storage space usage. Therefore, in a system with the virtual node approach, there are still some machines that may be overloaded in their storage space earlier than other machines. More operations are required by the virtual node approach to deal with storage
overloaded machines. Rao et al. [41] propose a virtual node migration mechanism based on three transfer schemes: one-to-one, one-to-many, and many-to-many, each of which indicates the number of lightly loaded machines and the number of heavily loaded machines, respectively. Their idea solves the storage load-balancing problem by migrating a virtual node with its stored objects from its heavily loaded machine to a lightly loaded machine without making the lightly loaded machine overloaded. When a virtual node migration process involves either multiple lightly loaded machines or multiple heavily loaded machines, the system should have a machine that chooses an appropriate source-to-target machine-pair from all relevant machines to accomplish the virtual node migration. Accordingly, in order to collect lightly loaded machines and heavily loaded machines together, the virtual node approach introduces a number of ‘directories’ that are stored in Chord as an object. All virtual nodes report their storage load information to the ‘directory’ that is assigned when the node joins the system. The migration operations are performed within each ‘directory’ across the nodes reporting to it. In order to embed the virtual node approach in a dynamic system, Godfrey et al. [18] employ the many-to-many scheme in their virtual node load-balancing approach. Instead of assigning a fixed ‘directory’ to each virtual node, virtual nodes in a ‘directory’ randomly choose another ‘directory’ after each periodically scheduled virtual node transfer that may include a set of virtual node migrations. Several load-balancing approaches for Chord [6, 48] are based on these two virtual node approaches. Based on the virtual node approach, there are also a number of derivatives. Karger and Ruhl [25] design a special virtual node system in which each machine active only one of its virtual nodes. In their system, each machine has a fixed number of virtual nodes, each of which is allocated on an independent Chord Ring. At a given time point, a machine uses only one virtual node on a specific Chord Ring from all others. Due to the existence of virtual nodes with large size, Yang and Chen [56] propose a virtual node splitting algorithm to solve a single virtual node overload. The original virtual node load-balancing approach in a Chord-based storage system is evaluated in an experiment described in Chapter 4. The experimental result, presented in Section 5.3.2, demonstrates that the virtual node load-balancing approach improves the storage load-balancing of the Chord-based storage system.

Although the virtual node approach provides improvements to Chord, the introduction of extra virtual nodes in each machine incurs extra storage requirements for system data structures, such as successor, predecessor and finger tables. Maintaining
these system data structures for the virtual nodes in each machine requires more operations that generate more network traffic than that in the system with only one node in each machine. Furthermore, virtual node migration between machines is an expensive operation that may produce a large amount of network traffic. In particular, if the migration operation happens between two machines with a long distance connection from each other, a long time wait would ensue before its completion.

2.3.2 Network Proximity

A DHT is a distributed system with participating machines that are connected through networks. Because the participating machines have heterogeneous network environments, some pairs of machines are well connected, whereas others may be connected through a large number of networks that cause inefficient data transmission between them. Therefore, exploiting the network proximity among participating machines can be expected to improve the efficiency of DHT operations and its load-balancing approach’s operations, such as the virtual node migration mentioned in Section 2.3.1. Machines’ proximity information helps a node to distinguish between ‘near’ nodes and ‘distant’ nodes in a DHT. Such a scheme has already been introduced in the description of Pastry (on Page 46).

Castro et al. [5] categorise the application of network proximity information in DHTs in three distinct ways: system structure construction, proximity routing and proximity neighbour selection.

Firstly, network proximity information can be considered when a node joins a DHT, which means that the DHT’s system structure reflects the network proximity information. For example, when a node joins into CAN according to the joining process described in Section 2.2.1.2, in order to make CAN space contain proximity information, the point of the node can be combined with proximity information, such as a network proximity scalar (described on Page 54). However, this leads to a skewed distribution of the nodes in CAN space, which breaks the natural load balancing property that is able to distribute objects evenly to all nodes.

Secondly, network proximity information is used to assist the DHT’s routing mechanism by choosing a ‘near’ node for the next hop. By using this greedy ‘nearest’ hop routing approach, the DHT pursues a more efficient routing mechanism. Proximity routing requires that multiple candidate nodes are available at each hop of the DHT’s routing mechanism. CAN, Tapestry and Pastry fulfil this requirement. Proximity routing is easy to implement because candidate nodes exist in the DHT’s routing data
structure, such as the neighbour table of CAN, Tapestry and Pastry, in order to obtain proximity information. Given a node $N_c$ and a set of nodes: $N_1, N_2, \ldots, N_x$, that are $x$ candidate nodes of $N_c$’s next routing hop, proximity routing can be implemented by comparing the network proximity information from $N_c$ to each of its next routing candidate ($N_c$ to $N_1$, $N_c$ to $N_2$, ..., $N_c$ to $N_x$).

Finally, proximity neighbour selection is the most useful application that finds ‘near’ nodes for a specific node, which is harder to accomplish than proximity routing, because this is a searching operation that may cross all nodes in the system. At this point, a more sophisticated mechanism is required to perform proximity neighbour selection whose result can be employed by a load-balancing approach for DHTs.

The network proximity information discussed in this section defines the network distance between two nodes according to a specific metric. The network distance between two nodes could be measured by any network proximity scalar, such as Round-Trip-Time (RTT) or IP routing hops, which outline the network distance in terms of time and intermediate points between the two nodes, respectively. However, direct measurement between any two nodes is not feasible in a DHT that is deployed in a distributed environment with a limited view of other nodes for each node in the system. Furthermore, the measurement of a simple network proximity scalar is a time consuming operation that prevents a quick result from being obtained. In this case, these simple network proximity scalars are not a feasible method to provide DHTs with any two nodes’ network ‘distance’, especially when the network distances are frequently required. Therefore, a more efficient method to produce network proximity information is required by DHTs.

Compared with the simple network proximity scalars, the landmark machine approach is a more appropriate, systematic method to provide network proximity information efficiently. A number of systems [43, 59, 47, 48] employ some kind of landmark machine approach to select ‘near’ nodes. In the system, a number of special machines are assigned as landmark machines that are known by all nodes. A node $No$’s landmark vector, $(LM_1, LM_2, \ldots, LM_m)$, is defined by a list of simple network proximity scalars to all $m$ landmark machines; for example, $LM_i$ is the RTT from the current node $No$ to the landmark machine $i$, if RTT is chosen as the simple network proximity scalar. Therefore, the ‘distance’ between two nodes is evaluated by the Euclidean distance in a $m$-dimensional Cartesian space between the landmark vectors of
two nodes, \(NC\) and \(ND\):

\[
\text{Network\_Distance} = \sqrt{\sum_{i=1}^{m} (LC_i - LD_i)^2}
\]  

(2.4)

where \(LC_i\) and \(LD_i\) \((1 \leq i \leq m)\) are the simple network proximity scalars from \(NC\) and \(ND\) to landmark machine \(i\), respectively, and there are \(m\) landmark machines. The landmark machine approach is able to evaluate the network ‘distance’ between two nodes if the two node’s landmark vectors are provided.

Because a multi-dimensional landmark vector is still hard to be directly employed by the load-balancing approach of DHTs, dimension reduction is required. Inspired by the landmark machine approach, a network coordinate system explores the network proximity information by putting nodes in a 2-dimensional Cartesian space. An \(m\)-dimensional landmark vector is reduced to a 2-dimensional point that is called a network coordinate. Using landmark machines, Ng and Zhang \([35]\) create a network coordinate system called Global Network Positioning (GNP), which transforms the node’s landmark vector into a point in a 2-dimensional space. Based on GNP, Dabek et al. \([8]\) construct a decentralised network coordinate system called Vivaldi by replacing the fixed landmark machines using randomly chosen other nodes in the system. Duan et al. \([12]\) employ network coordinates produced by Vivaldi to cluster ‘near’ nodes in a separated 2-dimensional space from the ID space of Chord, the Chord Ring. In this way, nodes in Chord with their clustering approach are able to find their ‘near’ nodes in the 2-dimensional space of network coordinates.

Duan et al.’s proximity neighbour selection approach for Chord \([9]\) shows a mismatch of the dimensions between the Chord’s system structure (1-dimensional Chord Ring) and network proximity information (2-dimensional network coordinate). Therefore a separated space is required for their proximity neighbour selection approach. If landmark vector or network coordinate is embedded in CAN, which has a multi-dimensional system structure, CAN space, a separated space is not necessary. Xu et al. \([54]\) propose a proximity neighbour selection approach that stores the multi-dimensional network proximity information, i.e. the landmark vector, in the same ID space of CAN as a soft-state. This means that the proximity information can be inserted or retrieved as normal objects. Therefore, the crucial issue to apply network proximity information in Chord is to match its dimension to the ID space of Chord. Section 3.3 will discuss a Hilbert table approach, which is used in this research to cluster the ‘near’ nodes in the Chord Ring for the storage load-balancing approach.
2.3.3 Replication

The storage load is not the only aspect that should be considered by a load-balancing approach for a DHT-based storage system. The natural load-balancing property of DHTs motivates the implementation of a storage system using a DHT, because DHTs are able to evenly distribute objects across all machines in the system, as described in Section 2.2.3. Although each machine’s storage load is important to a storage system, network issues are also crucial to the storage system in a distributed environment. Taking network proximity information into consideration is an initial attempt to consider network issues in any DHT’s storage load-balancing approach. However, due to the existence of popular objects in a DHT-based storage system, the retrieval requests for these may overwhelm a small number of machines, because each machine has a finite network capacity. The small number of machines that are overwhelmed by a large number of requests are called the hotspots in a storage system (These will be formally defined in Section 3.2.2). This is the place where replication is required to diminish the number of hotspots in the system. Section 2.2 has already mentioned replication in the context of Tapestry (on Page 45) and Pastry (Page 47). An analysis of the relationship between storage load-balancing and network issues will be presented in Section 3.2. This section focuses on prior replication approaches for the load-balancing of DHTs.

In order to increase an object’s availability, all four DHTs that are discussed in Section 2.2 suggest making replicas of the objects that are stored in their systems. This is the initial motivation for applying replication in DHTs. Tapestry [58] places replicas along the routing path of the object’s publishing message. Due to the fact that its ID space is not wraparound, the replicated objects break the natural load-balancing, according to the discussion in Section 2.2.2.1. In Chord [50], the objects are specifically key-value pairs. The replication of key-value pairs can be performed from two different ends: the node that stored the key-value pair and the node that inserted the key-value pair. When a key-value pair is inserted in Chord, an extra index is concatenated as a suffix of its key, which produces a number of different but related keys for the key-value pairs. By using these processed keys, the key-value pair is inserted in multiple different nodes, in order to provide more availability. Another replication approach for Chord is to make the node transfer the key-value pair that is stored in it to be replicated to a fixed number of the node’s direct and indirect consecutive successors. This approach is able to reduce the burden of a node due to a large number of retrieval requests by redirecting these requests to the node’s direct and indirect successors with replicas of the popular key-value pair. CAN [42] places replicas of an
object in the nodes surrounding the node that stored the object. A replication ‘shield’ is constructed around the node preventing a large number of requests that could force the node to be a hotspot in the system. Pastry replicates the objects stored in a node to all nodes in its leaf set, in which all nodes have the numerically closest ID to the node’s ID. Therefore, in the ID spaces of Chord, CAN and Pastry, they all place replicas of an object in nodes that are adjacent to the node storing the object to be replicated. However, these DHTs’ proposals for replication do not take the cost of replication into consideration. Although applying replication is motivated by an object’s availability in the DHT, a replication approach with a more sophisticated design can also solve the hotspot problem.

A number of load-balancing systems [20, 55, 44, 53, 36] for DHTs have been proposed using replication. According to these works, a systematic replication approach to reduce hotspots in DHT has to complete the following tasks:

**Hotspot Detection**
This is the first step to trigger the replication approach to replicate a specific object stored in a node, because it is not necessary to apply replication to all objects stored in a DHT. In the previously published replication load-balancing approaches, a certain criterion, such as the request rate, is set up to indicate an overwhelmed node in their simulation experiments.

**Replication Insertion**
The replication insertion determines the number of replicas of the original object and their locations in the system.

**Replication Routing**
When there are a number of replicas of the same object in the system, choosing an appropriate replica is the central target of the replication routing operation. Replication routing is actually an extension to the DHT’s original object retrieval operation, the only difference is that replication routing reaches the nodes with an object’s replicas instead of the node that stored the original object.

**Replication Deletion**
Some replicas of an object may not be requested after they have been inserted into the system, in which case replication deletion is an important operation to reduce the number of replications in order to release the used storage space. For a time-varying hotspot that merely affects the system for a specific time duration,
all replicas of the time-varying popular object are no longer required after the
time-varying hotspot has disappeared from the system. This situation should
also be considered by the replication deletion task in order to save the storage
space from unnecessary redundant replicas. Two methods are usually involved
in this task: a globally consistent time-out mechanism and monitoring of replicas
from the node storing the original copy.

In the previous published replication load-balancing systems, there are still some
disadvantages that require improvements before embedding them in a DHT-based stor-
age system to reduce the hotspots.

Firstly, a simple hotspot detection method is defined in these replication load-
balancing systems, because the hotspot detection is a system dependent operation
that is affected by the underlying operating system. Because the evaluation method
for these replication load-balancing systems entails simulation experiments, a simple
hotspot detection based on a set of criteria is enough. When a DHT-based storage sys-
tem is implemented in a specific practical environment, more investigation of hotspot
detection is required. The research in this thesis has completed this task by producing
a hotspot detection approach that is described in Section 5.4.2 on Page 142, accord-
ing to the deployment environment of the Chord-based storage system as described in
Section 4.4.

Secondly, most prior replication-based load-balancing systems have no consider-
ation of replication cost. Similar to virtual node migration in a virtual node load-
balancing approach, the distribution of the original object to make its replicas in other
nodes produces network traffic demands. Therefore, in this case, network proximity
information can make the replication operation more efficient.

Finally, while a time-varying hotspot is addressed by a replication-based load-
balancing approach, it would also be worthwhile for a replication load-balancing ap-
proach to perceive the location-varying property of hotspots. The popular objects are
not only related to time, but are also related to the location, because the popularity
of an object is culture-related. An object that is popular in one society’s culture may
not be that popular in another. Therefore, placing more replicas of an object ‘near’
to locations where the object is most popular will improve the retrieval efficiency of
a DHT-based storage system, since objects are evenly distributed across all nodes in
DHTs without regard to network proximity information, according to the natural load-
balancing provided by DHTs.

Based on an analysis of existing replication-based load-balancing approaches, in
Section 3.5, this research suggests a design for a replication-based approach with hot-zone targeting, in order to reduce hotspots in a Chord-based storage system.

### 2.4 Summary

This chapter has investigated the natural load-balancing of four DHTs, Chord, CAN, Tapestry and Pastry. Chord and CAN are based on consistent hashing, Tapestry and Pastry are based on the Plaxton mesh. Although they implement the DHT in four different ways, the object storage pattern of these four DHTs can be summarised by three functions, Function 2.1, Function 2.2 and Function 2.3, introduced in Section 2.2.3. Due to the former two uniform functions that assign Node IDs and Object IDs, three DHTs, Chord, CAN and Pastry have the same imbalance factor, $O(\log_2 N)$. Because Tapestry’s ID space is not wraparound, it achieves the worst natural load-balancing.

Tapestry and Pastry are designed to publish soft-state information that is a pointer to an object’s storage node. With soft-state, the approach to distribute objects to nodes is separated from their size, which determines that they are not appropriate for a storage system. Due to its simplicity, its provable correctness and the availability of an implementation, Chord is chosen for this research. Although the natural load-balancing motivates applying a DHT as a storage system, a more sophisticated and more dynamic extra load-balancing approach is required by a DHT-based storage system.

According to currently existing extra load-balancing approaches, three types of load-balancing techniques, virtual node approach, network proximity and replication, have been investigated in Section 2.3. The introduction of multiple virtual nodes in each participating machine further improves the natural load-balancing property in DHTs. Moreover, containing multiple virtual nodes in a machine splits the objects in the machine into a number of groups according to their stored virtual nodes. This provides an opportunity to transfer a group of objects in a machine to reduce the storage space usage by migrating a number of virtual nodes to other machines. Nonetheless, virtual node’s migration target should be chosen with more consideration. Embedding network proximity information in DHTs is not especially designed for a load-balancing approach to DHTs. Most DHT’s operations, including load-balancing operations, benefit from the consideration of network proximity information. The simple proximity scalars need to be transformed into landmark vectors or even network coordinates.
However, in order to maximise the usage of network proximity information, the mismatch of dimensions between a DHT’s system structure and network proximity information should be solved. The replication load-balancing approach that is introduced at the end of this chapter concentrates on another aspect, the network issue, of a DHT-based storage system, besides the storage load-balancing. It is mainly used to solve the hotspot problem caused by popular objects that are inevitable in a storage system. Existing replication approaches lose the consideration of a practical hotspot detection approach, the cost of the replication operation and location-varying hotspots.

Since Chord is chosen as the basis for this research, Chapter 3 starts from an analysis of storage load-balancing and network issues in a Chord-based storage system.
Chapter 3

Proximity-aware Load-Balancing Approach

3.1 Introduction

In a Chord-based storage system, the two important properties in each participating machine are network usage and local disk storage space usage, although the actual required computing resources are not only these two during operation of the Chord-based storage system. In each machine, there are also other computing resources, such as CPU and memory usage. The load-balancing research presented in this thesis concentrates on each machine’s storage space usage and relevant network issues.

On the one hand, all machines contribute their local storage space to the whole accessible storage space of the Chord-based storage system, which is consumed by the stored files through file insertion operations that can be performed from any participating machines in the system, according to the description of file operations in Section 1.3.2. In a DHT, the distribution of key-value pairs is not affected by any user’s preferences. It is also not appropriate to set a global component to control file insertions because a DHT’s scalability is adversely affected by the existence of global components. Therefore, in the system, there may be a storage overload that is defined by the emergence of any participating machine that exceeds its local storage quota due to the key-value pairs being stored through the insertion operations from other machines. In this case, from the perspective of the file insertion operation, a good load-balancing approach is required to ensure that all machines in the system have roughly the same local storage usage rate. The uniform distribution of local storage usage rate across all machines is a feasible means to prevent the emergence of storage
overload in the system.

In contrast, the network utilisation of each machine defines the Chord-based storage system’s capability for a participating machine to fulfil file operation requests simultaneously for other machines. The network bandwidth is occupied by data transmissions with other machines, when multiple file operation requests arrive at the machine at the same time. The more data transmissions a machine starts during a fixed period of time, the harder it is for the machine to fulfil more requests, the longer it is for each requestor to wait for its transmission to finish. Finally, due to too many simultaneous data transmissions from the single machine, it becomes a hotspot in the Chord-based storage system. A hotspot indicates a machine with persisting long-time high network transfer rate that prevents it from fulfilling further requests. Therefore, network issues, such as hotspots, should be considered when a load-balancing approach solves the storage overload problem.

Section 3.2 discusses the two important aspects of the Chord-based storage system, namely the storage space usage and the network utilisation. The former part of this section (Section 3.2.1) introduces the target of the storage load-balancing, three important time points to evaluate a system’s storage status, and the candidate method to solve the storage overload problem for the Chord-based storage system. The latter part (Section 3.2.2) discusses the network issues of the Chord-based storage system, which include the research target for the network utilisation and the definition of a hotspot in the system. In Section 3.3, a Hilbert curve approach is proposed to cluster the machines in the Chord-based storage system according to their network proximity information. A case study is provided to explain the Hilbert curve approach more clearly. Based on the proximity clusters that are produced by the Hilbert curve approach, a proximity-aware virtual node storage load-balancing approach is proposed in Section 3.4. Finally, Section 3.5 makes a suggestion to reduce the number of hotspots in the Chord-based storage system using a replication approach. Section 3.6 summarises the chapter.

3.2 Storage Space and Network Utilisation

3.2.1 Storage Space Usage

In a Chord-based storage system, a participating machine’s storage space is separated into two parts, namely system storage space and Chord storage space. Usually, one or more hard disk drives construct the whole machine’s local storage space. In the
system storage space, there are the machine’s operating system, the components of
the Chord-based storage system and all other local users’ files and applications. The
Chord storage space is specifically used for the Chord-based storage system, and it is
designated to be the space for storing key-value pairs inserted into the storage system.
Therefore, the target for storage load-balancing in this research is the Chord storage
space. However, the proportion of Chord storage space in each machine’s whole local
storage space and whether they are located in the same hard disk or multiple hard disks
are out of the scope of this research. It is assumed that a fixed amount of storage space
is allocated as Chord storage space in each machine. According to the available free
local storage space in each machine, a quota is given to each machine for the Chord
storage space in order to prevent there being overloaded storage which will affect the
entire machine storage space. More detailed settings of machine’s storage space in
the experiments will be presented in Section 4.4.2. In order maximally to exploit
the Chord storage space in each machine, a storage load-balancing approach will be
proposed for the Chord-based storage system. Before that, this research provides a
systematic definition of the storage overload based on the storage space utilisation
and two special time points, namely, Unreliable Time Point (UTP) and First Warning Time
Point (FWTP).

The storage space utilisation is considered as a criterion of a machine’s storage sta-
tus in the Chord-based storage system, because DHTs are designed to be deployed in
a heterogeneous environment in which all machines have different specifications, such
as different local storage capacity and different network bandwidth. The storage space
utilisation is a relative value which measures a machine’s storage capability, whereas
the used storage space is an absolute value which measures a machine’s storage us-
age. If the absolute value of the used storage space is set as a criterion, a negotiation
on the appropriate criterion is required among all the participating machines, in order
to prevent the situation where the absolute criterion value is bigger than some ma-
chines’ total storage capacity. When the collective storage space of the Chord-based
storage system needs to be extended by adding more machines with available Chord
storage space, there is a storage capacity limitation that is set by the absolute criterion
to restrict the joining machines to have a smaller storage capacity than the absolute
criterion. Otherwise, after each machine’s joining, the whole system’s absolute crite-
rion is updated, which is not feasible in the Chord-based storage system in which each
machine only has a limited view on the whole system. Therefore, in order to keep
the Chord-based storage system’s flexibility and scalability, it is not reasonable to lay
down an absolute criterion of the used storage space of machines. Instead of using an absolute criterion, each machine evaluates its own storage status according to the storage space utilisation, which is defined by the following equation:

$$U_j = \frac{L_j}{S_j}$$  \hspace{1cm} (3.1)

where $U_j$ (Utilisation) is the storage utilisation of Chord storage space in machine $j$; $L_j$ (Load) is the current used storage in machine $j$; and $S_j$ (Space) is the whole Chord storage space in machine $j$. Each machine in the Chord-based storage system determines itself to be a storage overloaded machine when it has 100% storage space utilisation. In this research, the storage space utilisation is employed. However, a hybrid scheme could also be used to determine a machine’s storage status by combining the available storage space left and the used storage space utilisation in a machine.

A single machine’s storage overload has a deep effect on the Chord-based storage system. If a machine’s storage space utilisation ever reaches 100%, that machine runs out of its Chord storage space. Storage overload happens in the system, which means that the machine is not able to accept any further key-value pairs in its Chord storage space. This malfunction of a single machine would cause key-value pairs to be lost if there is not any global component to avoid overloaded machines. However, in a DHT, the avoidance of any machines for key-value pair insertion is not feasible according to the description of the key-value pair insertion operation in Section 1.2.3, in which there is no global view or sub-global view across the whole system. Therefore, the first occurrence of a machine with 100% storage utilisation in the Chord-based storage system denotes that the storage system is on the verge of becoming unreliable. The time when the first machine in the system reaches 100% storage utilisation is called the Unreliable Time Point (UTP). In order to warn the system of a potential unreliable status before a machine’s storage utilisation reaches 100%, which is too late for a warning purpose, a lower storage utilisation for each machine in the system, for example $X\%$ ($0 < X < 100$), is predefined to tell the machine to be prepared for overload.

From the perspective of the whole system, there are two other time points, besides the UTP, to record two significant storage statuses for a warning purpose. The first time point is the time when any machine’s storage utilisation reaches $X\%$. If one machine’s storage utilisation exceeds $X\%$, the storage system has a warning of storage overload that requires reactions to reduce the trend of further storage space utilisation increasing in the warned machine. This time point is called the First Warning Time.
Point (FWTP), which requires a storage load-balancing approach to start working after this time point. When all machines have their storage utilisation above \( X\% \), every machine is warned by their potentially overloaded local storage usage. This is the second system status time point and is called the All Warning Time Point (AWTP). Section 3.4.3 discusses more details of these three system time points, \( UTP \), \( FWTP \) and \( AWTP \), and the value of \( X \).

If the Chord-based storage system reaches \( UTP \), there are no efficient operations that can help without adding more storage space into the system. This being the case, an objective of the storage load-balancing that is proposed in this research is to postpone the advance towards \( UTP \) once \( FWTP \) has been reached. After a storage load-balancing approach starts to work, in order to relieve a storage overloaded machine, the most direct and efficient theoretical solution is to migrate a part of its stored key-value pairs to other machines. In a Chord-based storage system, machines’ storage spaces are loaded with key-value pairs by only one operation, namely file insertion. Frequent file insertion operations would significantly increase the storage usage of the whole system. The only activity leading to a decrease of utilisation of system’s storage space is file expiration as explained in Section 1.3.2. Some other Chord-based systems are implemented with a file deletion operation that can be initiated by the file’s owner. However, both file expiration and file deletion are user-oriented. None of them is able to guarantee that storage overload would not emerge before the storage overload is released by any of these two operations. Moreover, the storage space released by any of these two user-oriented operations might not be in any of the storage overloaded machines. Because of this, the storage load-balancing approach that is proposed in Section 3.4 employs a system level migration mechanism based on virtual nodes and virtual node migration to manage storage overload in a Chord-based storage system.

In order to achieve the objective of the proposed storage load-balancing approach, extra network utilisation results from the operations of the load-balancing approach, such as virtual node migration. The effect of this is considered next.

### 3.2.2 Network Utilisation

According to the analysis of storage space usage in Section 3.2.1, the only operation that loads machines is the file insertion operation. Refusing to receive key-value pairs to be stored in overloaded machines is able to retard further overloading on these machines. However, in order to relieve the overloaded machines, a Chord-based storage system requires the migration of a part of the key-value pairs from overloaded
CHAPTER 3. PROXIMITY-AWARE LOAD-BALANCING APPROACH

machines to other machines and the redirection of the retrieval requests from the over-
loaded machines to the new machines holding the migrated key-value pairs. Both
request redirection and key-value pair migration are remote operations which rely on
the network and increase the system’s network usage. So, in this section, the network
issues when Chord is used to implement a distributed storage system are analysed.

Due to the lack of administrator access to network devices, such as routers and
switches, in the experimental environment as described in Section 4.4.1, the network
issues analysed in this section concentrate on the network utilisation of a single ma-
chine. There are two related network issues, namely the network structure connecting
all the system’s machines and the network utilisation of each individual machine. The
system’s network and its devices, such as routers and switches, connecting all ma-
chines are not considered in this thesis. The system’s network is not restricted to be
of any particular form. It could be a LAN (Local Area Network), a WAN (Wide Area
Network) or the most complex one, the Internet, according to the scale of a Chord-
based storage system and its underlying network infrastructure. In the experimental
work reported in Section 4.4.1, the underlying network is a LAN. Therefore, in this
thesis, the research on the network utilisation concentrates on a single machine of a
Chord-based storage system in a LAN-based cluster environment with good network
connections.

The inbound network traffic and the outbound network traffic comprises the whole
network utilisation in a single machine. TCP is employed by the Chord-based storage
system to transfer key-value pairs for the file insertion operation and the file retrieval
operation. TCP always attempts to send data at the maximum transfer rate which is ne-
gotiated with both the receiver’s capability and the network’s capability [51]. Because
the experimental Chord-based storage system is deployed in the LAN on a full duplex
100Mbit/s ethernet, as described in Section 4.4.1, a machine’s maximum transfer rate
for both the inbound network traffic and the outbound network traffic is around 12
MByte/s. Due to the existence of simultaneous key-value pair transfers in the Chord-
based storage system, not all file operations can be accomplished at this maximum
transfer rate.

Figure 3.1 sketches the case of a group of machines concurrently inserting different
key-value pairs into the same machine providing its Chord storage space. The inbound
network traffic of the server machine is contributed by the key-value pair transmissions
from a number of other machines acting in their client role. Due to the limitation of the
server machine’s inbound network bandwidth, the maximum receiving rate is around
12MBytes/s. Therefore, the outbound network traffic of the client machines hardly ever achieves the maximum transfer rate during the whole procedure of their key-value pairs insertion operations.

![Figure 3.1: Simultaneous inbound network traffic to a machine by key-value pair insertion operations.](image)

Figure 3.1 is the case of a number of client machines requesting key-value pairs from the same server machine at the same time. Similar to the situation of the inbound network traffic in Figure 3.1, the outbound network bandwidth of the server machine is shared by a number of other machines acting as client requestors, which causes a fulfilment delay of the retrieval requests from these requestors.

There are two possible reasons leading to a large amount of simultaneous network traffic to share one machine’s inbound or outbound traffic.

The first reason is the situation where the whole storage system is suffering from a large volume of frequent file operations. In this case, almost all machines would suffer from the same situation as presented in Figure 3.1 and Figure 3.2. A specific frequency of the file operations that a Chord-based storage system can tolerate is defined to be its operational capacity in Section 5.4.1.
A second reason is a popular file that attracts a large number of requestors. If many key-value pair retrieval requests arrive at a machine with the key-value pair of a popular file at the same time, the machine would try to fulfil all these retrieval requests as demonstrated in Figure 3.2. However, the key-value pair insertion operation has no similar concern in a Chord-based storage system. In practice, the situations in Figure 3.1 and Figure 3.2 are not simply two symmetric cases. Because Chord’s natural load-balancing property evenly distributes key-value pairs across all machines regardless of their insertion locations or their owners, as discussed in Section 2.2.1.1, two different key-value pairs are rarely inserted simultaneously into the same machine. Unless a large number of users are inserting the same key-value pair into the system, which is quite rare in real life, a machine cannot be simultaneously requested by a large number of key-value pair insertion operations.

Whichever reason causes a large number of file operations to arrive at one machine, it could be overwhelmed by these file operations that make the machine transfer data constantly at the maximum transfer rate. The network overload is defined by the emergence of the overwhelmed machines caused by either the frequent file operations or
the popular files. The overwhelmed machines are defined as the *hotspots* in a Chord-based storage system. Nevertheless, the operational capacity indicates the fulfilment capability of a Chord-based storage system that is difficult to be increased unless the setting of the Chord-based storage system changes. At this point, it is hard to solve the problem of the overwhelmed machines caused by operational capacity. This research suggests a replication approach in Section 3.5 to reduce the number of hotspots that are being overwhelmed by a large number of retrieval requests for the popular key-value pairs.

A hotspot delays the currently fulfilling file retrieval operations whose initiators have to wait for a long time until the transmissions of key-value pairs are completed. This decreases the service quality of a Chord-based storage system. Meanwhile, further file retrieval requests may be denied by a hotspot due to its limited network bandwidth, which actually interrupts the service of the Chord-based storage system from the perspective of the machines that initiate these file retrieval requests. The more hotspots there are in the storage system, the more machines are affected by the hotspots. Therefore, any hotspots should be detected, and subsequent steps taken to relieve their overloaded network utilisation.

By using TCP to transfer key-value pairs, the maximum network traffic does not necessarily denote that the machine is a hotspot. Therefore, the appropriate monitor target for a hotspot detection approach should be the queues for network transmission, because the existence of data in network queues implies that either the machine or the network cannot process this queued data currently. More details of the hotspot detection approach for the Chord-based storage system in the experimental environment (Section 4.4) is described in Section 5.4.2.

### 3.2.3 Recap

This section has analysed the target and the objective of the storage load-balancing approach for a Chord-based storage system. According to the analysis, three important time points, $U TP$, $FWTP$ and $AWTP$, are defined for the storage load-balancing approach to determine the storage status of the whole system. Inspired by the virtual node approach, as discussed in Section 2.3.1, a virtual node migration method will be employed to solve the storage overloads.

Because the operation of the storage load-balancing approach produces more network traffic, each machine’s network utilisation has also been analysed. A suggestion to reduce the number of hotspots in the storage system will be provided in Section 3.5.
In order to take proximity information into consideration for the storage load-balancing approach, the proximity information is discussed next.

### 3.3 Proximity Information

According to the above discussion of both the storage load-balancing and the network load-balancing for a Chord-based storage system, key-value pair migration is a direct and inevitable method to relieve storage overload. The choice of a candidate machine to which a key-value pair is migrated is an important factor to accelerate the migration operation and thus to improve the performance of the whole storage system. According to the investigation of the network proximity information in DHTs in Section 2.3.2, in this section the Hilbert curve is employed as the essential method to cluster machines according to their network proximity information in the proposed storage load-balancing approach for the Chord-based storage system. The Hilbert curve (Section 3.3.1) is used to produce another identifier for each machine, the Hilbert ID (Section 3.3.2), which preserves the machine’s network proximity to other machines in the storage system. Between two pairs of machines, \((M_1, M_2)\) and \((M_3, M_4)\), the comparison between the network distance of \(M_1, M_2\) and the network distance of \(M_3, M_4\) can be performed according to these four machines’ Hilbert IDs. The Hilbert ID plays an important role for both key-value pair migration and key-value pair replication, because both operations can be carried out efficiently among machines with small network distance. For storage load-balancing, the key-value pairs migration between two ‘near’ machines in the network can be accomplished in shorter time, which will reduce the network transmission time. As for network issues, the replicas of the key-value pairs can also be efficiently transmitted to machines with small network distance. In the Chord-based storage system, Hilbert IDs are organised and maintained by each Chord node in its Hilbert table (Section 3.3.3) which is the clustered proximity information required by the storage load-balancing approach. In this section, Hilbert curve, Hilbert ID and Hilbert table are demonstrated in the Chord-based storage system.

#### 3.3.1 Hilbert Curve

The Hilbert curve was introduced by David Hilbert [46] as a type of Space-Filling Curve (SFC), that is, a class of one-dimensional curves that are able to pass through all vectors in a multi-dimensional region. In the 19th century, the question arose whether
or not the interval $[0, 1]$ can be a surjective mapping onto $[0, 1]^2$, where $[0, 1]^2 = [0, 1] \times [0, 1] = \{(a, b) \mid a \in [0, 1] \text{ and } b \in [0, 1]\}$. G. Peano gave a positive answer to this question in 1890 and he found a curve for continuous mapping from $[0, 1]$ to $[0, 1]^2$. After Peano, the first discovered SFC that accomplishes this mapping is named the Peano curve. If discrete numbers are considered, an equivalent mapping can also be found from nonnegative integers to squares in a 2-dimensional region. In a two-dimensional region, the squares are located by a two-dimensional vector. For example, the left diagram in Figure 3.3 illustrates a discrete Peano curve in an order-2 region of 2-dimensional space. In the Peano curve diagram, the region is divided into $2^2 \times 2^2 = 16$ squares, which are labelled by vectors from $(0, 0)$ to $(3, 3)$. The discrete Peano curve labelled from 0 to 15 passes through every square only once and each nonnegative integer corresponds to a 2-dimensional square labelled by a vector. The region that is filled by a SFC is not limited to be 2-dimensional; for multi-dimensional space, similar SFCs can also be found. In other words, a mapping from $[0, 1]$ to $[0, 1]^2$ can also be extended to a mapping from $[0, 1]$ to $[0, 1]^n$, where $n$ is the number of dimensions ($n \geq 2$) and $[0, 1]^n$ is an $n$-fold Cartesian product. There are also mappings from nonnegative integers to vectors in a multi-dimensional region. In this case, the SFCs can be treated as a one-dimensional indexing of the vectors in a multi-dimensional region. The whole range of the one-dimensional index is called the interval in the rest of this section. Indexing a multi-dimensional region by a one-dimensional curve is the main function when SFCs are employed. The Peano curve is not the only SFC, Figure 3.3 illustrates two other types of SFC.

Figure 3.3: Peano Curve, Reflected Binary Gray-code (RBG) Curve and Hilbert Curve of order 2 pass through 2-dimensional squares.
Distance preserving is another function provided by SFCs and this has attracted researchers to employ it in secondary key retrieval [14], geometric data structures [1] and clustering objects with multiple attributes [22]. Hilbert first discovered the heuristic principle that was the first general geometrical generating procedure for a whole class of SFCs [46]. In the example of mapping from an interval to a two-dimensional square, if the square is divided into four equal sub-squares, there are also mappings from the four subintervals to the four corresponding sub-squares. This argument repeats, if each sub-square and each interval are partitioned further. Figure 3.4 illustrates the Hilbert curves of order 1, 2, and 3, in which the Hilbert heuristic principle is demonstrated. The squares in 2-dimensional space are labelled by a vector, for example, the lower left square is labelled as \( Q(0,0) \) in all three diagrams. The solid lines passing through all squares are Hilbert curves of order 1, 2, and 3, which index all squares. A mapping of smaller order can be put into a part of the bigger order, which means that two adjacent squares would be relatively indexed near to each other on the Hilbert curve compared to the whole Hilbert curve of the whole region of bigger order. For example, the adjacent squares \( Q(1,1) \) and \( Q(2,1) \) are mapped as 2 and 13 respectively on the Hilbert curve of order 2, which does not preserve the proximity of \( Q(1,1) \) and \( Q(2,1) \). However, when the whole Hilbert curve of order 2 is put into the lower left quarter part of the Hilbert curve of order 3, the order increasing makes the whole order 2 curve more local. In the Hilbert curve of order 3, \( Q(1,1) \) and \( Q(2,1) \) are mapped as 2 and 7 respectively. Although they are still not adjacent on the Hilbert curve, they are near compared with the whole interval of \([0,63]\). By this means, after the mapping from a one-dimensional interval to a multi-dimensional region, the proximity information

![Figure 3.4: Procedure of producing Hilbert curves of order 1, 2, and 3 [46].](image-url)
among the vectors in the multi-dimensional region is preserved in the curve to a large extent.

The Hilbert curve is not the only SFC with the distinct dimensions mapping and the proximity information preserving. Faloutsos et al. [14] compared three common SFCs from which they wanted to find a good distance-preserving mapping for secondary key retrieval. The basic Peano curve, the Reflected Binary Gray-code (RBG) curve and the Hilbert curve (see Figure 3.3) were chosen for their comparison. In their work, these three SFCs were evaluated by two tests, namely the test for the disk access of range queries and the test for nearest neighbour queries. Because the research in this thesis concentrates on SFC’s proximity information preserving property, only the experiment results of the test of nearest neighbour queries are taken into consideration. In their test, a Maximum Neighbour Distance (MND) is defined to measure how much worse the multi-dimensional vectors are indexed by these SFCs. Given a SFC and a radius, the MND of a point (square) \( X \) is the largest distance in multi-dimensional space from another point (square) whose mapped index is within the distance of the given radius from the index of point \( X \) on the one-dimensional curve.

The distance between two points (squares), \( X \) and \( Y \), in a multi-dimensional region is indicated by the Manhattan Distance [27] \( M_{\text{Dist}}_{X,Y} \), which is defined as follows:

\[
M_{\text{Dist}}_{X,Y} = \sum_{i=1}^{n} |X_i - Y_i|
\]  

(3.2)

where \( n \) is the number of dimensions, \( X_i \) and \( Y_i \) are the \( i \)-th elements in the coordinate of the point \( X \) and the point \( Y \), respectively. The results of the Faloutsos et al. [14] experiment are presented in Table 3.1, Table 3.2 and Table 3.3, where \( N = 2^b \) is the maximum value in each dimension of the point coordinate in a multi-dimensional space (\( b \) is the order). The radius is chosen to be \( N/2 \) in all tests in this section.

<table>
<thead>
<tr>
<th>Order ( b )</th>
<th>( N \times N ) Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 ( \times ) 2</td>
<td>1.00</td>
<td>1.00</td>
<td>1.50</td>
</tr>
<tr>
<td>2</td>
<td>4 ( \times ) 4</td>
<td>2.00</td>
<td>2.75</td>
<td>2.75</td>
</tr>
<tr>
<td>3</td>
<td>8 ( \times ) 8</td>
<td>3.28</td>
<td>5.00</td>
<td>4.84</td>
</tr>
<tr>
<td>4</td>
<td>16 ( \times ) 16</td>
<td>4.89</td>
<td>8.52</td>
<td>7.91</td>
</tr>
</tbody>
</table>

Table 3.1: Average MND of all points (squares) in two-dimensional space.

Therefore, the experimental results concluded that the Hilbert curve produces a
Table 3.2: Average MND of all points (cubes) in three-dimensional space.

<table>
<thead>
<tr>
<th>Order b</th>
<th>N x N x N Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 x 2 x 2</td>
<td>1.00</td>
<td>1.00</td>
<td>2.00</td>
</tr>
<tr>
<td>2</td>
<td>4 x 4 x 4</td>
<td>2.00</td>
<td>2.50</td>
<td>3.31</td>
</tr>
<tr>
<td>3</td>
<td>8 x 8 x 8</td>
<td>3.23</td>
<td>4.04</td>
<td>5.10</td>
</tr>
<tr>
<td>4</td>
<td>16 x 16 x 16</td>
<td>4.20</td>
<td>5.61</td>
<td>7.03</td>
</tr>
</tbody>
</table>

Table 3.3: Average MND of all points (vectors) in four-dimensional space.

<table>
<thead>
<tr>
<th>Order b</th>
<th>N x N x N x N Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 x 2 x 2 x 2</td>
<td>1.00</td>
<td>1.00</td>
<td>2.38</td>
</tr>
<tr>
<td>2</td>
<td>4 x 4 x 4 x 4</td>
<td>2.02</td>
<td>2.28</td>
<td>3.50</td>
</tr>
</tbody>
</table>

better distance-preserving mapping than the other two SFCs in all of 2, 3 and 4 dimensions. However, in their experiment, only the maximum distance (MND) from a point to this point’s neighbours was considered, which cannot guarantee that the order of distances from the point to its neighbours was also preserved by these three SFCs.

In order to make a more appropriate comparison among these three SFCs, a complementary testing has been undertaken by the author of this thesis to investigate the ranking of the distances between a point X and each point in the range of the given radius on a SFC from point X. Spearman’s correlation coefficient [3] is employed to relate the ranking of the distance on a SFC to the ranking of the Manhattan distance in a multi-dimensional space, because Spearman’s correlation coefficient is able to show the differences of a group of elements in two distinct ranking lists produced by two different criteria. The value of Spearman’s correlation coefficient R is yielded by the following equation:

\[
R = 1 - \frac{6 \times \sum t_i^2}{\text{num} \times (\text{num}^2 - 1)} \tag{3.3}
\]

where \( t_i = r_{1i} - r_{2i} \) is the difference of an element i’s ranking \((r_{1i} \text{ and } r_{2i})\) in two lists, \( r_{1} \) and \( r_{2} \), and \( \text{num} \) is the total number of elements in each of these lists. For a point X, the distances of the points in the range of the radius from point X on a SFC are ranked in order to produce the first ranking list. The second ranking list consists of the Manhattan distances from point X to the same set of points. Then the Spearman’s correlation coefficient of these two rank’s correlation is evaluated. For example, suppose the points \( P1, P2, P3 \) and \( P4 \) are in the range of the radius from point X. Assume that when these points are ordered by SFC distances from point X
(in ascending order), the list is $P_3, P_4, P_1, P_2$. Then, if the list when they are ordered by Manhattan distances from point $X$ (in ascending order) is also $P_3, P_4, P_1, P_2$, the coefficient is 1, which means that the two orders are identical. If the list when they are ordered by Manhattan distances from point $X$ (in ascending order) is $P_2, P_1, P_4, P_3$, the coefficient is $-1$, which means that the two orders are the reverse of each other. The closer to 1 the coefficient is, the more closely associated with each other these two orders are. So a Spearman’s correlation coefficient value closer to 1 indicates a better distance and order preserving mapping provided by a SFC. In Table 3.4, Table 3.5 and Table 3.6, the results of this complementary testing are reported.

<table>
<thead>
<tr>
<th>Order b</th>
<th>$N \times N$ Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2 \times 2$</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>$4 \times 4$</td>
<td>1.00</td>
<td>0.83</td>
<td>0.74</td>
</tr>
<tr>
<td>3</td>
<td>$8 \times 8$</td>
<td>0.89</td>
<td>0.70</td>
<td>0.71</td>
</tr>
<tr>
<td>4</td>
<td>$16 \times 16$</td>
<td>0.91</td>
<td>0.66</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Table 3.4: Average ranking correlation coefficient of all points (squares) in two-dimensional space.

<table>
<thead>
<tr>
<th>Order b</th>
<th>$N \times N \times N$ Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2 \times 2 \times 2$</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>$4 \times 4 \times 4$</td>
<td>1.00</td>
<td>0.91</td>
<td>0.66</td>
</tr>
<tr>
<td>3</td>
<td>$8 \times 8 \times 8$</td>
<td>0.88</td>
<td>0.76</td>
<td>0.54</td>
</tr>
<tr>
<td>4</td>
<td>$16 \times 16 \times 16$</td>
<td>0.73</td>
<td>0.70</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 3.5: Average ranking correlation coefficient of all points (cubes) in three-dimensional space.

<table>
<thead>
<tr>
<th>Order b</th>
<th>$N \times N \times N \times N$ Grid</th>
<th>Hilbert Curve</th>
<th>RBG Curve</th>
<th>Peano Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2 \times 2 \times 2 \times 2$</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>$4 \times 4 \times 4 \times 4$</td>
<td>1.00</td>
<td>0.95</td>
<td>0.63</td>
</tr>
<tr>
<td>3</td>
<td>$8 \times 8 \times 8 \times 8$</td>
<td>0.87</td>
<td>0.81</td>
<td>0.48</td>
</tr>
<tr>
<td>4</td>
<td>$16 \times 16 \times 16 \times 16$</td>
<td>0.74</td>
<td>0.70</td>
<td>0.48</td>
</tr>
</tbody>
</table>

Table 3.6: Average ranking correlation coefficient of all points (vectors) in four-dimensional space.

According to these experimental results of average ranking correlation coefficient, the distances of points on the Hilbert curve still achieves a higher association to the
distances of points in the 2, 3 and 4 dimensional spaces, which represents a better
distance and order preserving feature of the Hilbert curve over the other two SFCs.
Consequently, in this research, the Hilbert curve is employed to cluster machines ac-
cording to their network proximity information, which is used by the proposed storage
load-balancing approach described in Section 3.4.

3.3.2 Hilbert ID

The proximity information for the machines in a Chord-based storage system involves
two parts: landmark machines and network distances between machines. Landmark
machines are chosen from all participating machines in the Chord-based storage sys-
tem. There could be a number of criteria to choose landmark machines in a storage
system according to different requirements in different deploying environments. For
example, in a cluster environment, in which machines are grouped into several clus-
ters, every cluster has at least one landmark machine. The machine that is chosen to
be a landmark machine should be persistent in the system and have good connections
to the network and few irrelevant services running. However, these conditions cannot
be fulfilled completely in practice. In the experiments for this research, the landmark
machines are chosen from each laboratory without the Chord-based storage system
running. More details of the experimental settings are given in Section 4.4. Further
discussion of the method and process to choose landmark machines is out of the scope
of this thesis.

Each participating machine has a landmark vector that is produced using landmark
machines in the system. In a network, there are several factors affecting the efficiency
of data transmission between two machines, such as each machine’s network band-
width, data transfer latency between them and the routing path. Data transfer latency
between two machines in a network is a distance metric that is easy to measure but still
useful. Therefore, the Round Trip Time (RTT) is employed to measure the distance
between two machines in the network. A machine’s landmark vector is represented
by a vector of RTTs to all landmark machines from this machine. For example, if
there are 5 landmark machines in the entire system, then \((LM_1, LM_2, LM_3, LM_4, LM_5)\)
is a machine’s landmark vector, where \(LM_i\) is the latency of RTT from the machine to
landmark machine \(i\). When a machine joins the system, it learns the existing landmark
machines from its bootstrap machine that is introduced in Section 1.2.2. Each particip-
ating machine thus has the knowledge of its landmark vector when it joins into the
Chord-based storage system.
In a Chord-based storage system, it is difficult to harness a machine’s multi-dimensional landmark vector to cluster the machines, which is, then, employed by load-balancing approaches. This is where the Hilbert curve is required to map from a vector in a multi-dimensional space to an index in a one-dimensional interval. The mapped index in the one-dimensional interval from a machine’s landmark vector is the machine’s Hilbert ID. The comparison of two one-dimensional Hilbert IDs produces the network distance of the two machines owning these two Hilbert IDs, because the mapping from a machine’s landmark vector to its Hilbert ID preserves the proximity information in the multi-dimensional space. Different machines may have the same Hilbert ID if they have the same landmark vector when they are placed in very ‘near’ locations in the network.

The Hilbert ID space and Chord ID space can be chosen relatively independent of one another, but there are practical reasons that make it sensible to choose them to be of equal size.

If the Hilbert ID space is bigger than the Chord ID space, this would lead to an overlap of Hilbert ID space on Chord ID space. In this case, the head part and the tail part of the Hilbert ID space are situated in the same part of the Chord ID space. The loading status of machines with their Hilbert ID in either the head or the tail of the Hilbert ID space would be stored in the same area of Chord ID space. However, the machines with their Hilbert ID in the head and the tail of the Hilbert ID space are obviously not ‘near’ to each other.

If the Chord ID space is bigger than the Hilbert ID space, some nodes in Chord would not be responsible for maintaining the proximity information of other machines. This leads to a load-imbalance in the Hilbert ID space.

Therefore, if the Hilbert ID space and the Chord ID space have identical size, a Hilbert ID can be treated the same as a Chord ID in a Chord-based storage system and can be directly designated as the Data ID of the key-value pair, which is then called a Hilbert key-value pair. In this case, a Hilbert ID also has its successor and its predecessor on the Chord Ring. There is only one Hilbert key-value pair for each machine. Each machine’s Hilbert ID and the machine’s network information are stored in the storage system following the key-value pair insertion operation for normal key-value pairs which is presented in Section 1.2.3, and the machine’s network information is assigned to be the value of the machine’s Hilbert key-value pair. The Hilbert key-value pairs of machines with short network distance between each other are likely to be stored in the same Chord node, because machines’ Hilbert key-value pairs are stored
in the successor of their keys, namely their Hilbert IDs. In this way, the machines
with adjacent Hilbert IDs in the Chord ID space are clustered into the successor of
these machines’ Hilbert IDs. Section 3.3.4 provides an example that demonstrates the
insertion of a Hilbert key-value pair and its usage.

For the Hilbert curve, the interval of the one-dimensional indices is \([0, 2^{d \times b})\), where
\(d\) and \(b\) are number of dimensions and order, respectively. According to the implemen-
tation of Chord using SHA-1 to produce Chord ID, the Chord ID space \([0, 2^{160})\) should
have the same size as the Hilbert ID space \([0, 2^{d \times b})\), therefore \(d \times b = 160\). Table 3.7

<table>
<thead>
<tr>
<th>Dimension (d)</th>
<th>Order (b)</th>
<th>Each dimensional maximum value (2^b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>160</td>
<td>(2^{160})</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>(2^{80})</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>(2^{40} \approx 1.1 \times 10^{12})</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>(2^{32} = 4292967296)</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>(2^{20} = 1048576)</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>(2^{16} = 65536)</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>(2^{10} = 1024)</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>(2^{8} = 256)</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>(2^{5} = 32)</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>(2^{4} = 16)</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>(2^{2} = 4)</td>
</tr>
<tr>
<td>160</td>
<td>1</td>
<td>(2^{1} = 2)</td>
</tr>
</tbody>
</table>

Table 3.7: Possible values of dimension \(n\) and order \(b\), if \(d \times b = 160\).

lists the possible values of dimension \(d\) and order \(b\) that fulfil \(d \times b = 160\). Only 160’s
positive integer dividers are shown, because other integers would make \(d \times b \neq 160\), \(d\)
and \(b\) are all integers. Even if \(d \times b = 161\) or \(d \times b = 159\), the Hilbert ID space would
increase to twice the Chord ID space; or decrease to half the Chord ID space. The
dimension \(d\) corresponds to the number of landmark machines in the system. On the
one hand, more landmark machines would provide more resolution to the machine’s
network coordinate vector, although more measurements of RTT are required for each
machine’s proximity information updating. On the other hand, more landmark ma-
chines would reduce the resolution of the measurement of RTT, according to the data
in the last column of Table 3.7. If \(d = 160\), the elements of a machine’s network co-
ordinate vector reduce to a binary digit. Therefore, the middle rows (\(d = 8\), \(d = 10\),
\(d = 16\)) are appropriate to be used in the implementation of the storage load-balancing
approach for the Chord-based storage system.
3.3.3 Hilbert Table

The Hilbert curve is employed to cluster the machines according to their proximity information, namely the relative network coordinate to landmark machines. The Hilbert ID is the mapped identification of a machine from its proximity information, which still preserves machine’s proximity information. In a system with Hilbert IDs, there are two layers on the same Chord Ring, namely Chord ID space and Hilbert ID space. The machine’s locations are still fixed on the Chord Ring. In Chord ID space, the data key-value pairs are stored in the system according to the data’s Chord ID. In Hilbert ID space, however, the Hilbert key-value pairs containing machine’s network information, such as IP Address, are stored according to the machine’s Hilbert ID. Sequentially, the Hilbert key-value pairs with similar Hilbert IDs are located around the same region on the Chord Ring. They may share the same successor, and then are likely to be stored into the same node on the Chord Ring, according to the key-value pair insertion operation described in Section 1.2.3. The Chord node storing the Hilbert key-value pair of “adjacent” machines is called the rendezvous node of these machines. Each machine has a rendezvous node in the system. In a rendezvous node, all Hilbert key-value pairs are stored in a separate space called the Hilbert table. Each machine is able to find its ‘near’ machines from its rendezvous node’s Hilbert table. Finding a machine’s ‘near’ machines can be accomplished by querying the Hilbert table of the machine’s rendezvous node that can be located by routing to the successor of the machine’s Hilbert ID.

If there are \( M \) machines in the system, there should be \( M \) Hilbert key-value pairs for each machine and up to \( M \) rendezvous nodes to store these Hilbert key-value pairs. Accordingly, on average, each Chord node holds a Hilbert key-value pair because there are \( M \) Chord nodes in the system. If there are 2 virtual nodes in each machine, there could be \( 2 \times M \) Chord nodes in the system. Therefore, on average, each Chord node holds \( \frac{1}{2} \) Hilbert key-value pair, which means that at least half of the Chord nodes contain no Hilbert key-value pairs. A Chord node acting as a rendezvous node may store a limited number of Hilbert key-value pairs, so a limited number of ‘near’ machines can be obtained from these rendezvous nodes. Furthermore, the machines whose Hilbert key-value pairs in the rendezvous node’s predecessor and successor of a machine \( C \) are also ‘near’ to the machines whose Hilbert key-value pairs are in \( C \)’s rendezvous node. Therefore, a rendezvous node \( N_r \) should not only store the Hilbert key-value pairs whose Hilbert ID is in \( N_r \)’s responsible zone, but it should also store the Hilbert key-value pairs from \( N_r \)’s successor and predecessor and even further nodes such as its
successor’s successor. There are two options to accomplish this task:

1. A query for ‘near’ machines of a machine is not bounded within the machine’s rendezvous node. Following the rendezvous node’s successor pointer and predecessor pointer, the query is recursively propagated through the Chord Ring in two directions. However, it is difficult to determine the time when the propagated query should be terminated and to control the queries across the whole system. Moreover, the recursive query to further nodes on the Chord Ring would increase the query waiting time significantly.

2. Making all Chord nodes have a Hilbert table is another solution to make the query reach the Hilbert key-value pairs in other rendezvous nodes around the same region of the Chord Ring. Each Chord node not only stores the Hilbert key-value pairs coming from its responsible zone, but it also retrieves the Hilbert key-value pairs from its successor and predecessor. All retrieved Hilbert key-value pairs from the node’s successor and predecessor contribute to this node’s Hilbert table. In this way, each node in the system has a Hilbert table of Hilbert key-value pairs whose Hilbert IDs are ‘near’ to the Node ID on the Chord Ring. To achieve this, an algorithm is required to maintain each rendezvous node’s Hilbert table.

In this research, the second option is chosen. Figure 3.5 shows a Chord node’s Hilbert table which includes three parts. The middle section, which is called the main part, stores the Hilbert key-value pairs whose IDs are in the Chord node’s responsible zone on the Chord Ring. So the Chord node $N_r$ is the rendezvous node of the machines whose Hilbert key-value pair is in the main part of the Hilbert table in $N_r$. The successor part comes from $N_r$’s successor, which is filled from the main part of the Hilbert table of $N_r$’s successor. If there are insufficient Hilbert key-value pairs, the Hilbert

<table>
<thead>
<tr>
<th>Successor Part</th>
<th>Main Part</th>
<th>Predecessor Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hilbert key-value pairs from successor</td>
<td>Hilbert key-value pairs in responsible area</td>
<td>Hilbert key-value pairs from predecessor</td>
</tr>
</tbody>
</table>

Figure 3.5: Three parts of a Chord node’s Hilbert table.
key-value pairs in the successor part of $N_r$’s successor are used. The predecessor part follows the same rule to be populated with Hilbert key-value pairs from $N_r$’s predecessor. It is obvious that, without a limitation on the number of Hilbert key-value pairs in successor part and predecessor part, both these two parts would be populated by all machine’s Hilbert key-value pairs from two directions on the Chord Ring. Therefore, a capacity limit ($C_{\text{part}}$) is set on each node’s Hilbert table’s successor part and predecessor part in order to limit the number of entries in them. Because the main part reflects the actual number of Hilbert key-value pairs in a node’s responsible zone, there is no capacity limit on each node’s main part. Thereafter, even if there are no entries in the main part of a node’s Hilbert table, there are still $2 \times C_{\text{part}}$ Hilbert key-value pairs to fulfil the query of the node’s Hilbert table. In order to choose $C_{\text{part}}$ Hilbert key-value pairs from $N_r$’s successor or its predecessor from a large number of entries in the Hilbert table, a criterion for choosing is necessary. Since the Hilbert table reflects the network distance between machines, the most intuitive criterion is the Hilbert IDs of entries in the Hilbert table. From $N_r$’s successor or its predecessor, $N_r$ should always choose the Hilbert key-value pairs whose Hilbert IDs have the smallest difference from $N_r$’s Chord ID to populate the successor part or predecessor part of $N_r$’s Hilbert table.

Chord was designed for a highly dynamic environment. When the Hilbert table is embedded into a Chord-based storage system, it should adapt dynamically to its deployed environment. Accordingly, a two-part update mechanism is necessary to ensure entries in each Hilbert table are up-to-date.

The first part of the update ensures that the main part in each node’s Hilbert table has no stale Hilbert key-value pairs. Hilbert key-value pairs are maintenance information for the running of the load-balancing approaches in the storage system. Although Hilbert key-value pairs follow the same method as normal key-value pairs to be inserted into the Chord-based storage system, more maintenance of Hilbert key-value pairs in Hilbert tables is required, compared with normal key-value pairs whose reliability in the whole system could be handed over to the upper layer applications. Therefore, in order to ensure that the Hilbert key-value pairs are correct and up-to-date, each machine periodically inserts its Hilbert key-value pair into the Hilbert table of a node in the system. A machine’s Hilbert key-value pair is stored in the Hilbert table of a node with a timestamp representing the time that it was inserted or last updated. After the first Hilbert key-value pair insertion, the following insertion operations merely update the timestamp without the key-value pair’s transmission. This ensures that every machine has a correct rendezvous node to keep its Hilbert table pairs. However, if a
machine’s rendezvous node is changed, the entry in the Hilbert table of its previous rendezvous node would be obsolete. Therefore, a node should, by itself, check the timestamp of the entries in the Hilbert table periodically, in order to find any stale entries and delete them. This part of the periodic update method ensures that the entries in the main part of the Hilbert table in each node are correct and up-to-date. Recall that the main part of a Hilbert table is used to populate the successor part and predecessor part of the Hilbert table in other nodes. This information propagation along the Chord Ring could lead to the emergence of stale entries in successor part and predecessor part, because the changing of the main part of a Hilbert table is not known by other nodes. This is the reason why the second part of the update mechanism is required.

The second part of the update mechanism ensures that the successor part and predecessor part of a Hilbert table contain correct and up-to-date Hilbert key-value pairs. A similar periodic update is employed on the successor part and the predecessor part of a node’s Hilbert table in order to update changes of the main part of the Hilbert table in the node’s successor and its predecessor, respectively. According to the description above, from the main part of the node’s successor’s Hilbert table and the main part of the node’s predecessor’s Hilbert table, the entries with smallest network distance from the node’s Chord ID should be chosen according to their Hilbert IDs. However, stale entries could be propagated through this periodic update of the successor part and the predecessor part. In this case, another aspect, ‘freshness’ of each entry in a Hilbert table, should be considered before the network distance that is represented by Hilbert ID, because the Hilbert ID preserves limited network proximity information from a machine’s landmark vector. The Hilbert ID provides a certain level of indication of a machine’s network proximity information. Consequently, compared with the network distance provided by the Hilbert ID, a Hilbert key-value pair’s ‘freshness’ provided by its timestamp should be regarded as a more important criteria for choosing during the update process of the successor part and the predecessor part in a Hilbert table.

3.3.4 A Case Study

This section demonstrates a small world example of the Hilbert table deployed in a Chord-based storage system. Instead of following the codomain of a cryptographic hash function, such as \([0, 2^{160})\) of SHA-1 in the Chord’s implementation that is described in Section 1.2.1, the capacity of the Chord Ring is set to be 64 in the example for simplicity. The example assumes that there are 11 machines in 2 clusters, each of which has a landmark machine without a storage system running in it. Landmark
1 is the landmark machine in Cluster 1 with 6 storage machines; and Landmark 2 is that in Cluster 2 with 5 storage machines. There are not multiple virtual nodes in each machine, so Figure 3.6 shows the 11 nodes corresponding to the 11 machines on the Chord Ring. Each node’s Chord ID is tabulated alongside. The machines are marked according to their Chord ID: M0, M5, M9, M15, M32, M41, M49, M52, M55, M57 and M63. In Table 3.8, the information about the machines in the two clusters is listed.

Because there are two clusters, each of which has a landmark machine, it is natural
to use a 2-dimensional Hilbert curve of order 3, in order to ensure that $2^{b \times d} = 64$, where $d = 2$ and $b = 3$, according to the description in Section 3.3.2. A 2-dimensional Hilbert curve of order 3 is illustrated in Figure 3.7, which also shows the 11 machines on the 2-dimensional space in the form of dark squares, according to their network coordinates that are presented by the RTTs to the two landmark machines. The x-axis is the RTT to Landmark 1, and the y-axis is the RTT to Landmark 2. Through the machines’ network coordinates and the Hilbert curve, each machine’s Hilbert ID is produced and is shown in each machine’s dark square in Figure 3.7.

When these 11 machines join the storage system, each machine is given its Hilbert key-value pair that is presented in the form of $H5-M32$, which means that the Hilbert ID of the machine with Chord ID 32 is 5 in this example. Figure 3.8 demonstrates the Hilbert key-value pairs of these 11 machines on the Chord Ring. The Hilbert key-value pairs are labeled by squares on the Chord Ring according to their Hilbert ID. Following the process of key-value pair insertion that is described in Section 1.2.3, each Hilbert key-value pair is able to find a node to store itself. For example, between Node 15 and Node 32 on the Chord Ring, the Hilbert key-value pairs for machines M57, M52, M63, M9, M0 and M15 are placed. Therefore, as the successor node of the Hilbert IDs of these Hilbert key-value pairs, Node 32 is responsible to store these Hilbert key-value pairs. At the same time, Node 32 is also these machines’ rendezvous node. The stored Hilbert key-value pairs in each node construct the main part of the Hilbert table of the node. Each machine reinserts its Hilbert key-value pair periodically.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Machine name</th>
<th>Chord ID</th>
<th>RTT to Landmark 1 (µs)</th>
<th>RTT to Landmark 2 (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1</td>
<td>M0</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>M9</td>
<td>9</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>M15</td>
<td>15</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>M52</td>
<td>52</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>M57</td>
<td>57</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>M63</td>
<td>63</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>M5</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>M32</td>
<td>32</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>M41</td>
<td>41</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>M49</td>
<td>49</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>M55</td>
<td>55</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.8: The 11 machines’ Chord ID, RTTs to Landmark 1 and Landmark 2.
in order to prevent loss of its rendezvous node due the node’s disconnection from the system. For instance, the disappearance of Node 9 leads to two machines, M41 and M49, losing their rendezvous node, namely Node 9. The periodic Hilbert key-value pair reinsertion of M41 and M49 inserts their Hilbert key-value pairs into Node 15, their new rendezvous node. Because the machines are in 2 clusters, the Hilbert key-value pairs are not evenly distributed on the Chord Ring like normal key-value pairs. In Figure 3.8, Node 32 stores the Hilbert key-value pairs of all 6 machines from Cluster
1. As for Cluster 2, Node 9 has two Hilbert key-value pairs, and Node 5, Node 57 and Node 63 have only one Hilbert key-value pair in the main part of their Hilbert tables.

Therefore, the successor part and the predecessor part of the Hilbert table in each rendezvous node should be populated in order to make each rendezvous node have sufficient Hilbert key-value pairs for the storage load-balancing approach described in
Section 3.4. In this small world example with 11 machines, the capacity limit, $C_{part}$ is set to be 2 for the successor part and the predecessor part. Therefore, Figure 3.9 demonstrates the successor part and the predecessor part of the Hilbert table for all nodes. The Hilbert key-value pairs in dark boxes are the main part of the Hilbert table of these nodes. The boxes above and below the main part are the predecessor part and
successor part of the Hilbert table, respectively. As for Node 15, there are no entries in the main part of its Hilbert table. The entries of the predecessor part, \( H6-M49 \) and \( H7-M41 \), in Node 15’s Hilbert table come from the main part of the Hilbert table of Node 15’s predecessor, Node 9. The successor part of Node 15’s Hilbert table comes from its successor Node 32. After several iterations of the periodic update operation on the successor part and the predecessor part of these nodes’ Hilbert tables, the Hilbert tables should be as shown in Figure 3.9.

After the Hilbert table is constructed and populated properly in each machine, the Hilbert tables are able to provide machines’ proximity information to the load-balancing approaches which are described next.

3.4 Proximity-aware Virtual Node Storage Load-balancing

Recall that key-value pairs migration is the most direct and efficient method to relieve the storage overload of some machines in the Chord-based storage system, according to the analysis in Section 3.2.1, on Page 65. This section presents a new proposed storage load-balancing approach, P_VNode, that is based on the virtual node approach which is discussed in Section 2.3.1. Combining with the Hilbert curve approach introduced in Section 3.3, a dynamic storage load-balancing approach with consideration of network proximity information is constructed.

3.4.1 Features of the Load-balancing Approach

During the design process for this storage load-balancing approach for a Chord-based storage system, the following three features have been considered: 1) A dynamic storage load-balancing approach, 2) The consideration of network utilisation and 3) The utilisation of the existing Chord operations.

According to the description of DHT’s natural load-balancing (in Section 2.2.3) and the original virtual node approach (in Section 2.3.1) in Chapter 2, the storage load-balancing achieved by these two methods is a uniform distribution of the number of key-value pairs across all machines. There are two problems with this type of passive load-balancing for a Chord-based storage system. Firstly, the number of key-value pairs to insert is considered instead of the used storage space of the key-value pairs in each machine. Secondly, these two methods have no solutions to the situation when
storage overload has already happened in some machines. Therefore, the proposed P_VNode load-balancing approach is a dynamic load-balancing approach to deal with the storage overload situation and uses the storage utilisation to determine each machine’s storage status as described in Section 3.2.1. Each machine is warned when its storage utilisation reaches the X% that is a predefined global setting. The discussion of this setting, X%, is given in Section 3.4.3. A virtual node migration algorithm that is explained in Section 3.4.2 is employed by P_VNode to relieve the storage overloaded machines.

According to the analysis in Section 3.2.1, it is inevitable to increase the network utilisation due to migration operations among different machines. Nevertheless, data migration is the most intuitive and efficient means to ease the storage overload in a Chord-based storage system. In this case, the central issue for the P_VNode approach is the reduction of cost for virtual node migration, which relates to time consumption for the migration operation. The cost of migration in the Chord-based storage system can be affected by three aspects, namely migration data size, RTT between the two migration machines and busyness of the two migration machines. The choice of the smallest feasible virtual node in an overloaded machine makes the migration data size as small as possible. The Hilbert curve approach employed in the P_VNode scheme chooses a ‘nearest’ destination machine for the storage overloaded machine according to their Hilbert IDs, which potentially reduces the RTT between the two migration machines. Therefore, the P_VNode load-balancing approach naturally reduces these two aspects of the migration cost. For the third aspect, busyness of machines, the P_VNode scheme has not taken this into account.

Chord has already provided a number of advisable features, which include scalability, provable correctness, resilience to node failures, etc. [50]. Therefore, there are a number of existing maintaining operations in Chord to support these features. Maximising the utilisation of existing Chord operations is better than introducing new operations into Chord for the proposed storage load-balancing approach. The P_VNode scheme exploits the existing operations of Chord in two ways, namely the migration operation based on each individual virtual node and the introduction of the Hilbert table to each node. Because Chord well implements the node joining and node leaving operations, transferring a whole virtual node by a leaving-rejoining method is easier to implement than maintaining a large volume of information of the migrated key-value pairs in the Chord-based storage system. The P_VNode introduces a Hilbert table into each node of the Chord-based storage system. Similar to the other data structures in
a Chord’s node, such as successor pointer and finger table, the Hilbert table also requires periodic update. Accordingly, the update messages for the Hilbert table can be piggy-backed on the maintaining messages for other data structures.

### 3.4.2 Migration Algorithm

The essence of the P_VNode load-balancing approach to relieve the overloaded machines is to migrate a virtual node from the storage overloaded machine to another less loaded machine. In the storage overloaded machine, the central operation is the virtual node migration that starts from the CHECKING_STORAGE_UTILISATION procedure which is invoked before any I/O (Input or Output) operations on the local storage space in the machine. If the storage utilisation of a machine is bigger than $X\%$, this machine is determined to be a storage overloaded machine by itself. Two machines and a node, the storage overloaded machine ($EM_o$), the migration target machine ($EM_t$) and the rendezvous node of $EM_o$ ($N_r$), are involved in the migration operation of P_VNode as demonstrated in Algorithm 3.1, Algorithm 3.3 and Algorithm 3.2, respectively.

In the storage overloaded machine, $EM_o$, there are two tasks to be accomplished before the virtual node transferring, namely to find an appropriate virtual node to transfer and to find an appropriate machine to accept the virtual node.

The first task is defined as the $\text{find\_least\_loaded\_vnode}(\cdot)$ procedure, which can be found in line 11 in Algorithm 3.1. Its objective is to find a virtual node $VN_l$ in $EM_o$ with the smallest storage space usage whose migration is able to relieve $EM_o$’s storage overload (i.e. gets storage utilisation below $X\%$). If an individual virtual node migration cannot take the storage overloaded machine’s storage utilisation below $X\%$, the virtual node with the heaviest storage load is chosen to migrate (line 21 in Algorithm 3.1). The CHECKING_STORAGE_UTILISATION procedure will be invoked again before the next I/O operation on the local storage space in order to find another chance to take the machine’s storage utilisation below $X\%$. This repeat call to CHECKING_STORAGE_UTILISATION procedure will not be stopped until either the machine’s storage utilisation is below $X\%$ or no migration target machine can be found.

The second task is accomplished in two parts. The $EM_o$’s rendezvous node, $N_r$, is firstly found by the $\text{find\_rendezvous\_node}(\text{id})$ procedure (lines 24 - 28 in Algorithm 3.1), which uses the $\text{find\_successor}(\text{id})$ function provided by Chord. A remote query to $N_r$’s Hilbert table returns an appropriate machine as the migration target.

In $EM_o$’s rendezvous node, $N_r$, the $\text{find\_nearest\_machine}(\text{id}, \text{virtual\_node\_size})$
Algorithm 3.1 Overloaded machine: $EM_o$

1: procedure CHECKING_STORAGE_UTILISATION( )
2:     if $EM_o.storage\_utilisation \geq X\%$ then
3:         $VN_l \leftarrow EM_o.find\_least\_loaded\_vnode( )$
4:         $N_r \leftarrow EM_o.find\_rendezvous\_node(Hibert\_ID)$
5:         $EM_t \leftarrow N_r.find\_nearest\_machine(Hibert\_ID, VN_l.size)$
6:             if $EM_t \neq NULL$ then
7:                 $EM_o.transfer\_overloaded\_vnode(VN_l, EM_t)$
8:         end if
9:     end if
10: end procedure

11: procedure find\_least\_loaded\_vnode( )
12:     sorted\_vn\_list $\leftarrow$ SORT\_BY\_SIZE($EM_o.virtual\_node\_list$)
13:         $\triangleright$ sorted\_vn\_list is sorted in ascending order of used storage space.
14:     for all $vn \in$ sorted\_vn\_list do
15:         $temp\_util \leftarrow (EM_o.used\_storage\_space - vn.size) / EM_o.storage\_quota$
16:         if $temp\_util < X\%$ then
17:             return $vn$
18:             $\triangleright$ The returned $vn$ is the smallest that will fit into the target’s storage space.
19:         end if
20:     end for
21:     return $vn$
22:     $\triangleright$ The returned $vn$ is the virtual node with the heaviest storage load.
23: end procedure

24: procedure find\_rendezvous\_node(id)
25:     rendezvous\_node $\leftarrow EM_o.find\_successor(id)$
26:     $\triangleright$ find\_successor(id) procedure is provided by Chord.
27:     return rendezvous\_node
28: end procedure

29: procedure transfer\_overloaded\_vnode(virtual\_node, machine)
30:     $EM_o.virtual\_node.leave( )$
31:     $EM_o.notify\_machine\_migration(machine)$
32:     machine.virtual\_node.join(virtual\_node\_Node\_ID)
33:     $EM_o.transfer\_virtual\_node(virtual\_node, machine)$
34: end procedure
Algorithm 3.2 Rendezvous node: $N_r$

1: procedure find_nearest_machine(id, virtual_node_size)
2:     machine_distance_dict ← NULL
3:     for all machine ∈ $N_r$.Hilbert_Table do
4:         distance ← |machine.Hilbert_ID − id|
5:         machine_distance_dict.insert(machine, distance)
6:     end for
7:     sorted_machine_list ← SORT_BY_DISTANCE(machine_distance_dict)
8:     \[\triangleright\] The machine list, sorted_machine_list, is in ascending order.
9:     for all machine ∈ sorted_machine_list do
10:        ready ← machine.check_storage_space(virtual_node_size)
11:        if ready = EM$_t$.has_sufficient_storage then
12:            return machine
13:        end if
14:     end for
15: return NULL
16: end procedure

Algorithm 3.3 Target machine: EM$_t$

1: procedure check_storage_space(size)
2:     temp_utilisation ← (EM$_t$.used_storage_space + size)/EM$_t$.storage_quota
3:     if temp_utilisation ≥ X\% then
4:         return EM$_t$.has_sufficient_space
5:     else
6:         return EM$_t$.has_insufficient_space
7:     end if
8: end procedure
procedure is invoked when the remote query of its Hilbert table arrives from $EM_o$. The $\text{find\_nearest\_machine}(id, \text{virtual\_node\_size})$ procedure is defined in Algorithm 3.2. There are two criteria to find the ‘nearest’ machine for $EM_o$:

1. The absolute difference between the machine’s Hilbert ID and $EM_o$’s Hilbert ID is sorted in ascending order. The machines with smaller difference are considered early.

2. The machine’s storage space cannot be overloaded by the virtual node’s migration. This is the reason why the used storage space of the virtual node is provided as a parameter of the procedure of $\text{find\_nearest\_machine}(id, \text{virtual\_node\_size})$.

The checking for the second criteria is accomplished by a remote procedure, $\text{check\_storage\_space}(\text{size})$ (in Algorithm 3.3), on a migrate target machine, $EM_t$. If all the machines in $N_r$’s Hilbert table have insufficient storage space to accept the virtual node, the virtual node migration is terminated.

Finally, after both the virtual node to transfer and the migration target machine are found, the virtual node migration operation can be performed. The procedure $\text{transfer\_overloaded\_vnode}(\text{virtual\_node, machine})$ is defined by the pseudo code at lines 29 - 34 in Algorithm 3.1. Four operations are involved in this procedure:

1. $EM_o$ stops all operations on the virtual node that is being transferred and makes it leave the Chord Ring. However, this process is different from the node leaving operation, because the key-value pairs of the virtual node are not transferred to the virtual node’s successor as described in Section 1.2.2.

2. $EM_o$ notifies the migration target machine, $EM_t$, to be prepared for the virtual node transferring.

3. The migration target machine, $EM_t$, starts a virtual node with the same Node ID as the virtual node being transferred from $EM_o$. The join operation of Chord facilitates the newly created virtual node to join the Chord Ring.

4. $EM_o$ starts to transfer the key-value pair in the original virtual node to $EM_t$.

The reason why a leaving-rejoining method is used in virtual node migration is that the migrated virtual node needs to update its network information, such as the IP address of the machine containing it. After the leaving-rejoining process, the Chord’s node leaving and joining operations help the virtual node to update its network information.
CHAPTER 3. PROXIMITY-AWARE LOAD-BALANCING APPROACH

across the whole system. After these operations, the whole process of the virtual node migration is completed.

After the P_VNode load-balancing approach is deployed in the Chord-based storage system, the original file operations that are defined in Section 1.3.2 do not need to be updated, because the virtual node migration is based on each individual virtual node. Even if a virtual node is migrated to another machine, the operation message of the key-value pair that should be stored in this virtual node is still routed to the same virtual node, because the virtual node’s Node ID is not changed. The only difference is that the virtual node is in a different machine. After the leaving-rejoining process, the virtual node’s network information is updated. Therefore, the file operations of the Chord-based storage system are not affected.

3.4.3 Running Time of the Load-balancing Approach

Recall that both FWTP and AWTP are defined by a X% storage utilisation as described in Section 3.2.1. In this research, X is set to be 80, which is an initial moderate value as a starting point. All experimental results presented in Chapter 5 are based on the 80% threshold setting value. For each machine in a Chord-based storage system, the threshold of 80% is a check point to determine whether the machine itself is a potential storage overloaded machine.

According to the description of the P_Vnode scheme, it starts to work from FWTP_{80%} until AWTP_{80%}. FWTP_{80%} is the start time point for the P_Vnode load-balancing approach. If all machines’ storage utilisation exceed 80%, the P_Vnode scheme has no place to migrate virtual nodes. Therefore, AWTP_{80%} is the time point when the P_Vnode scheme stops working, theoretically. Accordingly, the P_VNode scheme theoretically works between FWTP_{X%} and AWTP_{X%}. If X is set to be a smaller value than 80, although P_Vnode could start to work earlier, it could also stop earlier.

In the theoretical world, due to the P_Vnode scheme’s migration, UTP should rarely emerge from FWTP_{80%} to AWTP_{80%}, which ensures that the P_Vnode scheme takes the whole system’s storage utilisation towards 80% before it stops. However, in a practical situation, the P_VNode may become useless much earlier than AWTP_{80%}, because UTP emerges before AWTP_{80%}. Two reasons contribute to this situation, namely 1) the migration operation based on each virtual node and 2) the Hilbert table with a limited capacity. If all virtual nodes in the storage overloaded machine, EM_o have large storage space usage, it is difficult for the $N_r.find.nearest.machine(Hilbert.ID,VN_l.size)$ procedure (in line 5 of Algorithm 3.1)
to return an appropriate migration target machine, because the candidate machines in
$N_r$’s Hilbert table have insufficient space to accept these large virtual nodes. This sit-
uation happens in the experiments for this research, because the Hilbert table only
clusters a limited number of machines. The Hilbert curve’s clustering is not a global
operation on all machines in the Chord-based storage system. The experimental results
are presented and discussed in Section 5.3.2.

3.5 Suggestion for a Network Load-balancing Approach

The network load-balancing is different from the storage load-balancing. The objec-
tive of the P_Vnode load-balancing approach is to find an appropriate machine for
virtual node migration. Only one machine is needed to accomplish the P_Vnode load-
balancing operation. However, a network load-balancing approach needs to find $2 \times C_p$
other machines for replications, where $C_p$ is the capacity of a Hilbert table’s succes-
sor part and predecessor part. In the case that some key-value pairs are extremely
popular, more than $2 \times C_p$ replications might be required to prevent the Chord-based
storage system from being overwhelmed by a large number of key-value pair retrieval
requests. This section introduces a suggestion for a network load-balancing approach
in the Chord-based storage system.

There was insufficient time to implement this suggestion, so no experiments have
been performed to assess how well it might work.

3.5.1 Replication Table

A network load-balancing approach should not modify any Chord operations. Even if
a key-value pair (KV) in machine $EM_o$ is replicated into several other nodes in other
machines, which are called the KV’s replication machines, the key-value pair retrieval
requests for KV still reach $EM_o$ which is called the KV’s owner machine. Then ma-
chine $EM_o$ decides an appropriate replication of KV to which a retrieval request is
redirected. Therefore, each machine requires a replication table to record replication
locations of the replicated key-value pairs stored in the node or nodes in this machine.
For each replicated key-value pair, the IP addresses of machines containing its replica-
tion are recorded in a replication table.

A machine uses this replication table to manage the replications of the key-value
pairs stored in it. Firstly, an expiration time is given to each replication of a key-value
pair in order to prevent unnecessary storage usage for unused replications, because a key-value pair is replicated to $2 \times C_P$ other machines. Therefore, an unused key-value pair replication is deleted from the replication machine and its entry in the owner’s replication table is also deleted. Secondly, because a machine redirects key-value pair retrieval requests according to the distance between the request machine’s Hilbert ID and the Hilbert ID of the machine with replication, some replications may be ‘far’ away from the retrieval request machines. These ‘far’ replications are rarely used as the targets of request redirections. Finally, if replication is frequently requested, its replication machine is notified to keep this replication and reset the expiration timer. So a machine is periodically enquired for replication deletion by replication machines in which the key-value pair replication timer has expired. This expiration mechanism can also prevent unmanageable replications if the key-value pair’s owner machine fails or departs from the system.

### 3.5.2 Changed Key-value Pair Retrieval

Due to the introduction of the network load-balancing approach, the file retrieval operation needs to be changed in the Chord-based storage system. Because each file retrieval operation consists of a group of key-value pair retrieval operations, as described in Section 1.3.2, the only changed Chord operation is the key-value pair retrieval operation.

If the key-value pair $KV$ is replicated to other machines, the operations before a retrieval request reaches $KV$’s owner machine are all the same as the Chord-based storage system without the network load-balancing approach. In the original Chord-based storage system, the key-value pair is sent to its request machine at this time. In the system with the network load-balancing approach, however, $KV$’s owner machine compares its request machine’s Hilbert ID to the Hilbert IDs of all $KV$’s replication machines and finds the ‘nearest’ replication machine’s Hilbert ID to the request machine’s Hilbert ID. The $KV$ retrieval request is redirected to the ‘nearest’ replication machine, which then transfers $KV$’s replication directly to its request machine. Therefore, the $KV$’s owner machine always redirects requests to their ‘nearest’ replication machines according to the requesting machines’ Hilbert ID.
3.5.3 Multiple-layer Replications

A question is raised when a key-value pair is replicated to $2 \times C_p$ other machines: is $2 \times C_p$ replications enough for all situations? The answer is negative. The fixed number of replications obviously cannot fulfil all situations; for example, an extremely popular key-value pair could overwhelm its replication machines. Therefore a multiple-layer replication is allowed in the network load-balancing approach, which makes the key-value pair’s replication machine further replicate its replication to other machines. In this case, theoretically, a key-value pair can be replicated to $(2 \times C_p)^{LR}$ machines, where $LR$ is the number of allowed replication layers. If a 2-layer replication is set in the Chord-based storage system, for each key-value pair, there are theoretically $(2 \times C_p)^2 = 100$ replications, when $C_p = 5$ in the Chord-based storage system.

3.6 Summary

This chapter has investigated the storage space usage and the network utilisation of a Chord-based storage system from the perspective of its load-balancing. Using the storage space utilisation as the criterion in each machine, three time points, $FWTP$, $AWTP$ and $UTP$, are defined to evaluate the storage status of the Chord-based storage system.

1. $FWTP$ is the time when any machine reaches $X\%$ storage space utilisation in the system.

2. $AWTP$ is the time when all machines reach $X\%$ storage space utilisation in the system.

3. $UTP$ is the time when any machine reaches 100% storage space utilisation in the system.

The value $X$ is set to be 80 in this research as a starting point. Each machine determines itself to be a storage overloaded machine, if it has 100% storage space utilisation. After the $UTP$, the Chord-based storage system is unreliable because the storage overloaded machines have no capability to store more key-value pairs.

As a complete investigation, the network utilisation of the Chord-based storage system has also been discussed. The research in this thesis concentrates on the network utilisation of a single machine in the storage system. The network overload is defined by the emergence of overwhelmed machines caused by either frequent file operations.
or popular files. The overwhelmed machines are defined as hotspots in a Chord-based storage system. In order to reduce the number of hotspots in the storage system, a suggestion for the design of a network load-balancing approach is provided at the end of this chapter in Section 3.5.

In order to relieve a storage overloaded machine, the most efficient and intuitive method is key-value pair migration, which requires a cost to accomplish the network transferring between machines. The Hilbert curve approach has been proposed by this research in Section 3.3 to reduce the cost of key-value pair migration. The essence of the Hilbert curve approach is to cluster machines according to their network proximity information. The Hilbert curve is one of a number of Space Filling Curves (SFCs) that can achieve a similar distance preserving mapping. The Hilbert curve is chosen by using Spearman’s correlation coefficient of a ranking test that is designed by this research in order to compare the Hilbert curve against two other SFCs, the Peano curve and the RBG curve. A Hilbert ID and a Hilbert table are introduced into each node in the Chord-based storage system to accomplish the clustering of machines. A small world case study is provided in Section 3.3.4 in order to illustrate the Hilbert curve approach.

Finally, Section 3.4 presents the P_VNode storage load-balancing approach that combines the load-balancing techniques of the virtual node approach and network proximity information, as described in Section 2.3.1 and Section 2.3.2, respectively. Three features are accomplished by the P_VNode scheme:

1. A dynamic storage load-balancing approach;
2. The consideration of network utilisation;
3. The utilisation of the existing Chord operations.

In order to evaluate the work proposed in this chapter, the next chapter introduces the evaluation method and experimental approach that is used in this research.
Chapter 4

Experimental Approach

4.1 Introduction

The storage load-balancing approach, P_VNode for a Chord-based storage system has been developed in Chapter 3 using the Hilbert curve approach. In order to evaluate the P_VNode scheme, this research employs experiments in a distributed environment that is constructed by practical machines. For either P2P system designs or DHT developments, simulation in distinct P2P simulators is mostly employed to evaluate their validity. However, simulators simplify a system from real environments and this implies there may be unexpected factors affecting the system. Therefore, in this research, the original Chord-based storage system is deployed in a practical experimental environment for empirical observation and analysis. The proposed storage load-balancing approach, P_VNode, is also evaluated in the same environment.

Section 4.2 compares the differences between simulation method and empirical experimental method and chooses experiment as the main methodology. In Section 4.3, the experimental objectives are presented. In Section 4.4, the experimental environment is described together with the experimental settings of experimental machines, the experiment control system and the experiment monitoring system. Section 4.5 details the methods and metrics used to measure the load-balancing and other related characteristics in the Chord-based storage system. The experimental processes and scenarios are described in Section 4.6. Finally, the chapter is summarised in Section 4.7.
4.2 Experiment or Simulation?

This study has been verified by an experimental method, although this is not the only option for evaluation purposes. The other two possible evaluation methods are analytical analysis and simulation. Analytical analysis is a theoretical work to verify a system, which is usually adopted during the design phase when a system is under development. However, empirical testing cannot be replaced by theoretical analysis, because a great number of practical details are not modelled by analytical analysis, and some assumptions are also considered as preconditions. Stoica et al. [50] have theoretically already proven the feasibility of Chord as a DHT, which is the base of the distributed storage system in which P_VNode is embedded. Therefore, the only absent part of the evaluation of P_VNode is an empirical testing. Compared with the analytical evaluation, both simulation and experiment are empirical evaluation approaches within a practical environment.

In a simulation approach, the experimental environment is simulated according to a practical environment. A well designed simulation with well implemented simulator is able to model closely the real world environment. In a simulation, researchers concentrate on a small group of research objects in a system by ignoring the other aspects, which is two-sided. On the one hand, this simplification reduces the duration of the evaluation process due to less effort spent on other, irrelevant aspects of the system in the real world. The implementation period in a simulator is much shorter than that in a real world system, which is still sufficient for prototyping tasks. Additionally, more controls can be applied on the simulation without practical environmental limitations, for example, the number of simulated machines in the system and their topologies in the simulated network environment. A number of P2P simulators have been available [2], using which the number of simulated machines can be scaled up to 1,000,000 in a simulation in a single practical experimental machine [13]. On the other hand, some unanticipated components would be neglected by simulators in their simplified environments. There is no way to explore or to observe the unmodelled aspects of a simulated system if those system components are not taken into consideration. In other words, for further research on additional aspects of a system by simulation, more detailed corresponding models should be introduced, which also may affect the existing components of the simulation. For example in the simulator, p2psim [15], which has been developed to validate Chord and is presented in the paper publishing Chord, a number of DHTs are implemented, such as Kademlia [29], Tapestry [57] etc. Nonetheless, the latest version of this simulator, p2psim-0.3 is not appropriate for the research
reported here without modification, for the following reasons:

1. The Chord ID space of p2psim is different from that of the Chord’s implementation for experiment. The Chord ID space in the simulator is defined as a primitive C type `unsigned long long`, which is $[0, 2^{64})$ in the machine running the simulator. However, in most practical cases, the Chord ID space is $[0, 2^{160})$ using SHA-1.

2. The research presented in this thesis focuses on the storage load-balancing of the Chord-based storage system. However, p2psim has not implemented any file operations, file insertion or file retrieval, although a number of other DHT protocols are provided by p2psim-0.3.

3. p2psim does simulate the network data flow and even network burst in its latest version. But an important component, the network queue, is not modelled in it. Section 5.4.2 presents the importance of the network queue in the hotspot detection approach.

Moreover, the models used to simulate the practical processes during a simulation are also required to be verified, in order to ensure a validated simulation being carried out.

The last option to evaluate a P2P approach is by experiment on practical machines in the real world, which is adopted by the research in this thesis. However, the following disadvantages of the experimental method prevent other P2P research from employing it.

Firstly, the duration of the evaluation process using experiments would be prolonged by both the system’s development and the experiment control system’s development, compared with the simulation method. An executable implementation of a Chord-based storage system for an experiment requires more work than its implementation in a specific simulator with a simplified environment. An experiment also requires a sophisticated experimental platform in order to control, instruct and monitor the Experimental Machines (EMs), and fetch experimental results from them. If a cross-platform experiment is designed to evaluate a storage system on distinct platforms, more effort is required for the cross-platform consideration of both the storage system’s development and the experiment control system’s development. Therefore, the experimental method has a higher development cost than the simulation method. More details about the control mechanism used in the experiment of this research will be presented in Section 4.4.
Secondly, the number of EMs is limited and the topology of them cannot be changed easily. Chord was evaluated on a network testbed called PlanetLab [37], which had only 300 nodes around the world when Chord was published in 2003. As of 2014, there are 1188 PlanetLab nodes [37], which is significantly fewer than the 1,000,000 machines that has been achieved in a simulation.

Finally, the experimental method could be subject to interference by unexpected and irrelevant network operations. A number of practical machines are required in an experiment, which is expensive for this single purpose usage. Therefore, network testbeds such as PlanetLab or multi-purpose computer clusters are usually employed.

The experimental method of the study in this thesis exploits idle times of machines in students’ laboratories in the School of Computer Science in Manchester University. However, other users of laboratories sometimes interrupt machines, which then affects the experimental results.

The experimental method is employed in the research in this thesis despite the disadvantages listed above. Investigation and evaluation are the two roles of experiments in this research. The load-balancing of a Chord-based storage system should be investigated in the real world, because its behaviour is different from that of a conventional distributed storage system. Observing the Chord-based storage system in designed experiments is an appropriately intuitive method to accomplish the investigation task. Furthermore, the available resources also determine that the experimental method is adopted. On the one hand, an acceptable experimental environment is available, which consists of five student laboratories of the School of Computer Science in the University of Manchester. The Local Area Network (LAN) network connecting these five laboratories and the machines in the laboratories construct a cluster environment. On the other hand, since the implementation of a Chord-based storage system called DHash [10] is available, the development cost to implement Chord-based storage system has been significantly reduced. Moreover, this research concentrates on the load-balancing of the Chord-based storage system in a cluster environment. Machine’s loads, such as network traffic and storage space usage, are the main research targets, which are quite expensive to simulate, especially the network traffic between machines. As for the simulation method employed by DHTs, the simulations focus on the routing algorithm and system dynamism, which are more conveniently and easily validated by simulations.

As a result of all these consideration, the experimental method has been chosen to investigate the load-balancing of the Chord-based storage system in the remainder of
this thesis.

4.3 Experimental Objectives

Experimentation in the practical cluster environment is employed as the main evaluation methodology for the research presented in this thesis. Experimental objectives are needed for the experiments.

The primary experimental objective is to evaluate the P_VNode storage load-balancing approach by demonstrating the improvement of the storage performance of the Chord-based storage system with P_VNode, compared with the one without P_VNode. This experimental objective makes the experiment concentrate on the storage load-balancing of the Chord-based storage system, which, in turn, affects other aspects of the system, such as the storage distribution across all EMs, the system’s reliability and the system’s storage capacity. Moreover, the storage load-balancing may also affect the network utilisation of the Chord-based storage system.

The first secondary experimental objective is to investigate the network utilisation of the Chord-based storage system, which can be used to make some progress on network load-balancing issues in future work. Two tasks are involved in this experimental objective, namely, determination of the system’s operational capacity and the detection of hotspots. The details of how this experimental objective is accomplished are described in Section 5.4.

The second secondary experimental objective is the investigation of the original Chord-based storage system, which provides initial insight into the storage system. The targets of this investigation include CPU running time, memory usage and RPC messages transferred in the system. This investigation is able to plot a picture of the Chord-based storage system running in normal conditions and to inspire the design of the P_VNode storage load-balancing approach and the network load-balancing approach in future work.

In order to achieve these experimental objectives, each experiment is carried out according to the environment and settings described in the next section.
4.4 Experimental Environment and Settings

4.4.1 Network Environment

The experiments on the Chord-based storage system and the proposed load-balancing approach have been carried out in five student laboratories of the School of Computer Science at the University of Manchester. These five laboratories have over 300 machines which are connected to a LAN on a 100Mbit/s full duplex ethernet. Although these machines are accessible from outside networks through the Internet, the experiments for this research are limited within the LAN. There is no network traffic from the experiments towards the outside of the LAN. Because these student laboratories are used for teaching purposes in the daytime, all experiments for this research are performed from 21:00pm to 07:00am of the next morning. Even at nighttime, however, there are still some machines that are used by students or remote users. In order to reduce interference from other users of this non-exclusive experimental environment, only those machines without other users are involved in each experiment. Therefore, from over 300 machines in all five laboratories, each experiment chooses 150 participating machines, which are called the EMs. This minimises the possibility of interruptions from other laboratory users during experiments at night.

However, there are still a number of invalid experiments due to interference from other users. During one night’s experiment, its result could be affected by other users’ login, because extra network traffic could be produced. A more serious situation is that some EMs could disappear from the network when they are shutdown by other users. An experiment control system that is described in Section 4.4.3 is able to detect these interferences during experiments. If either of these interferences happen, the experimental results could be unreliable, which then makes the experiment invalid. Furthermore, the experimental results from a local network environment merely reflect the situation in the cluster environment where the experiments are performed. If the storage system is deployed on the Internet, the experiment could produce a different set of results which might not be the same as those shown in Chapter 5. As a result, the discussion of the Chord-based storage system and its load-balancing issues in this thesis is based on this cluster environment.
4.4.2 Experimental Machines

In machines of the student laboratories, both Windows 7 and Fedora 14 with 2.6 Linux kernel are installed in the same hard drive. Because Linux provides a number of versatile facilities to access the underlying network stack in its open source kernel, the experiment is deployed in the Linux system of each EM. The method to monitor an EM is detailed in Section 4.4.3. In Section 3.2.1, each EM’s storage space is divided into system storage space and Chord storage space. In this context, the storage space for Windows is in the system storage space. The whole Linux file system is partitioned into system storage space and Chord storage space. Due to the lack of administrator privilege on the machines in the student laboratories, the only location available for this research’s experiment is the /tmp directory where both the Chord-based storage system and the Chord storage space are placed. Table 4.1 lists the five laboratories’ minimum size of available storage space of the file system where the /tmp directory is mounted.

According to Table 4.1, machines in the UF and GFPC laboratories have much less available storage space in the /tmp directory’s file system, due to the smaller size of hard disk installed, than machines in the other three laboratories: LF, FF and ENG. In order to prevent the whole Linux system’s fatal crash caused by overloaded storage space, a storage quota of 135GB is given to the Chord storage space of each EM. Therefore, the experiment system treats EMs with over 135GB Chord storage space usage as storage overloaded EMs.

In each EM, the Chord-based storage system, DHash, was running during experiments. DHash is an implementation of a storage system based on Chord, which was published by Dabek et al. [10]. Remote Procedure Call (RPC) is employed by DHash to accomplish the network communication between EMs in the LAN. Due to the existing implementation of Chord-based storage system, namely DHash, the length of the development process for the experiments was reduced significantly. DHash’s running

<table>
<thead>
<tr>
<th>Laboratory</th>
<th>Number of Machines</th>
<th>Minimum Storage Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>70</td>
<td>227 GB</td>
</tr>
<tr>
<td>FF</td>
<td>49</td>
<td>227 GB</td>
</tr>
<tr>
<td>ENG</td>
<td>42</td>
<td>227 GB</td>
</tr>
<tr>
<td>UF</td>
<td>70</td>
<td>153 GB</td>
</tr>
<tr>
<td>GFPC</td>
<td>80</td>
<td>153 GB</td>
</tr>
</tbody>
</table>

Table 4.1: Minimum size of available storage space of file system containing /tmp directory and the number of machines in the five experiment laboratories.


requires three extra packages which are summarised in Table 4.2.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Version</th>
<th>Usage Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMP</td>
<td>4.3.2</td>
<td>GMP provides operations for big numbers, such as 160 bit Chord ID.</td>
</tr>
<tr>
<td>SFSLite</td>
<td>0.8.17</td>
<td>SFSLite provides asynchronous RPC communication between machines [28].</td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>4.4.20</td>
<td>Berkeley DB locally maintains key-value pairs stored in each machine.</td>
</tr>
</tbody>
</table>

Table 4.2: Information on DHash’s three dependent packages.

Each EM has its own DHash which is running separately from other EMs during the experiments. Therefore, in order to instruct EMs before an experiment and to align experimental results of all EMs according to the time point, time synchronisation needs to be performed across all EMs. In the experiments, the time synchronisation is handled by NTP (Network Time Protocol) [30]. All EMs subscribe the real time from a NTP server and then adjust their system clocks according to the NTP. According to one paper on NTP [31], the average offset using NTP is 8.2ms with median 1.8ms and 18ms standard deviation. In this case, the time synchronisation using NTP is accurate enough for the experiment, because the time resolution for experiment results is one second. The time synchronisation ensures that the experiment control system (see below) works accurately, since the time-labelled experiment instructions are sent to each EM before the experiments begin.

### 4.4.3 Experiment Control System

In each EM, a daemon program is running in the background to communicate with a control machine which is not in any of the designated student laboratories. The control machine and the daemon program in the EMs construct an experiment control system that accomplishes three functions: pre-experiment preparation, experiment instruction distribution, and post-experiment operations.

#### 4.4.3.1 Pre-experiment Preparation

The control system filters out some machines and chooses 150 EMs from the five student laboratories before each experiment. In the student laboratories, some machines are not appropriate to be involved into an experiment due to three main reasons:
1. The machines with other users working on them.

2. The machines without the experiment control system.

3. The machines without sufficient storage space for the experiment.

### 4.4.3.2 Experimental Instruction Distribution

For each experiment, the control machine distributes the experimental instructions to all EMs. The experimental start time and end time for each EM is distributed by the control machine before the experiment, in order to ensure that the experiments start at the same time in all EMs. The experimental settings are distributed to every EM at the same, because the control machine does not contact with any EMs during an experiment in order to minimise the interference from this type of control network traffic. More details of the experimental settings are explained in Section 4.4.5.

### 4.4.3.3 Post-experiment Operations

Three operations are performed by the experiment control system after each experiment:  
1) clean up operations that restore EMs to their status before the experiment;  
2) collection operations that retrieve experimental results from EMs for further processing;  
3) validation operations that label the validity of the experimental results for each EM. The control machine retrieves the experimental results that are produced by each machine’s experiment monitoring system as described in Section 4.4.4. After the results have been retrieved, the daemon program in each EM deletes all files produced by the experiment. Then EMs are restored to the same status as before the experiment started so that they are ready for other users. Due to the existence of interference from other users as described in Section 4.4.1, the daemon program records these interferences for each EM. These records are used by the experiment control system to label the validity of an EM’s experimental result, which ensures that invalid experimental results will not be forwarded for further processing.

### 4.4.4 Experiment Monitoring System

There are also four experiment monitoring programs that independently collect the raw data of the experimental results in each EM. In order to prevent any interference from irrelevant network traffic produced by the monitoring programs, the control machine
collects the monitored data as the experimental results after the end of each experiment. The time point alignment across the experimental results of all EMs is provided by NTP as mentioned in Section 4.4.2, on Page 106. By this method, the system-wide experimental results are available in the control machine after each experiment’s running. In each EM, a 2GB temporary storage space is given to its monitoring programs to store the monitored data during each experiment’s running.

In each EM, there are four experiment monitoring programs running independently to record the different attributes. Each experiment monitoring program outputs its recorded raw data into a file. The following is a list of the four monitoring programs and their monitoring targets:

**cm_resource** program is used to monitor the CPU running time and the memory usage of the Chord-based storage system within a specific time interval. CPU running time and memory usage information come respectively from the file `/proc/[pid]/stat` and file `/proc/[pid]/statm` in the `/proc` pseudo-file system which is available in the Linux distribution, Fedora 14.

**net_global_resource** program is used to monitor the inbound and outbound network traffic through the network interface of each EM within a specific time interval.

**net_resource** program is used to monitor the RPC messages sent out from each EM within a specific time interval.

**dbcheck** program is used to periodically record the storage usage by Berkeley DB which is a local database for storing and managing all key-value pairs in an EM.

These four monitoring programs monitor the Chord-based system by actively checking the corresponding targets every five seconds. This is achieved by defining a **monitoring resolution** of five seconds, which is set in all the experiments.

In each EM, the monitored data is unprocessed raw data which has significance for a single machine. However, in order to reveal the system-wide attributes of the Chord-based storage system, further data processing is required after all EMs’ experimental results have been collected together. Section 4.5 presents all the measurement metrics, which are produced from the monitored raw data of the experiment monitoring system.

### 4.4.5 Experimental Settings

In order to reduce the interference caused by the network traffic of experiment control, each EM obtains most of its experimental settings before the start of the experiment.
There are also other experimental settings that are produced by each EM during the experiments’ running. This section presents all experimental settings required by the experiments on the Chord-based storage system.

Firstly, the experiment start time and end time are given to each EM, which ensure that each experiment has at least 10 hours running time during each night. Each EM refers to its system clock to start the experiment according to the instructed start time. Because all EMs’ system clocks are synchronised through the same NTP server, EMs are able to start and to end their experiments at the same time, independently.

Secondly, an experiment requires four Random Number Generators (RNGs), which are listed as following:

1. The experimental file content RNG is used to produce the experiment files with random content.

2. The experimental file size RNG is employed by a normal-distributed number generator to produce the experiment files whose size follows a normal distribution.

3. The file insertion RNG uses a Poission-distributed number generator to produce the start time points of file insertion operations in an experiment.

4. The file retrieval RNG uses a Poission-distributed number generator to produce the start time points of file retrieval operations in an experiment.

The random seeds (Random_Seed) for these RNGs are produced from the EM’s hostname and a timestamp:

\[
\text{Random}\_\text{Seed} = \sum_{i=1}^{n} \text{ASCII}(h_{\text{char}_i}) + \text{current}\_\text{time} \tag{4.1}
\]

where \(\text{ASCII}(h_{\text{char}_i})\) is the ASCII number of the \(i\)-th character of the EM’s hostname that has \(n\) characters and \(\text{current}\_\text{time}\) is the current system time in microseconds. In order to reproduce the same experiment, all random seeds in each EM are recorded. Although these random seeds are usually produced in each EM independently at the beginning of the experiment, each EM’s random seeds can also be assigned by the experiment control system together with other experimental settings before the start of the experiment.

Thirdly, as described in Section 4.4.2, each EM’s storage space quota for an experiment is set to be 135GB, which is sent to every EM within the experimental instruction
distribution messages. During an experiment, an EM’s storage overloading is triggered by storage space usage that is over 135GB.

Then, besides the 135GB Chord storage space for each EM to store key-value pairs of the Chord-based storage system, an extra 10GB temporary file storage space in each EM is required by the experiment to produce experimental files for file operations. Oldest produced experimental files in this buffer storage space are removed if new experimental files are required. The size of produced experimental files follows a normal distribution. More details of the experimental files’ production are described in Section 4.6.1.1.

As for the experimental settings of the Chord-based storage system, two factors are important in the experiment, namely the bootstrap machine and the number of virtual nodes. According to the introduction in Section 1.2.2, each joining EM requires a bootstrap machine that has already been in the Chord-based storage system during an experiment. One EM is chosen to be all other EM’s bootstrap machine, and this is set and is distributed to all EMs together with other experimental settings before an experiment starts. Because the machine joining operation is not the focus of this research, the bootstrap machine is chosen manually. The bootstrap machine starts its Chord program a minute earlier than other EMs in order to create a Chord-based storage system. When other EMs join in the storage system, the bootstrap machine is prepared. More details of the experiment process in each EM is presented in Section 4.6.1. In each EM, there could be a number of Chord nodes, each of which is called a virtual node as introduced in Section 2.3.1. Different numbers of virtual nodes in each EM are assigned in the different experiments as their settings. The experiments for this thesis set the same number of virtual nodes in all EMs before an experiment starts.

Finally, in order to maintain the Hilbert table mechanism in the storage system, a number of landmark machines are picked from all EMs according to the description in Section 3.3.

Each experiment is performed following a specific predefined scenario, in which different settings are required. More details of the experiment scenarios and the related experiment settings for each scenario are described in Section 4.6.

### 4.5 Measurement Metrics

This section presents a list of metrics that are used to evaluate the Chord-based storage system in the experiments. After all EM’s monitored raw data has been collected
together, the experimental results are processed according to the measurement metrics that are described in this section. These measurement metrics are divided into two types, namely single machine metrics and global metrics. The former metrics provide evaluations of the attributes for a single EM, whereas the latter metrics provide evaluations of the whole storage system’s attributes.

### 4.5.1 Single Machine Metrics

#### 4.5.1.1 CPU Usage Rate and Memory Usage

Because there are four processes for Chord-based storage system running, both CPU running time and memory usage come from the summation of all these running processes. Each process has its own directory (/proc/[pid]/) in the /proc pseudo-file system, which is labelled by its process ID ([pid]). For CPU running time, two entries are read from the file /proc/[pid]/stat, namely ‘utime’ for CPU time scheduled in user mode and ‘stime’ for CPU time scheduled in kernel mode. These two entries are measured in clock ticks, which is equal to 1/100 second in all EMs. The CPU usage rate \( U_{CPU} \) is evaluated by summing up the time duration of all clock ticks in both user mode and kernel mode during a fixed period of time.

\[
U_{CPU} = \frac{\sum_{i=1}^{4} (TU_i + TK_i)}{t_{val}} \times \frac{1}{100} \times 100\% \tag{4.2}
\]

where \( TU_i \) and \( TK_i \) are the number of user CPU ticks and kernel CPU ticks of the \( i \)-th process and \( t_{val} \) is each monitoring’s time duration. In the experiments, the used CPU clock ticks by all processes are sampled every 5 seconds, namely \( t_{val} = 5 \). As for memory usage, the entry ‘size’ in file /proc/[pid]/statm is read, which represents the memory usage by the relevant process at the current time point. The overall memory usage \( U_m \) is calculated as follows:

\[
U_m = \sum_{i=1}^{4} m_i \tag{4.3}
\]

where \( m_i \) is the memory usage of the \( i \)-th process in the EM.

#### 4.5.1.2 Network Utilisation

The measurement of network utilisation for each EM is much more complex than storage space usage or local computing resources, because the network implementation
component in EM’s Linux system, Fedora 14, is complex. In addition, since the whole experiment environment is constructed in student laboratories, privilege cannot be obtained on these machines to investigate every detail of the network components in their systems. Moreover, categorising the network traffic is also required in order to analyse the network utilisation by different types of messages during the Chord-based storage system running. Therefore, there are two measurement levels in each EM for its network utilisation.

The first level contains metrics that are measured from each EM’s operating system, Fedora 14. The metrics involved at this level are as follows:

**Network Transfer Rate**

Both the *sending network transfer rate* \( R_{\text{send}} \) and *receiving network transfer rate* \( R_{\text{recv}} \) are measured by periodically reading file `/proc/net/dev` for entries current sent bytes of data \( NS_t \) and current received bytes of data \( NR_t \). The network transfer rate represents the busyness of an EM during a specific period of time (from \( t \) to \( t + t_{\text{val}} \)).

\[
R_{\text{send}} = \frac{NS_{t+t_{\text{val}}} - NS_t}{t_{\text{val}}} \\
R_{\text{recv}} = \frac{NR_{t+t_{\text{val}}} - NS_t}{t_{\text{val}}}
\]

(4.4)

(4.5)

**TCP and UDP Queue**

In the Chord-based storage system, there are a number of established TCP connections for transferring key-value pairs. Each connection has a receive queue and a send queue. These are used to cache waiting to transport data (in the send queue) and unprocessed data (in the receive queue). In file `/proc/net/tcp` and file `/proc/net/udp`, all TCP sockets and UDP sockets in an EM are dumped, including their size. The TCP and UDP traffic of the Chord-based storage system can be extracted from these two files. Therefore, after filtering out other services network sockets, the total TCP queue size \( Q_{\text{TCP send}} \) and \( Q_{\text{TCP recv}} \), total UDP queue size \( Q_{\text{UDP send}} \) and \( Q_{\text{UDP recv}} \), the number of TCP connections \( n_t \) and the number of UDP sockets \( n_u \) are obtained:

\[
Q_{\text{TCP send}} = \sum_{i=1}^{n_t} S_i TCP_i
\]

(4.6)
These metrics indicate the network utilisation load on the network interface in an EM.

The second measurement level contains the metrics that are measured from the inside of the Chord-based storage system. In each EM, the network traffic of the Chord-based storage system is contributed to by two types of operational messages: data message and maintaining message. Data messages are the RPC messages containing key-value pairs, which are produced by the two file operations: file insertion and file retrieval. Maintaining messages are the essential RPC messages transferred in the Chord system to maintain the Chord Ring and each node’s data structures, such as the successor and predecessor pointers and the finger table as described in Section 1.2.2. The RPC message sending rate for the two types of RPC messages in EM\textsubscript{j} within a duration of \( t_{\text{val}} \) is defined as follows:

\[
R_{\text{data}_{\text{j},t_{\text{val}}}} = \frac{\sum_{i=1}^{n_d} SD_{i,j}}{t_{\text{val}}}
\]

\[
R_{\text{maint}_{\text{j},t_{\text{val}}}} = \frac{\sum_{i=1}^{n_m} SM_{i,j}}{t_{\text{val}}}
\]

where \( n_d \) and \( n_m \) are the number of data messages and maintaining messages during a fixed period time \( t_{\text{val}} \) in EM\textsubscript{j}; and \( SD_{i,j} \) and \( SM_{i,j} \) are the size of RPC messages that were transferred during the period \( t_{\text{val}} \). These two single EM metrics are used by the global metrics that are defined in Section 4.5.2.1.

### 4.5.1.3 Key-value Pair Operation Quality

As a storage system, the quality of file operations (file insertion and file retrieval) is important, from the perspective of users. According to the description of the Chord-based storage system in Section 1.3.2, file operations are accomplished by a series of key-value pair operations. As a result, the quality of a file operation closely depends
on the key-value pair operations. The metrics for operation quality work only on key-value pairs, as follows:

**Average Operation Latency**

The latency of a key-value pair operation (LI and LR) is the duration from the time point when the key-value pair operation starts (TIS and TRS) to the time point when the result of the key-value pair operation is fully received (TIR and TRR). This metric does not distinguish successful operations from failed operations. Although the blocks of files have the same fixed size as described in Section 1.3.1, there are still some different sized blocks in the system, such as some inode blocks, some indirect blocks and the tail blocks of files. In this case, the latency is measured according to the size of a key-value pair (Sblock).

\[
LI_i = \frac{TIR_i - TIS_i}{S_{\text{block}}} \tag{4.12}
\]

\[
LR_i = \frac{TRR_i - TRS_i}{S_{\text{block}}} \tag{4.13}
\]

The average latency of a key-value pair operation is measured periodically by collecting all key-value pair operations that start within the measuring period. The \(LI_i\) and \(LR_i\) in Equation 4.12 and Equation 4.13 are the \(i\)-th key-value pair operation during the measuring period. Therefore the average latency of a key-value pair operation is:

\[
\text{Ave}_LI = \frac{\sum_{i=1}^{n_{bi}} LI_i}{n_{bi}} \tag{4.14}
\]

\[
\text{Ave}_LR = \frac{\sum_{i=1}^{n_{br}} LR_i}{n_{br}} \tag{4.15}
\]

where \(n_{bi}\) and \(n_{br}\) are the number of key-value pair insertions and key-value pair retrievals during the measuring period.

**Successful Rates**

The successful rates (\(R_{\text{insert}}\) and \(R_{\text{retrieve}}\)) are the percentage of successful key-value pair insertions and key-value pair retrievals from all key-value pair insert operations and key-value pair retrieval operations during a fixed period of time. This is also an important quality attribute for a storage system.

\[
R_{\text{insert}} = \frac{NI_{\text{succ}}}{NI_{\text{total}}} \tag{4.16}
\]
CHAPTER 4. EXPERIMENTAL APPROACH

\[ R_{\text{retrieve}} = \frac{NR_{\text{succ}}}{NR_{\text{total}}} \]  

(4.17)

**Average Operation Hops**

For key-value pair operations, the request message routes through a number of EMs before it reaches to the appropriate EM with storage space for insertion operations or requested key-value pair for retrieval operations. The number of EMs by which the request message is forwarded is the *hops* of an operation. This is an important attribute for a Chord-based storage system. Because the underlying routing mechanism is the same for both block operations, this metric is also measured periodically without distinguishing the key-value pair operations.

\[ \text{Ave}_H = \frac{\sum_{i=1}^{n_b} H_i}{n_b} \]  

(4.18)

where \( H_i \) is the \( i \)-th block operation’s hops and \( n_b = n_{bi} + n_{br} \) is the total number of the two types of block operation.

### 4.5.2 Global Metrics

In order to evaluate the Chord-based storage system from the perspective of the load-balancing, it is not enough to measure the single EM’s attributes that are presented in Section 4.5.1. Therefore, this section introduces the global metrics that are produced from the single EM’s metrics by further processing. These global metrics provide an insight into the Chord-based storage system from a global view.

#### 4.5.2.1 Proportion of Remote Procedure Call messages

In each EM of the Chord-based storage system, there are two types of RPC messages: data message and maintaining message. From a system-wide view, it is also worthwhile to investigate the proportion of the different types of RPC messages across the whole system during a period of time. The proportion of each type of RPC message within a duration of \( t_{val} \) is evaluated as follows:

\[ P_{\text{data}_{t_{val}}} = \frac{\sum_{j=1}^{n} R_{\text{data}_{j,t_{val}}}}{\sum_{j=1}^{n} R_{\text{maint}_{j,t_{val}}} + \sum_{j=1}^{n} R_{\text{data}_{j,t_{val}}}} \times 100\% \]  

(4.19)

\[ P_{\text{maint}_{t_{val}}} = \frac{\sum_{j=1}^{n} R_{\text{maint}_{j,t_{val}}}}{\sum_{j=1}^{n} R_{\text{maint}_{j,t_{val}}} + \sum_{j=1}^{n} R_{\text{data}_{j,t_{val}}}} \times 100\% \]  

(4.20)
where \( R_{data}^{j,t} \) and \( R_{maint}^{j,t} \) are as defined in Equation 4.10 and Equation 4.11 and \( n \) is the number of EMs in the experiment.

### 4.5.2.2 System Storage Utilisation

In essence, the Chord-based storage system constructs a collective storage space that is contributed to by the Chord storage space of all EMs in the system. Therefore, from the perspective of the entire system, there is also a notion of storage utilisation of the Chord-based storage system at any given time point, besides the concept of one EM’s storage utilisation defined in Equation 3.1, on Page 64. The *system storage utilisation* \( U_{system,t} \) at time \( t \) is defined as follows:

\[
U_{system,t} = \frac{\sum_{j=1}^{n} L_{j,t}}{\sum_{j=1}^{n} S_{j,t}}
\]  

(4.21)

where \( L_{j,t} \) is the used storage space in EM \( j \) at time \( t \), \( n \) is the total number of EMs in the Chord-based storage system and \( S_{j,t} \) is the total Chord storage space of EM \( j \).

Nonetheless, the system storage utilisation never reaches 100\%, because the imbalanced storage space usage makes some of the EMs in the system reach their 100\% storage utilisation before others. As a result, the whole system’s storage capacity is not simply a summation of all the EMs’ Chord storage space. Recall that a time point, AWTP, is defined in Section 3.2.1 which represents the time when all machines reach \( X\% \) storage space utilisation. Because all machines have at least \( X\% \) storage space utilisation, an argument can be deduced using the system storage utilisation, as follows:

\[
U_{system,AWTP} \geq X\%
\]

(4.22)

where \( U_{system,AWTP} \) is the system storage utilisation at time point AWTP.

### 4.5.2.3 System Storage Capacity

Ideally, the storage capacity of the Chord-based storage system is the total storage space of all EMs’ local Chord storage space. However, in the original Chord-based storage system, some EMs may be more easily overwhelmed by the inserted key-value pairs in their Chord storage space, as discussed in Section 3.2.1. These EMs are determined to be the storage overloaded EMs in the system. The existence of these storage overloaded EMs makes a Chord-based storage system unreliable, because they have no storage space to store all the key-value pairs to be inserted in them. Therefore, a
reasonable definition of the system storage capacity of a Chord-based storage system should be the maximum accessible storage space when the system remains reliable. Because UTP labels the time of the first occurrence of a storage overloaded EM, the system storage capacity \( L_{capacity} \) of a Chord-based storage system is defined as the total used storage space of all EMs at UTP, as follows:

\[
L_{capacity} = \sum_{j=1}^{n} L_{j, UTP}
\]

(4.23)

where \( n \) is the number of EMs in a Chord-based storage system, \( L_{j, UTP} \) is the used storage space of EM \( j \) at UTP when there is an EM firstly reaching its 100% storage utilisation. Accordingly, the system storage utilisation at UTP, \( U_{system, UTP} \), is defined to be the reliable system storage utilisation of the Chord-based storage system.

### 4.5.2.4 Uniformity of Storage Utilisation

In a Chord-based storage system, maintaining a uniformly distributed storage utilisation across all machines is meaningful according to the discussion in Section 2.2. Therefore, a method to evaluate the uniformity of the storage utilisation of all EMs is required for the experiments of the Chord-based storage system. A D-statistic method [39] is employed to evaluate the difference between the distribution of EMs’ storage utilisation and a uniform distribution.

\[
D_{utilisation} = \sum_{j=1}^{n} \frac{|U_j - U_{mean}|}{U_{mean}}
\]

(4.24)

\[
U_{mean} = \frac{\sum_{j=1}^{n} U_j}{n}
\]

(4.25)

where \( D_{utilisation} \) is the D-statistic value, \( U_j \) is the storage utilisation of EM \( j \) and \( U_{mean} \) is the average storage utilisation of all \( n \) EMs.

### 4.6 Experimental Scenarios

Each experiment on the Chord-based storage system follows an experimental scenario. This section introduces all the experimental scenarios that have been used. Each scenario consists of a number of fundamental experimental processes, in other words, different permutations of experimental processes construct the different experimental
Chapter 4. Experimental Approach

4.6.1 Experimental Processes

Across the complete duration of an experiment, there are up to three experimental phases, namely, the stable phase, the file insertion phase and the file retrieval phase. Different tasks are designed to complete in these three different experimental phases. In some scenarios, there are only two phases. Although the number of experimental phases is not fixed for all experiments, the duration of each experiment is set the same, around 10 hours, due to the limitation of the experimental environment, in which experiments can only run overnight to be guaranteed to be without interference.

4.6.1.1 Stable Phase

The stable phase is the start phase of all experimental scenarios, which ensures that the Chord-based storage system is in a stable state before each experiment commences. The stable state of the Chord-based storage system means that the maintaining operations on all EMs have been completed to ensure that: i) the successor and predecessor are pointing to the correct nodes; ii) each node’s finger table has the correct entries; and iii) the Hilbert table in an EM is populated with correct entries. All these maintaining operations need to be completed in the stable phase, which is given one hour (3600 seconds) for all experimental scenarios.

In addition, this phase also ensures that the preparation works are complete for later experimental phases. The details of the preparation works in this phase are as follows:

Experimental Files Preparation

For an experiment on the Chord-based storage system, the input is the files, which need to be prepared before the experiment commences. Therefore, experimental files should be prepared before any file insertion phase. In the experiments for this study, each experimental file is constructed using random content. The random number generator from [40] is employed to produce the random content that consists of a series of random numbers in binary format. It is a fast random number generator with period \( \approx 3.138 \times 10^{57} \). The random seed of this random generator is introduced on Page 109. Each experiment has one specified file size. In each EM, file size is random with a normal distribution with the specified average size and a fixed variance. The research in this thesis has completed an investigation on two popular file sharing websites: Pirate Bay [52] and
Mininova [32]. The average file size of the files shared on these two websites is 89MB at the time of the investigation in 2008, so the average size is set to 89MB in the experiments of the Chord-based storage system. A random chosen variance, 10MB is used in this research to produce experimental files.

**Experimental Instruction**

In an experiment, the two file operations are file insertion and file retrieval, from the perspective of users. In order to simulate a practical storage system’s running, an experiment instruction is given to each file operation, which defines an operation frequency in each EM. Because Poisson distribution can represent the probability of a series of events occurring in a certain time interval (λ) or at a given average rate [21], it is an appropriate distribution to be used by the two file operations during the experiment. The parameter of Poisson distribution, λ, is given different values which represent the different average operation rates. The implementation of generating file operations’ random intervals is based on Knuth’s algorithm [26]. If all EMs are starting file operations at the same time, the network will experience a boost of network traffic at that time. Therefore, in each EM, the first file insertion and the first file retrieval wait for a random period of time. The random time is between 0 and the first produced operation interval in the EM. This prevents boosted network traffic due to similar starting times for file operations.

### 4.6.1.2 File Insertion Phase

In the file insertion phase, the file insertion operation is performed at specific average rates according to the file insertion instruction. According to the requirements of distinct experimental scenarios, the file insertion average rate can be fixed throughout the whole file insertion phase, or it can vary using different values during the procedure of the file insertion phase. Each average rate value lasts for at least an hour within the file insertion phase. For all experimental scenarios, file insertion is a compulsory experimental phase because all scenarios need at least one inserted file in the storage system, even though the file insertion phase is a short period of time in that case. The experimental files inserted in the Chord-based storage system in the file insertion phase are recorded in a database in the control machine in order to control file retrieval operations in any file retrieval phase.
4.6.1.3 File Retrieval Phase

The file retrieval phase has a similar setting method for average file retrieval rates compared with that for the file insertion phase. An experimental instruction for the file retrieval phase divides the whole phase into a series of consecutive sub phases, each of which lasts for at least one hour and is given a distinct average file retrieval rate. The file retrieval phase is not a compulsory part for all experimental scenarios. In insertion-related experimental scenarios, there is no file retrieval phase during experiments. The details of the experimental scenarios are introduced in the following section.

In an experiment, the file retrieval phase relies on a file insertion phase in which a number of experimental files are inserted in the Chord-based storage system for the following file retrieval phase. Because the experimental files that are stored in the Chord-based storage system are recorded in the database in the control machine, during a file retrieval phase, each EM can obtain a list of currently inserted experimental files from which an experimental file is randomly chosen as the retrieval target for the next file retrieval operation from the EM. Although maintaining a database in the control machine produces extra network traffic that is irrelevant to the network utilisation of the Chord-based storage system, it is necessary to provide this system-wide information to the file retrieval operation during an experiment.

4.6.2 Normal Scenarios

The normal scenarios are designed to simulate the normal running process of the Chord-based storage system without deliberate network overloads. Therefore, in normal scenarios, the file operation frequencies are set to a moderate value throughout the whole experiment process. There are three types of normal scenario: Insertion Normal Scenario (INS), Retrieval Normal Scenario (RNS) and Normal Scenario (NS).

The first two experimental scenarios concentrate on only one file operation in the Chord-based storage system, and the last one includes both file operations. All these three experimental scenarios start from the stable phase. For INS, there is only file insertion throughout the whole experimental duration after the stable phase. Although the normal scenarios are used to simulate normal running conditions without network overloads by using moderate file operations frequencies, the INS is also designed to stimulate storage overloads in the Chord-based storage system. For RNS, the three experimental phases are all involved. The stable phase is followed by a short period of a file insertion phase in which a certain number of experimental files are inserted into
the Chord-based storage system before the following file retrieval phase commences. The file retrieval phase starts after the file insertion phase stops and performs the file retrieval operation from the storage system by randomly choosing experimental files as retrieval targets from the inserted file database in the control machine. For NS, the file insertion phase is not stopped although the file retrieval phase starts some time later. In this case, after the file retrieval phase starts, the experiment performs both file insertion and file retrieval at the same time.

The normal scenarios can be used to examine the Chord-based storage system under normal running conditions, and to evaluate the storage system with the P_VNode storage load-balancing approach by stimulating storage overload.

4.6.3 Overload Scenarios

In order to evaluate the network performance of the Chord-based storage system, a deliberate network overload is required during the experiment. The overload scenarios are designed to accomplish this task. There are two types of overload scenarios: Insertion Overload Scenario (IOS) and Retrieval Overload Scenario (ROS). In IOS, there are only the stable phase and the file insertion phase that starts after the stable phase until the end of the experiment. In order to overload the network of the Chord-based storage system, the insertion frequency of experimental files is set at a high value in some experiments using IOS, compared with the operation frequencies that are set in the normal scenarios.

As for ROS, there are all three experimental phases. After the stable phase, a short period of the file insertion phase performs the file insertion operation to store a certain number of experimental files in the Chord-based system for the following file retrieval phase. The file retrieval phase starts after the file insertion phase stops. During the file retrieval phase, file retrieval operations for the stored experimental files are performed at a high frequency.

The overload scenarios are able to find the operational capacity and the hotspots of the Chord-based storage system as described in Section 5.4.

4.7 Summary

In this chapter, an experiment methodology has been chosen to investigate the Chord-based storage system and to validate the P_VNode load-balancing approach compared
CHAPTER 4. EXPERIMENTAL APPROACH

to simulation from the perspectives of time consuming, available resources and re-
search targets. The experiment objectives have been defined to contain three parts:

1. to validate the proposed storage load-balancing approach, P_VNode;

2. to investigate the network utilisation of the Chord-based storage system; and

3. to evaluate the original Chord-based storage system.

All experiments are performed in five student laboratories, in which all machines are
connected through LAN. The experimental machines in these laboratories are given the
same storage quota of 135GB and their system times are synchronised using NTP. In
order to perform experiments on these experimental machines, three packages, namely,
GMP, SFSLite and Berkeyley DB are installed. Outside of the designated student labo-
ratories, a control machine is used in the experiment control system and the experiment
monitoring system. Over each night, the experiment runs for around 10 hours during
which a number of metrics are monitored in each EM. All experimental metrics are
described in Section 4.5. Three experimental phases, the stable phase, the file insertion
phase and the file retrieval phase, and five experimental scenarios, INS, RNS, NS, IOS
and ROS are defined for the experiments. The experimental results are presented and
discussed in Chapter 5.
Chapter 5

Experimental Results

5.1 Introduction

After extensive experiments were finished, there were a considerable amount of experimental results in the form of raw log data produced by the experiment monitoring system. This chapter interprets this raw data into meaningful conclusions.

This chapter is divided into three parts. In the first part, Section 5.2 presents the experimental results that examine the original Chord-based storage system without P_VNode, with two purposes: 1) examining the usage of two important computing resources, CPU and memory in each EM during experiments, and 2) supporting the decision of adding the Hilbert table in the load-balancing approach described in Chapter 3. The second part is Section 5.3, which presents the experimental results of the Chord-based storage system with the P_VNode load-balancing approach, P_VNode. In this section, three aspects of the storage load-balancing in the Chord-based storage system are investigated, namely, the storage uniformity, the storage utilisation time points and the system storage capacity. Finally, Section 5.4 gives the experimental results for network utilisation in the original Chord-based storage system and the description of a hotspot detection approach. Section 5.5 summarises these results.

5.2 Original Chord-based Storage System

According to Section 4.3, as a part of the experimental objectives, investigating the original Chord-based storage system is important to the research in this thesis, because this information is hard to obtain in the simulation method that has been widely employed by most DHT related researches. Therefore, besides the storage usage and
network utilisation, other aspects of EMs are considered in the experiments, such as CPU running time, the memory usage and different types of RPC messages transferred through the network. Furthermore, the CPU, memory and RPC messages information in each EM during experiments are meaningful in the process of designing and developing the load-balancing approach for the Chord-based storage system. Each EM extracts the appropriate information during an experiment using the installed monitoring programs that have been described in Section 4.4.4. The collected experimental results from all EMs are presented in Section 5.2.1 and Section 5.2.2.

5.2.1 CPU and Memory Usage

The essence of the Chord-based storage system is to harness the collective storage space that is contributed by all participating machines. Although the Chord-based storage system does not exploit the CPU running time and memory usage from EMs as much as it does their Chord storage space, the storage system’s running requires amounts of CPU running time and memory usage in each EM. During the running of the Chord-based storage system, two factors are important, namely the number of virtual nodes in each machine and the busyness of the system. This section reveals the relationship between CPU and memory usage and the number of virtual nodes, and the relationship between CPU and memory usage and the busyness of the storage system.

The experimental results are measured according to the single machine metrics $U_{CPU}$ and $U_m$ that are described in Section 4.5.1.1. After all EMs’ $U_{CPU}$ and $U_m$ are collected, Table 5.1 concludes the average CPU usage rate and average memory usage in experiments with different numbers of virtual nodes in each EM. These experiments run as the normal scenario (NS) as described in Section 4.6.2, which is given the experiment instructions that define the same file insertion frequency and the same file retrieval frequency, which are a file operation every 360 seconds. The P_VNode load-balancing approach described in Section 3.4 is not deployed in these experiments.

<table>
<thead>
<tr>
<th>Virtual Nodes per EM</th>
<th>Average CPU Usage Rate</th>
<th>Average Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.52 %</td>
<td>64.80 MB</td>
</tr>
<tr>
<td>2</td>
<td>3.54 %</td>
<td>103.29 MB</td>
</tr>
<tr>
<td>5</td>
<td>4.97 %</td>
<td>243.78 MB</td>
</tr>
</tbody>
</table>

Table 5.1: Average CPU usage rate and average memory usage in experiments with different numbers of virtual nodes per EM.
According to Table 5.1, on the one hand, the introduction of the extra virtual nodes in EMs does not cause a significant increase in CPU running time demands. The values of the average CPU usage rate, 3.52%, 3.54% and 4.97%, are small, which indicates that the CPU requirements of a Chord-based storage system are not high when the system is running normally. This conclusion could be deduced from the following CPU requirements of the main functional operations of the Chord-based storage system:

1. Producing Chord IDs using cryptographic hash functions such as SHA-1 and using a Hilbert curve requires CPU running time. However, the Chord IDs such as Node ID (Section 1.2.1) and Hilbert ID (Section 3.3.2) are usually produced once for each node and each machine, respectively, so producing Node IDs and Hilbert IDs cannot entail too many CPU demands. In contrast to this, the Data ID is formed from the values of the SHA-1 hash function applied to the content of a block of data (1MB) according to the description in Section 1.2.1. Therefore, producing Data IDs requires more CPU time than producing other IDs. But this time is independent of the number of virtual nodes in EMs and so would not be expected to vary.

2. Message routing across machines in the system is not a CPU-intensive operation. The increasing number of virtual nodes produces more messages among virtual nodes, but the CPU usage rate does not increase significantly along with the increasing of the number of virtual nodes in Table 5.1.

3. In each EM, a local database is deployed to store key-value pairs from other EMs (Table 4.2 on Page 106). The insertion, lookup and retrieval operations of the local database also require CPU running time, just like other database systems. According to the employed implementation of the Chord-based storage system, DHash, all virtual nodes in each machine share the same database. Therefore, the increasing of the number of virtual nodes does not increase the number of local databases per EM. And an increasing number of virtual nodes does not produce any more database operations on local machines.

Since most functional operations of the Chord-based storage system do not become much more intensive when the number of virtual node increases, CPU requirements do not increase significantly, either.

On the other hand, the situation of memory usage is different from that of CPU usage rate. The memory utilisation increases linearly according to the increasing number
of virtual nodes per EM, because the virtual nodes in an EM work separately from one another. Each virtual node requires a separate finger table, successor and predecessor pointer and Hilbert table (when P_VNode approach is deployed) to make the storage system function normally. The codes to maintain these data structures are loaded into the memory for each virtual node in an EM. Therefore, the memory usage increases with an increasing number of virtual nodes as can be readily seen in Table 5.1.

The system busyness that is generated by file insertion and retrieval frequency is another factor affecting CPU and memory usage of the Chord-based storage system in experiments. Figure 5.1 and Figure 5.2 plot the CPU usage rate and memory usage in the experiments with different file insertion frequencies, which are represented by the time interval between successive file insert operations. These experiments run as the Insertion Normal Scenario (INS) as described in Section 4.6.2. For each experiment, Figure 5.1 presents the maximum, the minimum and the average CPU usage rates over every 5 second period for all EMs during the experiment’s running. Each experiment is given different file insertion intervals, the shorter of which indicates that the experiment files are inserted into the storage system more frequently, which means that the system is more busy than the experiments with longer file insertion interval.

According to Figure 5.1, the maximum CPU usage rate increases significantly as the file insertion rate increases, although the increasing of the average CPU usage rate is not significant. The maximum CPU usage rates in Figure 5.1 indicate that some EMs’ CPUs are more intensively used by the Chord-based storage system during certain 5 seconds periods in experiments with intensive file insertion operations than experiments with long file insertion operation time interval. The maximum CPU usage rate in the experiment with file insertion every 30 second is 72.2% within 5 seconds for the storage system. The experiment with file insertion every 360 seconds has the smallest value shown, namely 26.8% maximum CPU usage. Furthermore, the average CPU usage rate also increases as file insertion operations become intensive in busy experiments, such as the experiment with 30 seconds file insertion time interval, which has 8 times more average CPU usage rate than the experiment with 360 seconds file insertion time interval. According to the analysis of the functional operations of the Chord-based storage system on Page 125, producing Data ID and local database operations are both closely related to the frequency of file insertion operations. Therefore, the CPU usage rate is affected by the busyness of the storage system.

In Figure 5.2, the maximum memory usage for most experiments is similar, except for the experiment with 30 seconds file insertion interval, because, in that experiment,
there was serious network saturation which caused a large number of pending network messages to be cached in a number of EMs’ memory. This significant increasing of the memory usage is caused by a network overload in the Chord-based storage system from the memory consumption perspective. More discussion of the detection of network overload is given in Section 5.4.2. Nevertheless, the average memory usage does not show the same significant memory usage increase in Figure 5.2. On average, the experiment with 30 seconds file insertion interval uses 2 times more memory than the experiment with the biggest file insertion interval, 360 seconds. The experiment results in Figure 5.2 indicate that the busyness of the system does not affect EM’s memory usage significantly, as long as there is no overloading in an experiment.

Consequently, without network overload in experiments, EM’s CPU usage rate is more sensitive to the busyness of the system, and its memory usage is mostly determined by the number of virtual nodes in each EM. Therefore, if the Chord-based storage system is deployed in a memory limited environment, the fewer the number of virtual nodes in each machine, the better. Actually, the CPU usage rate and memory

Figure 5.1: Maximum, minimum and average CPU usage rate for different intervals between file insertions.
CHAPTER 5. EXPERIMENTAL RESULTS

Figure 5.2: Maximum, minimum and average memory usage for different intervals between file insertions.

usage are implementation dependent aspects. A different implementation in a different operating system would produce different results. A survey of all implementations of Chord-based storage systems is beyond the scope of this research. This thesis still focuses on the P_VNode storage load-balancing scheme, results for which will be presented in Section 5.3 and Section 5.4.

5.2.2 RPC Messages in the Chord-based Storage System

The implementation of the Chord-based storage system in experiments employs RPC to accomplish the communication between EMs, according to the description in Section 4.4.2. The RPC messages produced in an experiment can be categorised into two types: data messages and maintaining messages. Data messages refer to the RPC messages that are exchanged between EMs to complete the file insertion and file retrieval operations. Maintaining messages denotes all other RPC messages between EMs to ensure each node has a correct finger table, successor and predecessor. In order to
find the relationship between these two types of RPC messages, the experiments are designed under the INS and RNS scenarios (described in Section 4.6.2) with different file operation frequencies using the metrics, $P_{data_{t_{val}}}$ and $P_{maint_{t_{val}}}$ ($t_{val}$ is set to the whole time duration of experiments, $t_{exp}$) that are defined in Section 4.5.2.1. Figures 5.3 and 5.4 show the proportions of the overall message size for the two types of RPC message during these experiments. Because the proportion of data messages are all higher than 99%, in order to clarify the proportions of maintaining messages these two figures only present the information from 99% to 100% of the whole network traffic.

![Figure 5.3: Proportion of different RPC messages in the entire network sending traffic in the INS scenario.](image)

The proportion of the two types of RPC message in Figures 5.3 and Figure 5.4 demonstrate that data messages dominate the entire network traffic, when there are file operations in the Chord-based storage system. Increasing the frequency of the file operations makes the data messages proportion increase among the whole network traffic. Therefore, compared with data messages, the network traffic produced by maintaining messages is small. Even if two times the maintaining messages were produced by
all EMs, they would still hardly become a burden on the network. This experimental result provides freedom to add a load-balancing data structure that requires maintaining in each EM, namely the Hilbert table (Section 3.3.3), because extra maintaining messages for Hilbert table are unlikely to saturate the network in the way that data messages could do.

5.2.3 Recap

This section has examined the experimental results for the original Chord-based storage system without the P_VNode load-balancing approach. CPU usage rate is significantly affected by the frequency of the file operations, and the increasing number of virtual nodes in each EM makes memory usage increase. Moreover, most network traffic is produced by data RPC messages during experiments; maintaining messages, which are less than 0.5% of all RPC messages, contribute a very small part of the network

Figure 5.4: Proportion of different RPC messages in the entire network sending traffic in the RNS scenario.
traffic. Although this section does not include the evaluation of the P_VNode load-balancing approach, it still plays an important role for the research in this thesis, by indicating promising directions during the procedure of designing the load-balancing approach. The rest of this chapter concentrates on the improvements that are achieved by P_VNode storage load-balancing within the undertaken experiments.

5.3 Improvement of Storage Load-balancing

This section examines the experimental results that demonstrate the improved storage loads across all EMs in the storage system from three perspectives: storage uniformity, storage utilisation time points and system storage capacity. In order to investigate the reaction of the P_VNode load-balancing scheme, which is deployed in the Chord-based storage system, the experiments are performed using the INS scenario (Section 4.6.2) which includes deliberate storage overloads using high frequency file insertion operations. But the frequency of file insertion operations is not high enough to produce any of the network overloads that are discussed in Section 5.4. The experimental results are described in the rest of this section, which starts with the uniformity of storage usage in EMs.

5.3.1 Storage Utilisation Uniformity

According to the description in Section 2.2.1.1, Chord already has its natural storage load-balancing property, which relies on the uniformity of the produced Chord IDs. However, the uniformity of Chord IDs does not directly relate to the uniformity of the storage space usage by EMs to store key-value pairs. Accordingly, the uniformity of the storage space usage of all EMs in the storage system is studied in the experiments described here. Because the balancing target is the storage utilisation as defined in Section 3.2.1, the metric used to evaluate the uniformity of EM’s storage utilisation is the D-Statistic of storage utilisation ($D_{utilisation}$), that is defined in Section 4.5.2.4. Figure 5.5 and Figure 5.6 compare the $D_{utilisation}$ in the experiment with P_VNode storage load-balancing to the experiment without P_VNode. Figure 5.5 is a full picture, while Figure 5.6 zooms in on the smaller range and emphasises the differences of the comparison that are not obvious in the full picture.

In Figure 5.5, the $D_{utilisation}$ has a significant high value, 283.36, an hour after the experiment starts and file insertion operations are starting in the INS experimental
Figure 5.5: $D_{utilisation}$ represents the uniformity of the storage utilisations across all EMs. This figure compares the Chord-based storage system with P_VNode and the one without P_VNode.

Scenario. At the initial stage of the file insertion phase (Section 4.6.1.2), most EMs have empty Chord storage space. When there are only a few EMs that have key-value pairs in their Chord storage space, the distribution of the storage utilisation of all EMs is obviously skewed against a uniform distribution. However, the $D_{utilisation}$ then decreases significantly and is stabilised between 67 and 68 during the experiment afterwards. The decreased and stable $D_{utilisation}$ in the experiment afterwards shows that the natural load-balancing of Chord provided by its key-value pair insertion pattern has achieved a certain load balancing capability which decreases the $D_{utilisation}$ from 283.36 to the value between 67 and 68. A certain uniform distribution of EM’s storage utilisation is maintained by the natural load-balancing of Chord, as has been discussed in Section 2.2.

Furthermore, the introduction of the P_VNode load-balancing approach is able to provide even more uniform distribution of the EM’s storage utilisation. Section 3.2.1 introduces the First Warning Time Point (FWTP) that is the time when any machine’s
storage utilisation reaches a specific value. Recall that the FWTP is the first warning of the potential storage overload in the Chord-based storage system. The FWTP in this research is the time point when the first EM reaches 80% storage utilisation according to Section 3.4.3. In Figure 5.5, when FWTP emerges after around 7 hours and 44 minutes from the experiment starting, the P_VNode scheme starts to work. The solid line is the $D_{utilisation}$ of the storage system with the P_VNode load-balancing approach, while the dashed line is the same storage system without the P_VNode load-balancing approach. The difference between the solid line and the dashed line starts right after the FWTP that is labelled in Figure 5.6 which provides a clearer vision. The solid line has a significant decrease after the FWTP, which means that the P_VNode scheme makes the storage utilisation of all EMs more uniform by migrating key-value pairs from machines with higher storage utilisation to machines with lower storage utilisation.

According to the $D_{utilisation}$ metric in Figure 5.5 and Figure 5.6, Chord’s natural load-balancing property provides a certain degree of uniformity of storage utilisation,
which is then improved by the P_VNode load-balancing approach. This improvement of the uniformity of storage utilisation provides the Chord-based storage system with more improvements from other perspectives. These are presented in the remainder of this section.

### 5.3.2 Storage Utilisation Time Points

The $D_{\text{utilisation}}$ values show the improvement of the storage utilisation uniformity, which, in turn, changes the storage utilisation time points. Section 3.2.1 introduces three storage utilisation time points: Unreliable Time Point (UTP), First Warning Time Point (FWTP) and All Warning Time Point (AWTP). Recall that AWTP represents the time when all EMs reach a specific storage utilisation, and UTP is the time when any EM reaches 100% storage utilisation. The 80% storage utilisation is the specific storage utilisation used for FWTP and AWTP in this research according to the discussion in Section 3.4. In Figure 5.7, the two time points, FWTP and UTP, are shown for experiments with different numbers of virtual nodes per EM and experiments with different storage load-balancing settings. Figure 5.7 shows FWTP and UTP using two concatenated bars that denote the duration from the experiment start to FWTP and the duration from FWTP to UTP, respectively. Table 5.2 lists the values of FWTP and UTP related time durations that are shown in Figure 5.7.

<table>
<thead>
<tr>
<th>Number of VNs</th>
<th>Start to FWTP (seconds)</th>
<th>FWTP to UTP (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>no P_VNode</td>
</tr>
<tr>
<td>1</td>
<td>13720</td>
<td>2510</td>
</tr>
<tr>
<td>2</td>
<td>21780</td>
<td>2766</td>
</tr>
<tr>
<td>5</td>
<td>27895</td>
<td>4385</td>
</tr>
</tbody>
</table>

Table 5.2: Experimental duration (in seconds) from start to FWTP and from FWTP to UTP in experiments with 1, 2 and 5 Virtual Nodes (VNs) per EM.

The P_VNode load-balancing approach postpones the advent of both FWTP and UTP according to Figure 5.7’s comparison of the time duration from start to FWTP in experiments with 1, 2 and 5 virtual nodes per EM. An increasing number of virtual nodes per EM makes the distribution of storage utilisation more uniform, therefore, the time duration from experiment start to FWTP and UTP is extended. The deferment of FWTP makes P_VNode start later, while the deferment of UTP makes the storage system have more reliable running time.
CHAPTER 5. EXPERIMENTAL RESULTS

Figure 5.7: Durations from experiment start to the time points FWTP and UTP in experiments with 1, 2 and 5 virtual nodes (VNs) per EM. The right bars aside of experiments with 2 and 5 virtual nodes per EM are those for the experiment with the P_VNode load-balancing approach.

After P_VNode starts, it extends the time duration between FWTP and UTP further from 2766 seconds to 4087 seconds for the system with 2 virtual nodes per EM and from 4385 seconds to 8485 seconds for the system with 5 virtual nodes, according to Table 5.2. The same situation is also shown by the adjacent bars for 2 and 5 virtual nodes in Figure 5.7. According to these results, P_VNode provides more reliable running time for a Chord-based storage system than one without the P_VNode storage load-balancing approach.

Because P_VNode’s migration is based on the whole storage space of a virtual node, P_VNode only works on a storage system with multiple virtual nodes per EM. Each migration operation transfers a whole virtual node to another machine in the system, according to the algorithm presented in Section 3.4. Furthermore, there is no difference between the time durations from experiment start to FWTP for the storage system with P_VNode and the one without P_VNode. This is demonstrated by the slash
filled lower bars for 2 and 5 virtual nodes in Figure 5.7.

In Figure 5.7, both FWTP and UTP are shown, but AWTP cannot be found. AWTP is defined to be the theoretical stop point of P.VNode in Section 3.4. However, unlike FWTP and UTP, AWTP does not necessarily exist in an experiment using the Chord-based storage system, i.e. there is always at least one machine at less than 80% storage utilisation when the storage system becomes unreliable (i.e. one machine researches 100%). Indeed, according to the investigation on the utilisation of the system storage in the following section, AWTP has not been discovered in any experiments of this research yet.

5.3.3 System Storage Capacity

The introduction of both virtual nodes and P.VNode defers FWTP and UTP which are related to the performance of the system-wide storage space usage, system storage capacity ($L_{capacity}$) that is defined in Section 4.5.2.3. $L_{capacity}$ is evaluated by summing all used Chord storage space of EMs at UTP, which indicates the largest accessible system-wide storage space while the Chord-based storage system remains reliable. The system storage utilisation ($U_{system,t}$) is defined as a metric of the whole system’s storage space contributed by all EMs, according to the description in Section 4.5.2.2. Recall that the system storage utilisation at UTP ($U_{system,UTP}$) is defined as the reliable system storage utilisation in Section 4.5.2.2. Table 5.3 lists the $U_{system,UTP}$ and $L_{capacity}$ in experiments with different numbers of virtual nodes per EM and different storage load-balancing settings.

<table>
<thead>
<tr>
<th>Number of VNs per EM</th>
<th>$U_{system,UTP}$</th>
<th>$L_{capacity}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no P.VNode</td>
<td>with P.VNode</td>
</tr>
<tr>
<td>1</td>
<td>11.94 %</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>22.43 %</td>
<td>37.18 %</td>
</tr>
<tr>
<td>5</td>
<td>43.14 %</td>
<td>61.82 %</td>
</tr>
</tbody>
</table>

Table 5.3: Reliable system storage utilisation ($U_{system,UTP}$) and system storage capacity ($L_{capacity}$) for experiments with 1, 2, 5 virtual nodes (VNs) per EM. Experiments with 2 and 5 virtual nodes are provided a comparison between the storage system with P.VNode and that without P.VNode.

If there was an EM $M_o$ reaching its 100% storage utilisation that is defined as UTP in Section 3.2.1, the whole Chord-based storage system would be unreliable after the UTP, because further key-value pair insertions to $M_o$ would be discarded. Therefore,
after UTP, the Chord-based storage system is confronted with an unreliable situation. According to the results in Table 5.3, the maximum system storage utilisation is quite low (11.94%) when there are no virtual nodes or P_VNode in the storage system. In this case, only 2.38TB system storage capacity is available to the system’s users. An increase in the number of virtual nodes per EM increases the maximum system storage utilisation and system storage capacity. The introduction of the P_VNode storage load-balancing approach increases the maximum system storage utilisation and system storage capacity further. When there are 5 virtual nodes per EM and the P_VNode approach is used, the Chord-based storage system is able to achieve 61.82% maximum system storage utilisation and 12.28TB system storage capacity. Therefore, P_VNode load-balancing approach significantly improves the reliable system-wide storage capability in the Chord-based storage system.

In Table 5.3, the largest value of the maximum system storage utilisation is 61.82% in the storage system with 5 virtual nodes per EM and P_VNode. This largest value of the system storage utilisation demonstrates the absence of AWTP in all experiments. If there is AWTP in an experiment, then all EMs’ storage utilisation would be over 80%, which means that the system storage utilisation would be over 80%, according to Equation 4.22, on Page 116. However, the largest value of maximum system utilisation is only 61.82%, so AWTP cannot exist in any experiments.

In conclusion, the introduction of multiple virtual nodes and P_VNode provides more uniform distribution of storage utilisation across EMs. Then, more uniformity of the storage utilisation distribution postpones both FWTP and UTP. The deferment of UTP, in turn, increases the system storage capacity. Therefore, the storage load-balancing property of the Chord-based storage system is improved by the P_VNode load-balancing approach.

5.4 Network Overload in the Original Chord-based Storage System

As a part of the evaluation of the original Chord-based storage system, Section 5.2.2 has discussed the proportions of the two types of RPC message that are transferred through the network. This section investigates in more detail the network overload issues for each EM in the original Chord-based storage system. Although the network load-balancing approach has not been completed in this thesis, the investigation of network overload is still important to the P_VNode storage load-balancing approach,
because the network load-balancing is closely related to the storage load-balancing according to the discussion in Section 3.2. Section 5.3.3 illustrates the existence of the UTP, after which the Chord-based storage system is unreliable due to the probability of further key-value pairs to be stored in the storage overloaded EM with 100% storage utilisation. Therefore, further file insertions after UTP may be failed by the storage system. However, the advent of UTP is not the only reason for the Chord-based storage system to make file operations fail. Possible network overload should be taken into consideration, such as the operational capacity and hotspots in the system.

5.4.1 Operational Capacity

In a Chord-based storage system, users may perform file insertion and file retrieval operations from all participating machines. In the experiments, frequencies of the file operations in all EMs contribute to the frequency of the file operations for the whole system. According to the analysis in Section 3.2.2, on page 67, hotspots emerge if the file operation’s frequency is high enough. Therefore, file operations would be failed if the operations work on the hotspots. The operational capacity is defined as the maximum frequency of file operations that ensures that there is no hotspot which leads to the possibility of file operation failure. The operational capacity is obtained next by a series of experiments with different file operation frequencies.

According to the description in Section 1.3.1, the files are divided into a number of blocks to be inserted into the Chord-based storage system. Therefore, the operational capacity is narrowed down from these experiments by observing the metrics of key-value pair operations, average operation latency ($\text{Ave}_{LI}$ and $\text{Ave}_{LR}$), successful rates ($R_{\text{insert}}$ and $R_{\text{retrieve}}$) and average operation hops ($\text{Ave}_H$) that are defined in Section 4.5.1.3.

The experiments are repeated with different file operation frequency settings until the operational capacity is narrowed down to a rather small range. The final experiment discovers that the file insertion capacity is somewhere between 40 seconds insertion interval and 30 seconds insertion interval. The experimental results are shown in the following figures and tables. In Figure 5.8, the average block insertion successful rate is shown; Figure 5.9 shows the average latency of block insertions; Figure 5.10 describes the average number of hops for block insertions. In all these figures, the x axis represents the whole duration of the experiment, which lasts for 11 hours. The average values of the variables in the above figures in different experiment phases are listed in
CHAPTER 5. EXPERIMENTAL RESULTS

Table 5.4. As before, one hour at the start is used to establish a stable condition. Table 5.4 also includes insertion intervals of the different experiment phases. According to Figure 5.8, from the 7th hour, the insertion time interval is set at 30 seconds in each EM; and block insertions are not all successful. The corresponding data in Table 5.4 shows that block insertion failures actually start from the end of ‘Insertion 6’, Therefore, the file insertion capacity is somewhere between 40 seconds insertion interval and 30 seconds insertion interval.

<table>
<thead>
<tr>
<th>Experimental Phases</th>
<th>Insertion Interval</th>
<th>Successful Rate ($R_{insert}$)</th>
<th>Average Latency ($Ave.LI$)</th>
<th>Average Hops ($Ave.H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stable</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Insertion 1</td>
<td>90s</td>
<td>100%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 2</td>
<td>80s</td>
<td>100%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 3</td>
<td>70s</td>
<td>100%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 4</td>
<td>60s</td>
<td>100%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 5</td>
<td>50s</td>
<td>100%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 6</td>
<td>40s</td>
<td>99.9%</td>
<td>0.011ms</td>
<td>4</td>
</tr>
<tr>
<td>Insertion 7</td>
<td>30s</td>
<td>98.5%</td>
<td>0.021ms</td>
<td>4</td>
</tr>
<tr>
<td>No OP</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.4: Experiment insert instructions and insertion successful rate ($R_{insert}$), average block insertion latency ($Ave.LI$) and average hops ($Ave.H$) in different experiment phases. After ‘Insertion 7’, there is a ‘no operations’ period until the end of the experiment.

Table 5.5 shows the corresponding experimental results for the block retrieval success rate, average latency and average hops in order to find the file retrieval capacity of the system. The ROS scenario (Section 4.6.3) is used in the process of finding file retrieval capacity. The file retrieval capacity is also somewhere between 40 second retrieval interval and 30 second retrieval interval, which is the same as the results for the file insertion operation in the experiment above, using scenario IOS. Table 5.5 lists similar experimental results to the data in Table 5.4, because the experiments running in the IOS and ROS scenarios have similar communication patterns. In IOS, a single machine (client mode) inserting a file into the system needs to communicate with several other machines (server mode) for all the blocks of the file. Then these blocks are transferred concurrently to other machines which act as server machines to provide storing service for the client. In ROS, the roles of server and client are simply exchanged. A number of machines make requests for several different blocks from a single machine. Then the single machine is a server to fulfil these requests, and the
Figure 5.8: Average block insertion successful rate ($R_{insert}$).

Table 5.5: Experiment retrieval instructions, block retrieval successful rate ($R_{retrieve}$), average block retrieval latency ($Ave_{LR}$) and average hops ($Ave_{H}$) in different experiment phases. After ‘Retrieval 7’, there are no file operations until the end of the experiment.
other machines are client machines making the requests. As the file operation frequency increases, the burden on the single machine in both IOS and ROS scenarios increases. Therefore, file insertion operation and file retrieval operation have similar operational capacity. However, this similarity is based on the experimental scenario of ROS in which the retrieval files are chosen randomly using a uniform random number generator (Section 4.6.3). In the real world, there are usually skewed retrieval request patterns in a storage system.

### 5.4.2 Hotspot Detection Approach

Section 5.4.1 has investigated the file operation capacity which is the maximum file operation frequency before hotspots emerge in the Chord-based storage system. Although observing block operation success rate is a good method to find operational capacity, it is not a reliable method to detect hotspots in the system, because, for forecasting purposes, it is always too late to report operation failures by checking the occurrence of hotspots in the storage system. Therefore, a hotspot detection approach is required.

Figure 5.9: Average latency of block insertions (Ave_LI).
Because, according to the description in Section 3.2.2 on Page 66, the Chord-based storage system transfers key-value pairs using TCP which always attempts to send data at a maximum transfer rate (see Page 66), the monitored network transfer rates, $R_{\text{send}}$ and $R_{\text{recv}}$ (Section 4.5.1.2) reaching their maximum value, around 12MBytes/s in the LAN on a full duplex 100Mbit/s ethernet, are not a reliable indicator of the emergence of a potential hotspot. Therefore, the appropriate monitor target for a hotspot detection approach should be the queues for network transmission, because the existence of data in network queues implies that either the machine or the network cannot process this queued data currently.

In a TCP connection, there are two queues at each endpoint, namely a receive queue and a send queue in the Fedora Linux that is the operating system of the experimental machines as described in Section 4.4.2. The size of both the receive queue and send queue for all TCP connections in an EM is monitored by the experiment monitoring system, as described in Section 4.5.1.2 The length of the receive queue for the Chord-based storage system.

Figure 5.10: Average hops of block insertions ($Ave_H$).
TCP ($Q_{TCP_{recv}}$) indicates the capability of the program in the machine to process the data received from the network, whereas the size of the send queue ($Q_{TCP_{send}}$) represents the amount of data unconsumed by the network and still waiting in the server machine’s operating system. In order to detect a hotspot, the monitoring target is the send queue for all connected TCP channels ($Q_{TCP_{send}}$). This research proposes a hotspot detection approach with a timer that is a parameter to control the sensitivity of the hotspot detection. Figure 5.11 is the state transition diagram of the timer that is used to determine the hotspot status of a machine. In the hotspot detection approach,

![State diagram of hotspot detection approach](image-url)

Figure 5.11: State diagram of hotspot detection approach.

the changing rate of the TCP send queue’s size is considered instead of just simply the size of the queue, because the TCP send queue is normally implemented using a FIFO (First In First Out) queue. If the size of the send queue is stable at a certain value, the data in the send queue may change over time, which indicates that the machine keeps sending out data normally. Therefore, the hotspot detection approach times only the duration of TCP send queue increases. If the size of monitoring queues decreases, the timer is suspended. But the timer is resumed if the size of the monitoring queue changes to be increasing. If the TCP send queue becomes empty, the timer is stopped and reset to zero. If the timer runs out, the machine is determined to be a hotspot.
After a hotspot reducing method, such as the replication approach suggested in Section 3.5, stops the increase of the TCP send queue, the timer starts again to detect the next hotspot.

The timeout value for the timer of the hotspot detection approach affects the sensitivity of the detection approach. There are two network reasons that could contribute to data being in the TCP send queue, namely network boost traffic and network saturation.

The network boost traffic is a temporary situation that is caused by a number of file operations happening to come through the same machine at the same time. Usually, network boost traffic happens only once without any repeats afterwards. At this point, it is not appropriate to assert a machine to be a hotspot according to its non-empty TCP send queue contributed by the network boost traffic.

Network saturation, however, is a long term situation caused by too frequent file operations in a busy storage system. This is the situation that should be reported as a hotspot in the system.

Therefore, the timeout value of hotspot detection timer should be long enough to exclude situations caused by network boost traffic, and it is also required to be short enough to report the network saturation situation at an early stage, because a saturated sending network fails file operations if the file’s key-value pair operations have longer waiting time than the key-value pair operation’s timeout value, according to the description in Section 1.3.2.

In order to determine a reasonable timeout value given the tradeoffs involved, the experiment is repeated using different timeout values. The IOS scenario is employed in the experiment to find an appropriate timeout value for the hotspot detection approach. The experiment settings are similar to the experiments used in Section 5.4.1 targeting the file insertion capacity. Table 5.6 lists the number of detected hotspots in experiments using different timeout values. The experiment phases in Table 5.6 are the same as those in Table 5.4.

Referring to Table 5.4 and Figure 5.8, the actual block insertion failures happen in the experimental phases from Insertion 6 to Insertion 7. When the timeout value is set to be 30 seconds and 60 seconds, a large number of hotspots are reported in the experiment phases from Insertion 1 to Insertion 5. According to Figure 5.11, when timeout value is set to be 30 seconds, an EM $M_h$ would determine itself to be a hotspot if its TCP queue had data for more than 30 seconds. Even if $M_h$’s TCP queue is decreasing by transferring data after a large volume of data populate the queue, the $M_h$...
CHAPTER 5. EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th>Experimental Phases</th>
<th>TO = 30s</th>
<th>TO = 60s</th>
<th>TO = 90s</th>
<th>TO = 120s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stable</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 2</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 3</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 4</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 5</td>
<td>121</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 6</td>
<td>281</td>
<td>24</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Insertion 7</td>
<td>16069</td>
<td>7242</td>
<td>4578</td>
<td>3342</td>
</tr>
<tr>
<td>No OP</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.6: The number of hotspots detected by the hotspots detection approach in different experimental phases. For each experiment, the timeout value (TO) is set to be 30 seconds, 60 seconds, 90 seconds or 120 seconds.

is still determined to be a hotspot if the TCP queue cannot be evacuated to be empty within 30 seconds. Timeout values of 30 seconds and 60 seconds are too sensitive to report a true hotspot in the storage system. When the timeout value is set to be 90 seconds or 120 seconds, the situation of the number of hotspots is similar to that shown in Table 5.4 and Figure 5.8.

However, the timeout value should not be too big, because a big timeout value like 120 seconds would fail to report the hotspots in Insertion 6 experimental phase. Therefore, the timeout value should be set between 90 seconds and 120 seconds instead of 30 seconds or 60 seconds.

This work has not been followed up further in this thesis, but it does provide a helpful starting point for future work on network load balancing.

5.5 Summary

Firstly, this chapter has investigated CPU running time, the memory usage and RPC messages of the original Chord-based storage system. CPU running time is significantly affected by the frequency of the file operations, and the memory usage is affected by the number of virtual nodes in each EM. Two types of RPC messages, namely data messages and maintaining messages, are compared during the running of the Chord-based storage system. The comparison indicates that only a small part of
network traffic is contributed by the maintaining RPC messages. This provides freedom to introduce the Hilbert table into each EM to embed the Hilbert curve approach in the Chord-based storage system.

Secondly, the experimental results shows that the P_VNode storage load-balancing approach improves the storage utilisation of the Chord-based storage system from three perspectives:

- P_VNode provides more uniform distribution of the EMs’ storage utilisation than the natural load-balancing of Chord;
- P_VNode extends the time duration from FWTP to UTP and provides more reliable running time for the Chord-based storage system than one without the P_VNode load-balancing approach;
- Because the advent of UTP is postponed by P_VNode, the system storage capacity is increased in the Chord-based storage system.

Thirdly, the operational capacity of the Chord-based storage system has been investigated in this chapter. The experimental results show that both file insertion and file retrieval have the similar operational capacity range, somewhere between 40 seconds operation interval and 30 seconds operation interval.

Finally, by monitoring the TCP send queue in EMs, a hotspot detection approach has been designed at the end of this chapter. Four timeout values are tested for the hotspot detection approach in the experiments. In the experimental environment as described in Section 4.4, the timeout value should be set between 90 seconds and 120 seconds in order to achieve an appropriate detection sensitivity.
Chapter 6

Conclusion

6.1 Overview of Thesis

The research reported in this thesis studies a Chord-based storage system in a cluster environment, in which machines have good and persistent network connection, in order to make the Chord-based storage system provide the functionality of a data centre. As a DHT, Chord has the following advantages to accomplish this target.

- Chord has flexibility and scalability to maintain the participating machines that contribute their local storage space to the whole storage space of the Chord-based storage system, according to the maintaining mechanism described in Section 1.2.2.

- Chord has inherited DHT’s natural load-balancing, which ensures a roughly even distribution of the stored key-value pairs across all participating machines in the Chord-based storage system. The natural load-balancing property of DHTs is surveyed in Section 2.2.

- As a DHT, Chord defines the put (file insertion) and get (file retrieval) operations for key-value pairs in a distributed environment as described in Section 1.2. These two operations have already provided the data access mechanism for the Chord-based storage system.

Because each participating machine has a limited storage space, storage load-balancing across the storage space of all machines is important for a Chord-based storage system. Three important time points are defined by this research to indicate the storage status of the whole Chord-based storage system:
1. **FWTP** is the time when any machine reaches $X\%$ storage space utilisation in the system.

2. **AWTP** is the time when all machines reach $X\%$ storage space utilisation in the system.

3. **UTP** is the time when any machine reaches 100% storage space utilisation in the system.

FWTP is used for warning purposes and UTP indicates that the Chord-based storage system is unreliable afterwards. For the purposes of experiments, $X\%$ is set to 80\% as an initial value on which to base evaluation.

This research on the Chord-based storage system is evaluated by an experimental method in a practical experimental environment, which consists of five teaching laboratories in the author’s school. An experimental platform has been developed for the experiments with two main purposes:

1. To evaluate the original Chord-based storage system from all possible perspectives.

2. To evaluate the storage load-balancing improvements of the Chord-based storage system with P_VNode storage load-balancing approach included.

According to the survey of DHT’s natural load-balancing and their extra load-balancing approaches in Chapter 2, the existing storage load-balancing still needs to be improved if Chord is used to implement a Chord-based storage system, although a certain degree of storage load-balancing is achieved by the natural load-balancing approach and the virtual node approach which achieves some extra load-balancing. In the experimental results, as demonstrated in Section 5.3.1, the decrease of $D_{utilisation}$ from 283.36 to a value between 67 and 68 demonstrates that Chord’s natural load-balancing property improves the uniformity of the distribution of the storage utilisation across all machines. From the perspective of the time points, the original virtual node load-balancing approach extends the time duration from experiment start to FWTP and from FWTP to UTP with an increasing number of virtual nodes per machine, according to the experimental results in Section 5.3.2. This time duration extension means that the original virtual node load-balancing approach achieves a better storage load-balancing when the number of virtual nodes per machine increases.
Combining the virtual node approach with network proximity information, the P_VNode storage load-balancing approach is proposed by this research. It uses a virtual node migration method to reduce the burden from any potentially storage overloaded machines. According to the experimental results, as presented in Section 5.3.1 and Section 5.3.2, P_VNode achieves a more uniform distribution of the storage utilisation across all machines and extends the time from FWTP (when P_VNode starts to run) to UTP even more than the Chord-based storage system without P_VNode. Furthermore, in one experiment the maximum system storage utilisation of the Chord-based storage system is increased from 43.14% to 61.82%, which means that more storage space of the whole system storage is accessible by the users of the Chord-based storage system.

In order to efficiently complete the virtual node migration operation of P_VNode, the Hilbert curve approach is embedded in the Chord-based storage system by introducing a Hilbert table into each virtual node on the Chord Ring. The introduction of the Hilbert table is supported by the experimental results of the RPC messages in the original Chord-based storage system, which confirm that the maintaining RPC messages form a small portion of the whole network traffic in the experimental environment. The Hilbert curve is one of a number of Space Filling Curves (SFCs) that can achieve a similar distance preserving mapping. The Hilbert curve is chosen by using Spearman’s correlation coefficient of a ranking test that is designed by this research in order to compare the Hilbert curve against two other SFCs, the Peano curve and the RBG curve. The Hilbert curve’s correlation coefficient is 0.91, 0.73 and 0.74 in 2, 3, and 4 dimensional space with order of 4. The corresponding values of the other two SFCs are 0.66, 0.70 and 0.70 for the RBG curve and 0.61, 0.58 and 0.48 for the Peano curve. The Hilbert curve’s correlation coefficient is closer to 1, which means that more ranking is preserved by the Hilbert curve than by the other two SFCs.

As a secondary study of network issues in the Chord-based storage system, the operational capacity and the hotspot detection approach have been investigated by this research. Both file insertion and file retrieval have similar operational capacity ranges, between 40 seconds interval and 30 seconds interval. For the experimental environment as described in Section 4.4, the timeout value should be set between 90 seconds and 120 seconds in order to achieve an appropriate detection sensitivity.

Albeit that this section summarises the tasks that have been accomplished by this research, there are still some open questions about the research presented in this thesis. These are presented next.
6.2 Critique

This research does not involve all aspects of a Chord-based storage system. This research omits investigation of the appropriate fixed block size into which the files to insert are broken. The value (1 MB) used in this research follows the implementation of the original Chord-based storage system. However, the maximum block size can affect the uniformity of storage usage across all machines. A more complete investigation would investigate this effect.

When the Hilbert ID of a machine is produced, this research does not consider the dynamism of each machine’s network proximity information. The Hilbert ID of each machine is fixed in the Chord-based storage system. If the dynamism of the Hilbert ID of machines is taken into consideration, more precise network proximity information could be provided by the Hilbert curve approach. However, an extra Hilbert ID updating mechanism would be required.

The restrictions of the experimental environment is another omission of this research. The experiments are performed in the teaching laboratories, in which the experiment time and duration is restricted to be around 10 hours at night. There is no continuous long time running of the experiments since the machines are occupied during the day by students for teaching purposes. Another problem of the experimental environment is the availability of the machines. Different experimental machines may be available for the experiments on different days. Therefore, the comparison experiments and continuous experiments need to be carried out further by checking the difference between the mixtures of EMs. The third problem of the experimental environment is the number of available machines. Because the experiment environment is public, some experimental machines may be logged in by other users for their work, even at night when the experiment is running. This is the reason why only 150 machines can be reliably used for the experiments.

This research deploys the Chord-based storage system in a LAN-based cluster experimental environment in which all machines are connected to the same network router. Therefore the experimental results from this research may be compatible only with this experimental environment, because the experiments are not performed in other types of environment with different underlying network infrastructures.

The clustering result of the Hilbert curve is only evaluated by the theoretical ranking test using the Spearman’s correlation coefficient. In the practical experimental environment, the clustering result is not evaluated because of the similarity of the network distance between any experiment machines connecting to the same router.
The preparation of each experiment and the development of both the storage system and the experimental platform have prolonged the duration of this research. This makes this research omit some aspects, such as the threshold of the P\textsuperscript{V}Node load-balancing approach, $X\%$. Only an initial value 80\% is used in all experiments that currently have been completed. The suggestion of a network load-balancing approach that is presented in Section 3.5 is not evaluated due to the limited time to carry out a series of experiments as completed for the P\textsuperscript{V}Node storage load-balancing approach. More threshold values and the evaluation of the network load-balancing approach would be required.

As a result of these deficiencies, further work is desirable for the research of the load-balancing approach for the Chord-based storage system to implement the data centre functions. This is presented next.

6.3 Future Work

Because the fixed block size is not taken into consideration in existing experiments, further experiments are required to compare the Chord-based storage system with different fixed block sizes. A smaller fixed block size can provide more uniform distribution of storage usage by all machines, which may lead to a further increasing of the system storage capacity. Nevertheless, smaller fixed block size can also increase the number of blocks for a file to insert. Consequently, more key-value pair operations connecting to other machines are required to accomplish a file operation.

Currently, the P\textsuperscript{V}Node load-balancing approach’s migration is based on single virtual nodes, because this can maximally utilise the existing operations in the original Chord-based storage system. In the future, it is worth changing the P\textsuperscript{V}Node system to work on a single key-value pair, because this provides more control on the size of the data to migrate. The system storage capacity of the Chord-based storage system could be improved further.

Looking for a more reliable experimental environment is another future work, which will provide an opportunity to apply the Chord-based storage system in different experimental environments. In a reliable experimental environment, long-term continuous experiments can be performed to evaluate the load-balancing of the Chord-based storage system.
At the end of Chapter 3, this research makes a suggestion for a network load-balancing approach to reduce the number of hotspots in the Chord-based storage system. However, there are no meaningful experimental results for this replication-based network load-balancing approach. Therefore, as future work, the experimental results of this part should be processed, which may require repeating some of the experiments reported here.

6.4 Summary

Overall, the research presented in the thesis shows some promise for using an enhanced Chord-based system as a distributed storage system, but more work is needed to obtain a more complete evaluation.
Bibliography


