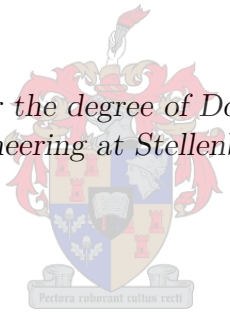
A State Management and Persistency Architecture for Peer-to-Peer Massively Multi-user Virtual Environments

by

John Sebastian Gilmore



Dissertation presented for the degree of Doctor of Philosophy in the Faculty of Engineering at Stellenbosch University

Promoter: Dr. H.A. Engelbrecht

March 2013

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety of in part submitted it for obtaining any qualifications.

Date: 26 February 2013 .....

Abstract

Recently, there has been significant research focus on Peer-to-Peer (P2P) Massively Multi-user Virtual Environments (MMVEs). A number of architectures have been presented in the literature to implement the P2P approach. One aspect that has not received sufficient attention in these architectures is state management and state persistency in P2P MMVEs. This work presents and simulates a novel state management and persistency architecture, called Pithos.

In order to design the architecture, an investigation is performed into state consistency architectures, into which the state management and persistency architecture should fit. A novel generic state consistency model is proposed that encapsulated all state consistency models reviewed. The requirements for state management and persistency architectures, identified during the review of state consistency models, are used to review state management and persistency architectures currently receiving research attention.

Identifying some deficiencies present in current designs, such as lack of fairness, responsiveness and scalability, a novel state management and persistency architecture, called Pithos, is designed. Pithos is a reliable, responsive, secure, fair and scalable distributed storage system, ideally suited to P2P MMVEs. Pithos is implemented in Oversim, which runs on the Omnet++ network simulator. An evaluation of Pithos is performed to verify that it satisfies the identified requirements.

It is found that the reliability of Pithos depends heavily on object lifetimes. If an object lives longer on average, retrieval requests are more reliable. An investigation is performed into the factors influencing object lifetime. A novel Markov chain model is proposed which allows for the prediction of objects lifetimes in any finite sized network, for a given amount of redundancy, node lifetime characteristics and object repair rate.

Samevatting

Onlangs is daar 'n beduidende navorsingsfokus op Eweknie Massiewe Multi-gebruiker Virtuele Omgewings (MMVOs). 'n Aantal argitekture is in die literatuur beskikbaar wat die eweknie benadering voorstel. Een aspek wat nie voldoende aandag ontvang in hierdie argitekture nie is toestandsbestuur en toestandsvolharding in eweknie MMVOs. Hierdie werk ontwerp en simuleer 'n nuwe toestandsbestuur- en toestandsvolhardingargitektuur genaamd Pithos.

Ten einde die argitektuur te ontwerp is 'n ondersoek uitgevoer in toestandskonsekwentheidargitekture, waarin die toestandsbestuur- en toestandsvolhardingargitektuur moet pas. 'n Nuwe generiese toestandskonsekwentheidargitektuur word voorgestel wat alle hersiene toestandskonsekwentheid argitekture vervat. Die vereistes vir die toestandsbestuur- en toestandsvolhardingargitekture, geidentifiseer tydens die hersiening van die toestandskonsekwentheidargitekture, word gebruik om toestandsbestuur- en toestandsvolhardingargitekture te hersien wat tans navorsingsaandag geniet.

Identifisering van sekere leemtes teenwoordig in die huidige ontwerpe, soos 'n gebrek aan regverdigheid, responsiwiteit en skaleerbaarheid, lei tot die ontwerp van 'n nuwe toestandsbestuur- en toestandsvolhardingargitektuur wat Pithos genoem word. Pithos is 'n betroubare, responsiewe, veilige, regverdige en skaleerbare verspreide stoorstelsel, ideaal geskik is vir eweknie MMVOs. Pithos word geïmplementeer in Oversim, wat loop op die Omnet++ netwerk simulator. 'n Evaluering van Pithos word uitgevoer om te verifieer dat dit voldoen aan die geïdentifiseerde behoeftes.

Daar is gevind dat die betroubaarheid van Pithos afhang van die objek leeftyd. As 'n objek gemiddeld langer leef, dan is herwinning versoeke meer betroubaar. 'n Ondersoek word uitgevoer na die faktore wat die objek leeftyd beïnvloed. 'n Nuwe Markov ketting model word voorgestel wat voorsiening maak vir die voorspelling van objek leeftye in eindige grootte netwerke, vir gegewe hoeveelhede van oortolligheid, nodus leeftyd eienskappe en objek herstelkoers.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- my promotor, Dr Herman Engelbrecht, for his continued guidance and support;
- MIH, for their financial assistance towards this research and the many opportunities they provided;
- my parents, John and Coreen Gilmore, for making me the man I am today and making my studies possible;
- Dr Shun-Yun Hu, Dr Gregor Shiele and Dilum Bandara, for their valuable input throughout my studies;
- Francois van Niekerk, Peter Hayward and Rampie Bell, for all the interesting ideas, talks and feedback;
- all the MIH media lab members, for making my time there so enjoyable;
- the Holy Father, for keeping me and blessing me with so much.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

To my wife, Jacki Gilmore, for all your love, support and understanding.

Contents

List of Figures

List of Tables

Nomenclature

Statistical distributions

α	Heavy-tailed degree of the Pareto distribution
β	Scale parameter of the Pareto distribution
λ	Exponential rate parameter
$h(x)$	Failure or hazard rate of a distribution

Pithos performance

R_{Pithos}	Reliability of Pithos
R_{group}	Reliability of group storage
R_{overlay}	Reliability of overlay storage
P_{group}	Group probability

Storage networks

N	Maximum number of nodes in the network
n	Number of nodes currently in the network
$\tilde{n}$	Mean number of nodes in the network
R	Required number of replicas
r	Number of replicas currently in the network
μ	Object repair rate
T_{repair}	Object repair period
θ	Departure rate of peers under steady state
ϕ	Arrival rate of peers under steady state

Markov chains

i	The number of object replicas in the network in the current state
k	The number of nodes in the network in the current state

j	The number of object replicas in the network in the next state
l	The number of nodes in the network in the next state
$p(ik, jl)$	State transition rate moving from the dual state ik to the dual state jl
S	Number of states in a Markov chain
$S_{\text{transient}}$	Number of transient states in a Markov chain
t_{ik}	Time spent in the dual state ik
$\mathbf{P}$	Transitional rates matrix
$\hat{\mathbf{P}}$	Normalised transitional rates matrix
$\mathbf{I}$	Identity matrix
$\mathbf{Q}$	Transient transitional rates matrix
$\mathbf{N}$	Fundamental matrix
$\mathbf{L}$	Object lifetime vector

Units

s	Seconds
Bps	Bytes Per Second
bps	Bits Per Second
%	Percentage

Abbreviations

AI	Artificial Intelligence
ALM	Application Layer Multicast
AoE	Area of Effect
AoI	Area of Interest
AoP	Area of Propagation
C/MS	Client/Multi-Server
C/S	Client/Server
CDF	Cumulative Distribution Function
DHT	Distributed Hash Table
DOS	Denial of Service
FPS	First-Person Shooter
ID	Identifier

IM	Interest Management
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MMO	Massively Multiplayer Online
MMORPG	Massively Multiplayer Online Role-Playing Game
MMOSG	Massively Multiplayer Online Social Game
MMVE	Massively Multi-user Virtual Environment
MTTF	Mean Time To Failure
NEO	New Event Ordering
NFS	Network File System
NPC	Non-Player Character
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PC	Player Character
PCU	Peak Concurrent Users
PDF	Probability Density Function
RACS	Referee Anti-Cheat Scheme
RPC	Remote Procedure Call
RPG	Role-Playing Game
TCP	Transport Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
VE	Virtual Environment
VoIP	Voice over IP
WoW	World of Warcraft

Chapter 1

Introduction

1.1 Research objective

The objective of this work is to design and develop a novel state management and persistency architecture, specifically designed for P2P MMVEs that satisfies all P2P MMVE storage requirements and takes into account the requirements of the consistency architecture in which it will operate.

1.2 Massively Multi-user Virtual Environments

Massively multi-user virtual environments (MMVEs) are characterised by thousands of users interacting in the same virtual environment; socially, cooperatively or competitively. MMVEs can be serious, such as air traffic control simulations and military war games or casual, such as computer games. MMVEs as computer games are referred to as massively multiplayer online games (MMOGs) and can themselves be hardcore, where significant investments of time and money are required, or casual, where little time and money are required. Hardcore games usually also have better graphics, and require large amounts of computing power, storage space and Internet bandwidth, compared with their casual counterparts.

In this work, a focus is placed on hardcore MMOGs, or MMVEs that classically require large server clusters to host the virtual environment and consume significant amounts of Internet bandwidth.

With the advent of broadband Internet, MMOGs have seen tremendous growth over the past decade, growing from less than 500,000 active subscribers in 1999 to over 21 million in 2011 [112]. In 2011, the MMOG market was a $2,6 billion industry in the United States alone [83]. MMOGs are characterised by expansive worlds, where a large number of players interact online with each other and the virtual environment to achieve certain goals through collaboration and teamwork.

From an academic perspective, MMOGs also hold great value. An MMOG is a complex networked application, with clients requiring reliable real-time feedback on actions taken. The design of an MMOG requires in-depth knowledge of server architectures and network design. The design of a server architecture determines how many players the game will support and what the user experience will be in terms of quality of service.

1.2.1 Modern MMOG implementations

Some examples of modern hardcore MMOGs are: World of Warcraft, Eve Online and Second Life. These games are presented here to familiarise the reader with the classic MMOG tropes. World of Warcraft presents a fantasy MMORPG that occurs in some tolkienesque imagined world, where players are heros trying to save the world from some great evil.

Eve Online is situated in space, where a player travels from one solar system to the next. Players can partake in mining, trading or pirating activities. The game is based around trade and space exploration.

What distinguishes Second Life from the first two games mentioned is its focus on player generated content. This is as opposed to the first games that are more focused on consuming the available game content.

Throughout the development of MMOGs, role play has been tightly coupled to this type of game. This is perhaps due to the exploration and player interaction aspects. Role play allows players to fully immerse themselves in the game world and might, therefore, provide for a more compelling experience. Because of this tight coupling, the terms massively multiplayer online role-playing game (MMORPG) and MMOG have almost become synonymous. Throughout this work, a distinction will, however, be made between the two, where MMORPG refers to the specific genre and MMOG refers to the “massive” and “online” characteristics of the game.

1.2.1.1 World of Warcraft (Fantasy MMORPG)

An MMORPG that has been very lucrative and has become well known in Western culture is Blizzard’s World of Warcraft (WoW). In WoW, players are represented by avatars that inhabit a virtual fantasy world. An avatar has a race, a class, attributes, skills and professions. In the virtual world there are quests that a player may undertake to gain experience in classic role-playing game (RPG) style. Gaining experience allows a player to gain levels, which improves its skills and attributes, making the player more powerful.

There are different reasons why players play the game. Some players play the game socially, to meet new people and make friends, other players play the game

to become sufficiently powerful to play the end-game content. End-game content requires large groups of players to work together in a highly coordinated way to achieve some set of objectives, usually culminating in destroying a "boss". This activity is called "raiding", which is usually done by groups of players that have decided to play together and form a "guild". Guilds have complex social structures, which allows for various social interactions. Usually it is this high degree of social interaction that attracts players to MMOGs. Players are no longer playing by themselves in a lonely world, but rather playing with other players in a large open virtual space, waiting to be explored.

When a character is created in WoW, the creator must first choose a server on which the character will be stored. Characters on different servers cannot interact in the virtual world and cannot easily move between virtual worlds. Every server, which itself is a server cluster, hosts a complete copy of the virtual world. From a character perspective, the fact that there exists multiple copies of the game world is not known. This is termed sharding and will be discussed in detail in Section 3.2.1.

After eight years of operation, WoW still has 10,2 million subscribers, each paying $15 per month [87].

1.2.1.2 Eve online (Space MMORPG)

Eve Online, developed by CCP Games, brought many new innovations to the MMORPG. It was the first successful MMORPG to feature a science fiction theme and it was the first MMOG to have a single distributed server architecture. In 2006, CCP Games launched the largest supercomputer in the gaming industry to upgrade their existing infrastructure and enable Eve to support more than 50,000 concurrent users [10]. This number was surpassed in 2010 with 60,453 concurrent users in-game [21].

Another innovation of Eve was the in-game economy. CCP games appointed Dr. Eyjólfur Guõmundsson as chief economist of Eve online in 2006 [74]. His duties were to monitor and predict market trends in the game world and produce detailed quarterly economic reports [53]. The economy is based on an open market system, ruled by supply and demand. No other game has implemented an in-game economy in such a rigourous fashion.

1.2.1.3 Second life (MMOSG)

Second life is classified as a massively multiplayer online social game (MMOSG). It focusses more on social interaction and creatively, as opposed to the usual conflict-based MMORPGs, such as WoW or Eve online. Players in Second Life can create virtual items, such as clothing, furniture and architecture, and sell them for real

money. Players can buy property and build on the property they bought. This can be sold to other users, also for real money.

From a network architecture perspective, user generated content changes adds a lot of extra load to the system. Players no longer only have to be aware of other players in the virtual world, they also have to be made aware of the content that other players generate. Usually, the player's client also know exactly how another player looks, based on her class and equipped items. With user generated content, the complete shape of the object is transferred. User generated content, therefore, increases bandwidth requirements.

1.2.2 Requirements

The design requirements of an MMOG are the same as the design requirements for a classic single player game, with the added requirements of networking capability and scalability. Classic game design requirements include: a graphics engine, a physics engine, handling user input, game mechanics and logic, artificial intelligence, level design and the creation of art assets, sounds and music.

What is additionally required for an MMOG is a network and state consistency architecture. The network architecture defines how hosts are connected, the roles of different hosts and how information is distributed between hosts.

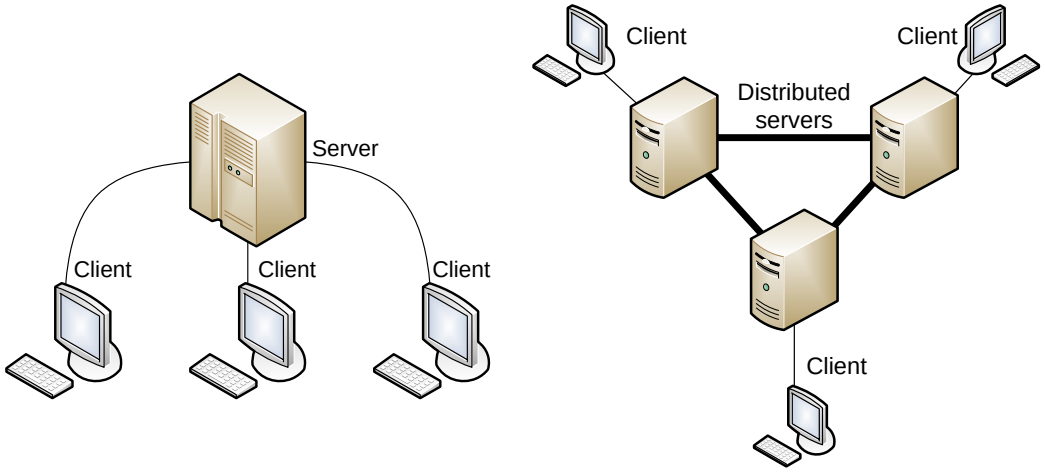
There are many social as well as technical aspects to consider when designing a virtual world [3], but an essential requirement of all MMVEs, including all the MMOGs presented in Section 1.2.1, is that multitudes of users should be able to interact with each other and the virtual environment. This is called the consistency architecture. The consistency architecture ensures that users share the same view of the virtual environment they inhabit. It is also responsible for relaying user actions to other users and informing other users of any new users or objects in the virtual world.

User data should also be stored when users log off from the virtual environment. Virtual object states should also be stored as well as the states of computer characters in the virtual environment.

A separation between authoritative and non-authoritative state is also required in MMVEs. If two users do not agree on the state of the virtual environment, due to latency or cheating, is must be possible to return the virtual environment to a state on which both users can agree. This "true" state is called the authoritative state, with users usually possessing non-authoritative states. When two entities do not agree on the environment state, authoritative state is used as a state on which both can agree upon and revert to.

1.2.3 Classic client-server MMVEs

A classic network architecture, used in the design of all MMVEs presented in Section 1.2.1 and in all commercially successful MMOGs to date is the client-server (C/S) network architecture.



(a) Client/Server (b) Client/Multi-Server

Figure 1.1: Network architectures

Figure 1.1a shows the C/S model. The server is the entity on which the MMOG is hosted and is controlled by the operator. The server contains the authoritative environment state, to which clients must revert if their states deviate from the authoritative state. Clients are computers operated by users that connect to the server to be able to interact with the virtual environment and other users in it. The server is responsible for handling all queries from clients. Clients never communicate with other clients; they send their actions to the server and receive the updated states of other users from the server. All client environment states are non-authoritative.

1.2.3.1 Advantages

The C/S architecture has two main advantages that has made it the architecture of choice for all MMOG developers. Because of the centralised approach of the architecture, both administration and security are greatly simplified. Administration is simplified, because the operator has full control over the server, server data and code. Efficient logging is also supported, because the server is able to not only log all server actions, but also all client actions.

Security is a significant issue in MMOGs, since some users sell virtual currency for real-world currency [33]. This makes the MMOG a platform that is capable of producing income, which increases the incentive of users to gain an unfair advantage over others. The more popular an MMOG, the greater the security threat. Because the operator has full control over the server code and is never required to furnish the client with the server code, a potential attacker never has any knowledge of the server architecture and code. Because clients are never allowed to communicate directly, all malicious users can be filtered out of the network by the server when detected and even banned from the network.

Operators are able to ban users, since these virtual environments usually require an account, which is linked to a copy of the software as well as some payment method. This introduces a large cost to users whose accounts are banned. The server or cluster is also housed in a secured location, where access can be controlled. These factors simplify the security of the C/S model by allowing the developers to place all intelligence in the server.

1.2.3.2 Disadvantageous

The C/S architecture has the following disadvantages: weak robustness, weak scalability, high cost to the operator, high latency, high amount of required server bandwidth and weak handling of transient loads.

- The robustness of the system is weak because it is a single point of failure. If the server fails or goes down for maintenance, the virtual environment is off-line and users are unable to interact with it.

- The system is also weakly scalable, since a single server cannot easily be extended with more resources. Even if an off-line approach is used, where hardware is upgraded after the system is taken down for maintenance, the hardware required to support a virtual environment hosting more than 3000 users, become prohibitively expensive, as described in Section 1.2.4.

  The server hardware should be able to support peak system loads, which means that sufficient resources should always be provisioned to support these peak load. This is not an economically viable solution, because resources to handle peak loads are not used most of the time. This translates to operators paying for the provisioning of resources, without having active users that pay for these resources.

- Because no clients are allowed to communicate with other clients, every change that is made to the virtual environment by a client, first has to be communicated to the server, which in turns relays this message to all clients after

applying environment logic and artificial intelligence (AI) algorithms. This two hop path, with the additional time for computation added by the server as well as possible buffering at the server when many clients communicate, significantly increases the latency of the system compared to a system where direct communication is used.

1.2.3.3 Client/Multi-Server

In an effort to address some of the C/S issues, the distributed C/S, also called the Client/Multi-Server (C/MS) model, was introduced, shown in Figure 1.1b. In a C/MS model, the server functions are distributed amongst multiple machines to distribute the server load.

In general, the issues addressed and improved by the C/MS architecture are robustness, scalability, and peak load handling. The system is more robust, because the failure of one server will not necessarily lead to the failure of the whole system for certain system designs. The system is more scalable, because many less powerful servers may be used, which allows for the hosting of more users than what is currently possible with single server hardware. It also handles transient loads better, because, for cases where loads can be predicted, resources can by moved between servers to improve the user experience.

The disadvantages of this system is that the administration complexity is greatly increased. Such systems, although capable of handling many more users than a single server, is also more expensive. These disadvantages are, however, not technical problems and so it is assumed for current virtual environments, that these systems are what is required if a virtual environment is to be hosted for a large number of users.

1.2.4 The cost of doing business

With the fast growing MMOG market, many companies are spending a significant amount of money to produce premium MMOG titles. Some development cost estimates are: $18 mil. for Aion, $20 mil., for Everquest [91], $63 mil. for World of Warcraft [31] and $100 mil. to $200 mil. for Star Wars: The Old Republic [86], [96]. Although these figures are purely estimates, it does show that to develop a premium MMOG title costs a lot of money.

The issue with MMOG development is that, although they cost more to develop than single player or smaller scale multiplayer games, they are just as likely to fail. Despite this, game publishers are spending a lot of money in an attempt to recreate the success that is World of Warcraft.

Because of the large revenues being generated from MMOGs, many competitors are entering the MMOG space. Currently, the rate at which new MMOGs are added to the market is outstripping the growth of the market itself [83]. Furthermore, because of the recession, over the past two or three years, game subscriptions have been shown to stabilise or even decline [112].

The significant initial investment required to develop an MMOG also does not present the complete picture. Another factor driving up costs for an MMOG is the money required for server hardware, maintenance and support. An MMOG is not finished when it goes live. A team of developers is required to maintain the game, release patches fixing bugs and to produce more content to keep the player base sufficiently interested to ensure that players will continue to pay $15 per month to play. Development and maintenance costs for World of Warcraft for four years is estimated at $100 mil. to $200 mil. [31].

With the costs involved, it is therefore difficult for a new developer to enter into this space. After the large initial investment into the game's development, all server hardware must be acquired and staff appointed to maintain the game. This money is spent before it is known whether the game will succeed or fail. It has been estimated that during the lifetime of an MMOG, 80% of the game revenue goes into hardware and maintenance costs [65].

1.2.5 The peer-to-peer proposal

To our knowledge, the first distributed multi-user virtual environment was MiMaze, developed in 1998 by Gautier and Diot [48]. This revealed a new research field, which attempted to establish the peer-to-peer (P2P) model as a viable alternative to the classic C/S and C/MS architectures. In 2004, Knutsson et al. proposed the first scalable architecture to host massively multiuser virtual environments [66].

P2P MMVEs form the focus of this work. There are various advantages to moving from C/S to P2P in MMVEs. These include: increased robustness, improved scalability, lower operator costs, improved handling of transient user load and lower latencies. The advantages are described in Section 1.4.1 in detail, but firstly it would be beneficial to acquire a greater understanding of the basics of the P2P network model.

1.3 Peer-to-Peer networks

1.3.1 Definition

One definition of a P2P system is: "A Peer-to-Peer system is a self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed

resources in a networked environment avoiding central services." [84]. There are various other, slightly different, definitions of P2P networks [97], but the one shown above is sufficient for the purposes of this work.

What the definition means is that a P2P network is a distributed network that exists out of many participating peers and has the following properties [92]:

- *High degree of decentralisation*: No or little centralised control exists. Server functionality is distributed amongst all peers.
- *Self-organisation*: Little or no manual organisation is required in the network. Peers are given an initial IP to allow them to join the network, but thereafter new neighbours are automatically acquired and peers remain connected to the network, even with other peers joining and leaving.
- *Multiple administrative domains*: Peers are not under the control of any single authority. Peers in the network belong to different organisations or individuals and direct administration is impossible.

P2P systems have been popularised by mainly three systems developed in 1999: the Napster music sharing service, the Freenet data store and the SETI@home volunteer-based distributed computing project. These three projects highlighted the advantages of P2P networks being: low barrier to entry, scalability, resistance to faults and attacks, and an abundance and availability of resources.

1.3.2 The OSI model and P2P overlays

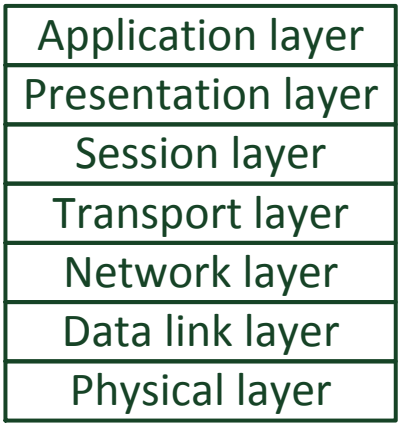


Figure 1.2: Layers of the OSI model

A basis of computer networking is the layered architecture model, called the open systems interconnection (OSI) model [127]. It defines various protocol layers that

allow for abstraction of complex operation in the lower layers in the higher layers. As shown in Figure 1.2, the OSI layers are, from bottom to top: the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer and the application layer. In practice, the session, presentation and applications layers are usually all folded into the application layer.

The physical layer is the physical transmission medium and carries physical signals. Physical level protocol standards include: IEEE 802.11 (Wi-fi), USB, Bluetooth, etc.. The data link layer, sometimes referred to as the MAC layer in the Internet protocol stack, carries frames and is responsible for point-to-point data transfer on the same local area network (LAN). Signals from the bottom layer are converted into bits and sequences of bits are grouped into frames, which is seen to be sent over the data link layer. As can be seen, every higher layer abstracts elements of the lower layer to reduce complexity.

The network layer allows communication between different LANs, using routers, gateways and host addressing. Every computer in a network is referred to as a host. A well known network layer protocol is the Internet protocol (IP), which routes all Internet traffic. The transport layer is referred to as the end-to-end layer and represents the abstract connection between a source and destination host, where all intermediate routers and LANs may be ignored, since they are abstracted away by the network layer. Well known protocols in this layer include the user datagram protocol (UDP) and transport control protocol (TCP). Above the transport layer, for the purposes of this work, is the application layer. The application layer has access to all network services, including routing and reliable transmission.

In the C/S architecture, the client and server are both located in the application layer and communicate with each other using the transport layer. They need not be aware of any of the lower layers.

P2P networks are created and maintained in the application layer of the OSI model, called the P2P overlay. An overlay is required so peers may know which other peers are part of the P2P network, since most nodes on the Internet will not be part of the network. The overlay is then defined by the routing table information stored on each peer. The overlay can be thought of as an additional layer in the OSI model. The overlay interfaces with the transport layer and P2P applications are one layer higher and communicate by using the overlay.

Peers in an overlay network might have neighbours that have no relationship to their physical position in the underlying physical network.

1.3.3 Structured and unstructured P2P overlays

Overlays can broadly be classified into structured and unstructured types. The classification is mostly based on the differing methods of routing and content retrieval in the network. This section only provides a brief comparison between structured and unstructured overlays. For a detailed comparison between the two types that also deals with many of the myths of structured overlays, please refer to [19].

With unstructured approaches, one is never assured that a data item will be retrieved, even if that data item is present in the network. If many duplicates of a data item are contained in the network, this becomes less of a problem, since it is assumed that the request will be routed to some set of peers that do possess the item.

An unstructured architecture is well suited to content sharing, for example: P2P TV, BitTorrent, and Gnutella. The reason for this is the high level of duplication in these networks, especially for popular content. It is also easier to perform keyword searches in unstructured networks and the overlay requires less maintenance.

Structured overlays have been proposed that provide for efficient routing and reliable retrieval of data items. Some well known structured overlays are: CAN [88], Chord [104], Tapestry [126], Pastry [90] and Kademlia [75]. The basic principle of a structured overlay is that peers have to follow a certain set of rules on how to connect themselves to other peers, such that the emerging overlay topology follows a defined structure.

In structured and unstructured overlays, all nodes are identified by unique identifiers (IDs). A popular method to create the IDs in structured overlays is to use hashes to a circular key space, using for example, the SHA-1 hash function. Any node in the overlay network is then able to efficiently route a query with a given ID, to a node with an ID closest to the given ID. An accurate comparison is that unstructured overlays are good at finding “hay”, while structured overlays are good at finding “needles” [92].

1.3.4 Features of structured P2P overlays

A structured P2P overlay has certain key features that determines its lookup speed, space consumption and bandwidth requirement [100]. These features are network topologies, routing algorithms, join/leave mechanisms, routing table maintenance and bootstrapping.

1.3.4.1 Network topology

An overlay's network topology determines how nodes are structured in the application layer network. The network topology allows for deterministic routing and determines the number of required lookup hops and how the network will be maintained during times when nodes join and leave the network.

1.3.4.2 Routing algorithms

The routing algorithm is directly related to the specific network topology. The routing algorithm determines which nodes are traversed when a message is sent to a target node. The routing algorithm and the network topology determine the number of expected hops for a message to reach a destination.

1.3.4.3 Join mechanism

P2P networks experience constant churn, where nodes are joining and leaving the network. Mechanisms for a node to join the network should be present. This is usually in the form of a well known directory (bootstrap) server. When a node wishes to join the network, the directory server sends a set of nodes that the node may want to join.

1.3.4.4 Leave mechanism

If a node leaves the overlay, its neighbours have to be informed so they may update their routing tables. This is termed *routing table maintenance* and has to happen every time a peer leaves the network. Neighbouring nodes of a node that left the network might also have to inform their neighbours of the change. Two types of routing table maintenance exist: opportunistic and active maintenance.

Opportunistic maintenance attaches routing information to existing request packets. Routing data is essentially "piggy backed" onto existing messages. This reduced messages overhead but the rate of routing table maintenance is directly proportional to the message rate. Active maintenance makes use of explicit update messages, which requires more bandwidth but is more reliable.

1.3.4.5 Bootstrapping

After having been informed of a peer to join, a joining peer may initiate the process of positioning itself in the P2P network, called bootstrapping. This is the process of finding out where the joining node fits into the overlay in terms of the network topology. If the overlay requires that all nodes be connected in a ring organised by

ID, then the joining peer must discover the nodes whose IDs are one more and one less than its own and join those two peers.

The process of bootstrapping is complete when a joining peer becomes a functioning member of the P2P overlay.

1.3.5 Advantages

Many advantages can be obtained from using a P2P network, but it should be noted that the P2P network should be designed to provide these advantages. The advantages listed here are not automatically provided out of the system being a P2P system.

- P2P networks have a low barrier to entry, since little or no centralised infrastructure is required to maintain the system. This makes P2P networks inexpensive to operate and is one of the reasons Napster was able to provide its service for free.
- P2P networks are considered scalable. Pure P2P networks can theoretically grow from hundreds to millions of peers, with the service remaining functional. This is all possible without the need for the operator to acquire more infrastructure, as opposed to the centralised client/server network, which requires more powerful server clusters as the network grows to handle the growing number of client requests.
- A P2P network is also resistant to faults and attacks, since the failure of a single peer has little to no effect on the network. This is because there are usually few peers that are critical to the correct functionality of the system. To incapacitate a P2P network, an attacker usually has to shut down a large proportion of the network.

1.4 Peer-to-Peer MMVE network architectures

P2P MMOGs are considered a sub-class of P2P Massively Multi-user Virtual Environments (MMVEs), a class that also includes large scale military simulators.

This architecture does, however, still have a few major issues that need to be solved before MMVEs can be developed that make use it. If these issues, discussed in Section 1.4.3, can be solved, a P2P architecture holds some powerful advantages over a C/S system.

The core idea of the P2P model is that each peer contributes sufficient resources to the network to host itself. This also means that all functions of the server in

the classic C/S model are distributed amongst all peers. The authoritative environment state should now also be distributed amongst the peers in the network. Each peer will, therefore, contain a section of non-authoritative state for its own display purposes, and contain sections of the authoritative environment state.

1.4.1 Advantages

- In P2P MMVEs, it is generally assumed that peers provide abundant and highly available resources, such as computational power, and long term and short term storage. Peers are usually personal computers, with gigahertz of processing power, multiple processing cores, gigabytes of short-term storage and terabytes of long-term storage space. In other words, it is assumed that P2P MMVEs do not consist of resource-constrained peers.
- The system is scalable in terms of world size, number of peers, objects and number of events per second, because every peer contributes sufficient resources to the system to host itself.
- Because of the high scalability, no extra resources are required from an operator perspective, when more peers join the network.
- P2P networks also efficiently handle transient loads, since the joining peers contribute their own resources. If many users suddenly enter the virtual environment, no resource provisioning issues will arise, since peers already possess the required resources.
- P2P architectures create a lot of opportunities for independent developers, because a large initial investment is no longer required to purchase the expensive server hardware. Not only are hardware costs reduced, but running costs are also reduced.
- The bandwidth requirements of the virtual environment server is now shared amongst users, which means that little bandwidth costs will be incurred by the operator.
- Latency is improved, because it is now possible to directly communicate between peers and it is not necessary to communicate via a server. The distribution of the load as well as direct communication will further reduce latency.

1.4.2 Requirements

Key requirements have been identified for the creation of a P2P MMVE. What follows is an expansion and reinterpretation of the requirements identified by Schiele

et al. [95]. Some of those key requirements, that are specific to P2P MMVEs, are briefly highlighted in this section.

1.4.2.1 Distributed computation

Non-Player Characters (NPCs) are characters that are not controlled by any human user, but are rather controlled by some artificial intelligence routine or script executing on some host machine. These characters represent the traders and monsters in MMVEs and usually contain sets of rules that determine how they should interact with Player Characters (PCs) as well as their own state information. An NPC's state can be how much money and items it has to trade or how much health it still has after being attacked by a user.

Some virtual objects require computational power to function. An example of this is the Artificial Intelligence routines of NPCs or the computation of physics effects on virtual objects. In P2P MMVEs, it might be required to distribute these computational tasks to offload computation load from peers that do not have sufficient resources.

1.4.2.2 Consistency

A key challenge with any networked virtual environment is how to maintain state consistency between users in the virtual world. In other words, to ensure that all users perceive a virtual world in the same state. Solving the state consistency problem for P2P MMVEs is one of the major development challenges and forms the focus of this work. The challenge of state consistency will be described in detail in Chapter 2.

1.4.2.3 Persistency and state management

P2P MMVEs require persistent state, which means that the information of users in the virtual world should be maintained. The items that a user possesses in the virtual world should remain on his virtual avatar, even if she logs out and later back on to the virtual environment.

Persistency is different to state management. Persistency is the long term storage of data, usually on secondary storage. State management deals with managing the state of objects in the virtual world. This includes handling all requests for that object from peers who do not yet possess the object, or processing changes that users have made to the objects in the virtual world.

1.4.2.4 Interactivity

P2P MMVEs should be interactive and responsive. This means that they should support low latency interactions and possibly mask high latency operations, using techniques such as dead reckoning [4], so as not to break immersion. Interactivity means that all user actions should be handled as soon as possible.

1.4.2.5 Scalability

MMVEs are by definition massive, supporting thousands to tens of thousands of users in the same virtual environment, as discussed in Section 3.3.1. A P2P MMVE should support as many users as a classic MMVE, preferably more.

1.4.2.6 Security

Security, which includes cheating mitigation, has been identified as a major issue for P2P networks [66], [82], [47]. The challenges reside in the fact that peers are not under the control of the operator. Since the authoritative environment state is distributed amongst peers, all peers have access to sections of the authoritative environment state. Peers also have access to the code that was classically only housed on the server. This creates the opportunity that a peer might attempt to alter the authoritative state in ways that are not consistent with the environment rules.

One advantage that can be exploited to prevent cheating is that no peer contains all authoritative environment state and no single peer has more authority than another.

1.4.2.7 Efficiency

The network architecture of a P2P MMVE should be efficient. This usually goes without saying, but the reason why it is specifically stated is because the P2P network architecture basically implements the functionality of the server in the classic server model.

The server functionality has to be distributed accross all users of the P2P system. In addition to server functionality, a peer still requires the functionality of a classic client as well. This means the rendering of intricate graphics and calculation of complex physics. The P2P MMVE network architecture should, therefore, require as little as possible resources, of the resource that would ordinarily have been used for graphics and physics.

1.4.2.8 Incentives

P2P schemes require all users to share resources in order to ensure correct functionality. Users might, however, not want to share their resources, whilst still benefiting from the resources of others. The purpose of incentive mechanisms is to ensure that all users contribute resources, by incentivised contribution.

All distributed resource sharing models require incentive mechanisms. For example, Bittorrent systems use the tit-for-tat protocol to ensure that all people downloading data are also contributing data [29]. Such mechanisms are also required with P2P MMVEs. One advantage in designing an incentive mechanism for a P2P MMVE is that users can be made to contribute resources for the duration of play. The issues with file sharing systems are not present where a peer, after downloading a file, has no more incentive to contribute.

1.4.3 Key challenges

Using P2P MMVEs has many advantages, but some challenges still remain. The main challenges are state consistency, limited peer bandwidth, cheating mitigation, incentive mechanisms and distributed computation. This section will present some information on work that has been done in the different areas and talk about the maturity of the various solutions. In general though, most solutions are still research projects and none have yet been implemented as commercial products.

1.4.3.1 Peer bandwidth

In a paper by Miller and Crowcroft, a packet simulator was created to determine the required bandwidth and effective latency, if a game such as World of Warcraft were to be implemented using P2P technologies [77]. Their simulation results indicate that today's networks are not able to host P2P MMVEs, with the required bandwidth and latency constraints. Such a significant result requires verification, but at the least, it shows that reducing bandwidth and latencies for P2P MMVEs should be a primary design requirement.

1.4.3.2 Cheating mitigation

There are various security issues that are usually classified according to the level in the OSI protocol stack where they occur. The areas identified by [47] and expanded upon by [121] are: game level, application level, protocol level and infrastructure level. The description is consistent with the generally used layered security model [8].

- Game level cheats are ways in which a malicious user may gain an unfair advantage over other users, within the confines of the game. These cheats

are usually because of software bugs and some examples are duplication and teleport cheats.

- Application level cheats are where malicious users alter the game software to gain an unfair advantage. This is usually done by gaining access to the game state to which they should not have access at the current time. An example of this is "map reveal" cheats in strategy games, where the "fog of war" is removed and the user can observe all the opponent's movements. Other cheats that are used include augmenting the user's UI with extra information that allows the user to make more informed decisions.

- Protocol level cheats are cheats based on the different methods of communicating data across the system. These usually concern dropping, delaying of modifying IP packets to achieve certain outcomes in the game.

- Infrastructure level cheats concern exploiting the underlying infrastructure on which a game is built. Types of infrastructure cheats include denial of service (DOS) attacks on the P2P overlay to prevent messages that should be forwarded by a peer from being forwarded.

As with all taxonomies, all cheats may not cleanly fit into one if these boxes, some cheats may occur over multiple levels or a cheat with a specific outcome can be implemented differently on different levels. The field of P2P security has recently received more attention than in the past and has started to bear fruit [120].

This is, however, an ongoing research field with many issues still open. For an in-depth review of the security issues facing peer-to-peer system in general, refer to [118]. These issues are the same issues facing P2P MMVEs, with the exception of the game and application layer issues.

1.4.3.3 Incentive mechanisms

Some proposed incentive schemes increase a user's reputation when resources are provided [64] [107]. This might create a type of meta-game, where users try to gain as much reputation as possible. It can however be argued that this scheme does not really enforce the provisioning of resources. A user who does not want to provide resources might not see a higher reputation as sufficient incentive to provide resources.

Other issues with incentive schemes is that sometimes users might have insufficient resources. Such users should be aided by other users with sufficient resources and not be disallowed from interacting with the virtual environment. When limited resources are taken into account, the issue of reporting a false amount of available

resources becomes a problem. A peer that has sufficient resources, might report insufficient resources, to avoid being penalised. It is evident that there is a need for more research in this field.

1.4.3.4 Distributed computation

Some architectures assume that the computational requirements will be fulfilled where the object state is hosted [42], but other schemes exist that allow for the CPU power to be distributed amongst peers. One such scheme makes use of a "job board" like mechanism, where tasks are advertised on specialised peers. All peers monitor these specialised peers and may elect to perform the advertised tasks [38].

1.4.3.5 State consistency

The state of the art of state consistency techniques will be reviewed in detail in Chapters 2 and 3.

1.5 Related work

1.5.1 Mediator

The basis for this dissertation is the work done by Lu Fan, in his dissertation entitled: "Solving Key Design Issues for Massively Multiplayer Online Games on Peer-to-Peer Architectures" [37] as well as the paper by Fan et al.: "Design Issues for Peer-to-Peer Massively Multiplayer Online Games" [39].

In the works by Fan, a survey is performed as to the main challenges that are still present with P2P MMVEs at that time (2009). The main challenges are identified as interest management, event dissemination, cheating mitigation, task sharing, incentive mechanisms and state persistency.

Fan states that: "Game state persistency is a major challenge for P2P MMVEs as existing P2P storage infrastructures are designed to support file sharing, and seldom fulfil the performance and security requirements of a MMVE. . . . the persistency area is still immature with many problems waiting to be investigated." [37].

Fan, however, does not focus on solving state persistency issues of P2P MMVEs. A focus is instead placed on self-organisation in the P2P network, super peer selection, fault tolerance mechanisms, membership management mechanisms, heterogenous task mapping and an accounting scheme for interest management [37]. The Mediator framework is designed and the novelty is reported to be the task mapping infrastructure that allows processing jobs to be advertised on a job-board system

for other peers to perform, thereby sharing the processor load of all peers in the network [38]. A brief overview of this work is presented in Section 1.4.3.4.

1.5.2 Peer-to-peer Second Life

Recent work that is most similar to ours is the dissertation by Varvello entitled: "A Peer-to-Peer Architecture for Networked Virtual Environments" [115]. They develop P2P Second Life by altering the Second Life implementation to run over an existing P2P file sharing network, called KAD [103], which runs on the Kademlia P2P routing overlay [75]. Their focus is thus on a practical implementation, taking an already implemented MMVE and running that as a P2P system over an already implemented P2P network. They also design and simulate a responsive distributed storage system for P2P MMVEs, called Walkad [114]. Walkad is deployed to a cluster and ModelInet [111] is used to emulate wide-area latencies in the cluster underlay. A synthetic Internet topology is generated to model the wide-area latencies, using Inet [122].

There are some similarities between Pithos and P2P Second Life. Both provide the design and implementation of a storage system for P2P MMVEs, although the designs are different as will be explained shortly. Some of the requirements for P2P MMVE state management and persistency were also identified by P2P Second Life, these are: scalability, security and responsiveness. Additionally we identify fairness and reliability as key to designing the P2P MMVEs of the future. We argue for the various identified requirements from the perspective of the generic consistency model that we design. Both our systems also make a distinction between local and non-local storage requests, since it is recognised that users have a finite Area of Interest (AoI).

The KAD P2P network is, however, a file sharing network where user profiles, such as session time, do not necessarily model user lifetimes in P2P MMVEs. It is also not possible to control the parameters of the peers in the network, such as churn. The Walkad simulation is also on a relatively small scale, running on 1024 peers with less than 600 objects in the virtual environment.

Our design philosophy was to design a P2P storage architecture from the ground up, for the P2P MMVEs of the future, supporting thousands of peers and able to store hundreds of thousands of objects. In order to evaluate our design, we elected to implement it in the form of a large scale network simulation based on the Omnet++ network simulator. This allows for thousands of nodes and hundreds of thousands of objects to be simulated, with realistic latency characteristics in the underlying network based on real-world Internet measurements. In our simulation, most results are shown for a smaller network of 2500 nodes, but in order to show the scalability

of our system we also simulate it for 10,000 nodes and millions of objects.

The design philosophy of our storage system compared to that by Varvello is also different. They use a cell-based storage system, where each cell has multiple coordinators. Each cell coordinator replicates all the objects of the cell it coordinates. Our storage system is a group based system, where users are dynamically grouped and objects are stored in groups, rather than cells. Our system also attempts to distribute the load to all peers in the group, by having all group peers store objects. In P2P Second Life, only cell coordinators store objects in the cell. Avatars in the cells do not store objects for that cell.

In terms of performance, there is also some difference between Pithos and P2P Second Life. The responsiveness of Walkad, the improved storage systems developed for P2P Second Life, varies from $O(1)$ in the best case, for local queries to $O(N_c)$, also for local queries, where N_c is the number of cells in the virtual world. The actual performance depends on how equally all regions of the virtual environment are partitioned. If all environments are partitioned exactly the same number of times, requests to neighbouring cells have $O(1)$ responsiveness. Our storage system has a fixed performance of $O(1)$ for local (group) queries. For non-local queries, both our storage systems have order $O(\log(N))$ performance, but again using different implementations.

For our design, the bandwidth requirements of the different methods are also evaluated, since bandwidth usage of P2P systems is of great importance as discussed in Section 1.4.3.1. Bandwidth usage is not something that is evaluated in P2P Second Life.

Pithos is also evaluated from a security perspective. Some designs that improve security are presented and evaluated for various percentages of malicious users in the network. In the work on P2P Second Life, security is stated as being of high importance, but the system is not designed for security or evaluated in an environment containing malicious peers.

1.5.3 Time prisoners

Another work with some similarities to ours is "Time Prisoners" MMOG developed by Merabti and El Rhalibi in 2004 [76]. Merabti and El Rhalibi propose a hybrid client/server P2P MMOG model, where the network topology starts out as C/S, but as peers join they can form groups in regions. In essence, the virtual world is segmented into regions of fully connected peers. Apart from these fully connected regions, Merabti and El Rhalibi also propose the use of Freenet for data storage [28].

There are some similarities in the design of Time Prisoners and Pithos. Both designs use hybrid storage techniques and both designs make use of fully connected

peer groups.

There is, however, a different focus between the two works and the two designs themselves are also different. The focus of the work by Merabti and El Rhalibi was more on the design of users agents and how they interact with the virtual world and each other. Agent communication is described in detail as are agent state migrations. After the network level design of Time Prisoners is completed, a high level discussion is presented on the topics of: scalability, network efficiency, data storage and policing.

No detail design is provided on how state is stored at super peers and on Freenet. No evaluation is performed on the responsiveness, reliability, fairness, security and scalability of Time Prisoners with respect to the storage of game state. No discussion exists on the consistency architecture used for the implementation of Time Prisoners.

In contract, the focus of this work is almost exclusively on the storage of state in virtual environments. While a hybrid design is used that contains overlay storage, the group storage used in Pithos varies greatly from that used in Time Prisoners. Although Time Prisoners make use of fully connected groups of peers that are contained in regions, this fully connected network is not used to store state. All state is instead stored on the regional servers (regional super peers). Pithos in contrast distributes state across all peers in a group to improve fairness, reliability, security and responsiveness. The class of hybrid storage design that the work by Merabti and El Rhalibi form part of is evaluated in detail in Section 3.4.3.

Apart from the Pithos design being different from that of the state storage design by Merabti and El Rhalibi, Pithos is also implemented in simulation and evaluated in terms of the identified performance measured for more than 10,000 nodes and millions of objects stored and retrieved. Although the Time Prisoners design is implemented as part of a real-world MMOG, no evaluated of the storage performance is performed.

1.6 Contributions

- A generic state consistency model is developed that encompasses both C/S and P2P consistency models. This is presented in Chapter 2.
- A review of various state management and state persistency systems is performed in Chapter 3.
  - The state management survey identifies some key metrics by which state management systems may be compared, namely: fairness, overhead, reliability, responsiveness, scalability and security.

  - Storage types described in the literature are found to be classifiable as super peer storage, overlay storage, hybrid region-based storage and distance-based storage.
  - A journal survey paper has been published on this work, appearing in the IEEE Transactions on Parallel and Distributed Systems [51].

- A novel state management and persistency architecture, called Pithos, is proposed and then implemented in simulation. Pithos increases reliability and responsiveness compared to classic P2P MMVE state persistency methods.
  - Distance-based storage is adopted on a group level, instead of a peer level.
  - Two types of storage implementations (safe and fast) are evaluated, with fast storage found to be more applicable to P2P MMVEs.
  - Three types of retrieval (fast, parallel and safe) are evaluated and are all found to be useful in different situations.
  - Two types of repair are evaluated (periodic and leaving), with leaving repair found to be more adaptable to changing network conditions.
  - Pithos is evaluated for more than 10,000 peers, which is the largest scale simulation of its kind that we are aware of.
  - The design, implementation and initial evaluation of Pithos has been published at the International Workshop on Massively Multiuser Virtual Environments (MMVE) at the IEEE International Symposium on Audio-Visual Environments and Games (HAVE) [50].

- A statistical model, using a Markov chain, is developed to predict expected object lifetimes and is used to verify the Pithos results. The model can be used to design distributed storage systems for required mean object lifetimes.

1.7 Summary of this work

In order to design a state management and persistency architecture for P2P MMVEs, the environment in which such an architecture will function first has to be understood. This environment was found to be the consistency model of the P2P MMVE. In Chapter 2, a study is undertaken to determine all the required parts of a consistency model. From studying the literature on classic consistency architectures, as well as the literature on modern P2P MMVE consistency architectures, a generic consistency architecture was developed. Both P2P MMVE state consistency or C/S state consistency are all specialised cases of the generic model.

All modules that form part of the generic consistency model are described, and a summary of research into the various fields are presented. During the investigation of the consistency model, state management and state persistency are found to form part of state consistency in the authoritative storage module.

Knowing that state management and state persistency will be required by the consistency model, the other modules that interface with the authoritative storage module are investigated. These are found to be the environment logic module and object update module.

Chapter 3 explores the various state management and persistency schemes described in the literature. It first defines key requirements that state management and persistency architectures should possess, based on what was learned in Chapter 2. Key requirements are found to be: fairness, overhead, reliability, responsiveness, scalability and security. The storage types reviewed were found to be broadly characterisable into: super peer storage, overlay storage, hybrid region-based storage and distance-based storage. A discussion of each storage type is presented in terms of the requirements identified earlier.

It is found that none of the storage types currently available in the literature satisfy all the identified requirements. Chapter 4 presents the design of a novel state management and persistency architecture, called "Pithos", to satisfy all identified requirements. Pithos is designed as a hybrid storage, containing slower, but scalable overlay storage and faster but non-scalable group storage.

The design firstly describes which modules and mechanisms are used to satisfy the P2P MMVE state management and persistency requirements identified in Chapter 3. Chapter 4 also presents the Pithos design from a use case perspective, showing how the other modules, identified in Chapter 2, are allowed to interact with Pithos. The design of each module and mechanism is described on a conceptual level, without referring to any specific implementation environment.

Chapter 5 describes how Pithos is implemented in the simulation environment, called Oversim. The chapter first provides a motivation for selecting simulation and Oversim as the implementation environment. It also describes how Oversim is expanded to allow for the complete generic state consistency model developed in Chapter 2 to be integrated into it. Pithos nodes are then described in terms of their Oversim module implementations. The chapter also presents some of the major implementation specific issues encountered during the implementation of Pithos, along with the chosen solutions. Specific attention is paid on how group consistency was ensured. The implementation of the application driving Pithos, called PithosTestApp, is also presented.

Chapter 6 provides an evaluation of Pithos in terms of the requirements identified

in Chapter 3. Pithos is found to have high reliability, while having a responsiveness of 1.6 hops in the best case. Responsiveness is found to heavily depend on the underlying network architecture, as well as the probability of inter-group, compared to intra-group requests.

The bandwidth requirements of Pithos are also presented and found to be in the order of tens of kilo bits per second, but depending on the size of the object stored in the system. It is found that Pithos generates 88% overhead, when compared to the data received and sent to the higher layer. Data is defined as only the data of a stored object, ignoring any packet header information received and sent from and to the higher layer and ignoring any responses not containing object data sent and received to and from the higher layer.

Of the overhead generated by Pithos, it is found that the group storage section of Pithos requires only 14% overhead, but that the overlay storage section, which is based on a third-party implementation, requires 86% overhead.

The fairness of Pithos is evaluated and found to compare well with that of overlay storage.

The Pithos implementation is evaluated for various percentages of malicious peers present in the network, that modify all objects requested from them. The quorum mechanism implemented in Pithos is shown to improve retrieval reliability for relatively large percentages of malicious nodes present in the network.

Finally, the results of the object repair mechanism, by which Pithos ensures long-term object survival is shown. It is found that object survival, which is of high importance when designing a distributed storage system, depends heavily on the repair rate, number of initial replicas and expected object lifetime.

Chapter 7 extends an existing statistical model, to improve the accuracy of object lifetime predictions in networks, such as Pithos, where group size is limited. This is done using an expanded Markov chain model, which allows for the theoretical predictions of object lifetimes. The theoretical predictions are verified using the Pithos implementations and found to closely match.

Chapter 8 summarises the main results found in this work. It also presents a discussion of future work that might be undertaken in each of the areas where this work provided contributions.

Chapter 2

State Consistency in Virtual Environments

In order to design a state management and persistency architecture for P2P MMVEs, the requirements of such an architecture should first be understood. State management and persistency form part of the state consistency architecture of an MMVE. The requirements of the state management and persistency architecture, therefore, depends on what the state consistency architecture requires of it.

The role of this chapter is to describe the state consistency architecture in which the state management and persistency architecture will exist. The chapter provides a broad overview of consistency architecture and also introduces a novel generic state consistency model that may be applied to both C/S and P2P network architectures to achieve state consistency in an MMVE. Two basic state consistency architectures are also presented along with a discussion of how they fit into the generic architecture and their various advantages and disadvantages. Finally, the chapter presents an overview of the prevalent consistency model for P2P MMVE and discusses implementations of the various sections as they relate to the generic consistency model.

2.1 The virtual environment

A virtual environment (VE) can be characterised by its *environment state*. When discussing how to segment environment state, it is sometimes easier to speak in terms of *objects*, since they are separable. For the purposes of this work, objects are entities with both state and logic, which means they consume both storage space, as well as computing resources. Objects can also produce events, which should be sent to other objects. When this definition is used, NPC objects may be classified as a specific type of object, which forms part of the global environment state.

Both mutable and immutable objects exist. Immutable objects are those that cannot be altered, for example: rocks, hills and trees. Mutable objects are those that can be altered, for example: NPCs, chests and doors. What objects are mutable is a design decision made by the VE designer, depending on the VE requirements. The environment state includes all positions, health and other attributes of all avatars, NPCs and other objects in the VE.

Mutable objects may further be divided into ephemeral and permanent objects. Ephemeral objects need only be available for certain times, such as a player's (user agent's) *avatar*, when that avatar is online. Permanent objects should remain in the virtual world, whether the user who owns it is on-line or off-line, such as a virtual house.

Agents are defined to be some form of intelligence that can interact with the virtual environment and generate actions. An *avatar* is an agent's representation in the virtual environment. It is the form the agent assumes while in the virtual world.

User agents are users that connect to a VE and control their avatar through some external input/output interface and non-user agents are AI scripts that control an NPC avatar. It is assumed that a single user agent is tied to a specific node in the network. Non-user agents can be hosted on any node and the hosting is an implementation of distributed computation, discussed in Section 1.4.2.1. Both user and non-user agents may interact with the virtual world. When no distinction is required, the generic term "agent" is used.

On a physical level, each computer that executes a virtual environment (VE) is executing a copy of the virtual environment. This is because a computer can only operate on what is stored in its primary memory. Because computers cannot access each other's primary memory banks, the necessity of multiple object copies is created. The existence of multiple object copies necessitates an object consistency model.

In MMVEs, each computer uses the information of the virtual environment, stored in memory, to display the virtual environment to the user. The user makes decisions based on what is displayed and the executed operations on the environment. These operations can affect change in the environment and the change has to be communicated to all other entities capable of interacting with the environment. The fact that users operate on a local version of the environment and that the environment should appear the same to all users requires state consistency mechanisms.

2.2 Environment change scenarios

In a VE, three main actions should be possible concerning the environment state. It should be possible to add new objects to the virtual environment, an agent should be able to discover objects in its AoI and an agent should be able to alter mutable objects in the virtual environment through its actions.

2.2.1 Object creation flow

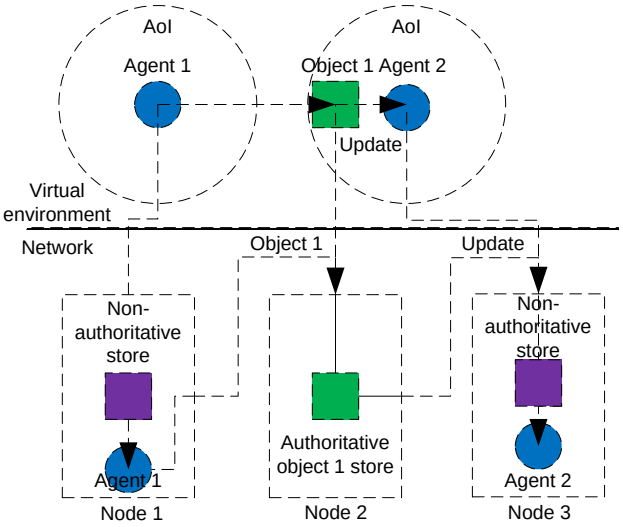


Figure 2.1: Flow of messages in the network when an agent creates an object in the virtual world.

Figure 2.1 illustrates the process of an agent creating a new object and that object being stored on some authoritative node. The figure shows both the virtual environment and the network. Agent 1 creates an object in the virtual environment. For that object to remain in the virtual environment, a node has to be selected to store the authoritative version of the object. In the network, creating an object translates to the node that houses Agent 1, to create Object 1, and select a node (Node 2) to store the authoritative version of Object 1. Once Object 1 has been created in the virtual environment, all other agents that can perceive the new object should be informed of its existence.

For an agent to perceive an object, the object should be within its AoI. When Node 2 therefore stores the authoritative object, it should discover which nodes can perceive the object and then send updates to those nodes informing them of the new object. In the example in Figure 2.1, Agent 2 can perceive Object 1, which means Node 2 must inform Node 3 of the existence of Object 1, using an object update.

Node 3 then receives the object update and adds a non-authoritative version of the new object to its non-authoritative object store. The display of Agent 2 should then be updated to display the new object.

2.2.2 Range query flow

As an agent moves through the virtual environment, it should discover new objects that enter into its moving AoI.

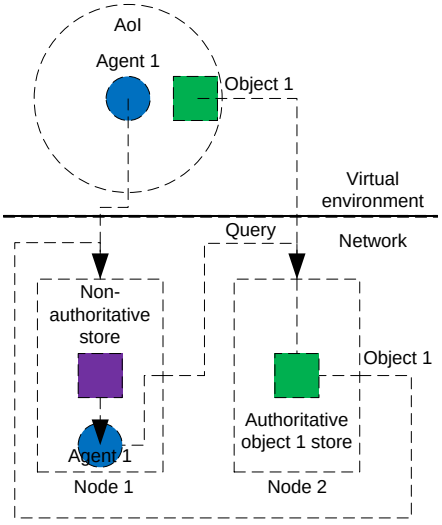


Figure 2.2: Flow of messages in the network when an agent queries its AoI for objects in the virtual world.

Figure 2.2 illustrates the process of an agent discovering new objects in its AoI. As Agent 1 moves through the virtual environment, it has to query nodes that house authoritative versions of objects in its AoI. Node 1, containing Agent 1, has to determine which nodes contain objects that exist within Agent 1's AoI. For the example shown, Node 1 knows that Node 2 contains an object that is located within the AoI of Agent 1. Node 1 sends a query, also called a range query, to Node 2 informing Node 2 of Agent 1's AoI. Node 2 checks which objects it houses that exist within Agent 1's AoI. Node 2 then informs Node 1 of Object 1, which is the only object in Agent 1's AoI.

The agent performing the queries might have to query multiple nodes and the queried node may return multiple objects.

2.2.3 Event-update flow

For the state consistency framework presented, it is assumed that changes in the virtual environment follow the event-update pattern. This means that agents are

not allowed to alter the state of an object in the virtual environment directly, since this leaves the VE open to abuse.

2.2.3.1 Terminology

Events are generated by agents and can be thought of as actions taken by those agents, where agents can be humans or artificial intelligence (AI) scripts. These include casting a spell, using an item or walking.

Environment logic is applied to events to determine what updates should be applied to the environment state. Environment logic is thus a "think" function, which determines how the environment should change as a result of an event. Another way to think about environment logic is to see it as the environment rules.

An agent casting a spell might cause another agent's health to be reduced, her own health to be increased or a monster to spawn. When an agent is walking, the logic will cause the agent's position to update at the agent's walking speed.

Environment logic communicates how the world should change via *state updates*. State updates are the incremental changes that specify how the environment state should change.

An object exists in two forms: the *authoritative object* and the *non-authoritative object*. The state of the authoritative object is considered to be the true state or absolute state. Copies of the authoritative object can be made, but these copies are all non-authoritative objects. If a discrepancy occurs between an authoritative object and a non-authoritative object, the state of the authoritative object is considered to be the true state. The work of the consistency mechanism in the VE is to ensure that all non-authoritative object states follows the authoritative object state within a time frame that is reasonable for the particular VE and the particular type of object.

Authoritative objects are the objects traditionally housed on the server in the C/S based MMVEs. All clients obtain replicas of these objects (termed the non-authoritative versions) and duplicate them locally in order to perform low latency computations. An example would be an NPC monster. When agents perceive the monster in the virtual world, a duplicate of the NPC objects is sent to the agent's computer for display purposes. When another agent attacks the NPC, the change in health will be computed at the server and an update is then sent in order to ensure consistency between authoritative and non-authoritative objects.

It should be noted that a peer does not have to request object state every time it wishes to render it. A peer will render the object according to the local state of the object. It might be possible that a local object's state has not yet been updated with a state update that is en route. To maintain interactivity in the virtual environment,

all operations should occur on the current state of local objects. If it is found that an operation occurred on an out-of-date object and the change is not consistent, the change to the old object has to be rolled back, since the authoritative object's state is always accepted.

2.2.3.2 Motivation

It is impossible for multiple object states to always be consistent, because of the finite time taken to disseminate an object update that has occurred. A well designed object consistency model should not allow conflicting changes to local copies of object states.

An example of conflicting change is one avatar in a virtual game world dealing five damage to another avatar with five health. When the damage is dealt, the target avatar is killed. If, however, the avatar drank a potion that gives it some health, the damage dealt by the first player would not kill him. The state of the player being dead and the player being alive is not possible at the same time.

To overcome the difficulty of conflicting states, it was found that nodes participating in a VE should not be able to update the environment state directly. Nodes should rather express the effect they would like to have on the virtual world in terms of actions (events) and all nodes actions should then be combined by a third, centralised party, which then updates the state of the VE according to the combined causal actions of all nodes in the VE, to ensure that conflicts cannot occur. A disconnect is, thereby, created between the actions that can be performed on objects and the effect that the actions can have on objects. This is called the event-update cycle, also called the "request-process-update-display sequence" [57].

In P2P MMVEs there still exists a need for an authoritative object, but no centralised location exists to store that object. For each authoritative object, a node is selected to store that object and acts as arbitrator for events operating on that object. That node is termed the authoritative node for the specific authoritative object it houses.

It should be noted that in this work, a node refers to a computer that is connected to a network, and a peer refers to a computer that is connected to a P2P network. The term "node" can, however, also be used to mean "peer", since practically, there is no difference between a computer connected to a C/S network and one connected to a P2P network, except for the software that it executes. The term peer is used to clarify that what is talked about is specific to a P2P network.

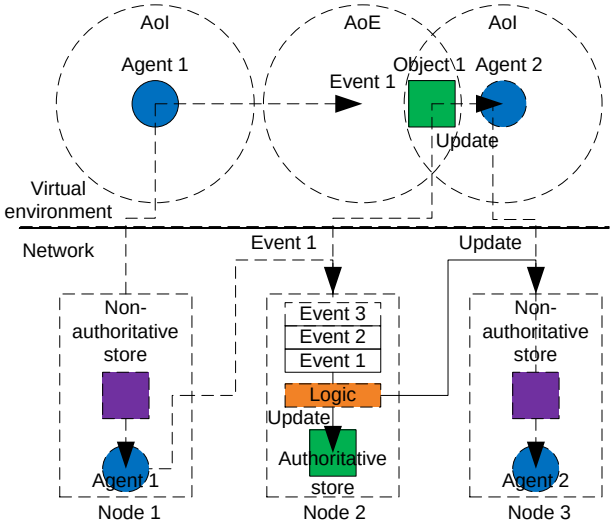


Figure 2.3: Flow of messages in the network when an agent generates an event that updates an object in the virtual world.

2.2.3.3 Description

Figure 2.3 depicts the flow of events and updates in response to the generation of an event by some agent.

In Figure 2.3, Agent 1 generates Event 1 at the remote location shown in the virtual environment. To maintain generality, events are defined to possess areas of effect (AoEs). An event's AoE determines which objects are in range to be potentially affected by an event. When an event's AoE encloses an object, that object is in range to be affected by the event. Whether the object is affected by the event, will be determined by the environment logic housed on the authoritative node.

In Figure 2.3, Event 1 affects Object 1. Node 1, housing Agent 1, therefore has to determine which objects are affected by an event that it generates. It has to determine on which nodes those objects are housed and send the event to those nodes. In this example, only Object 1 is affected by Event 1. Node 1, therefore, sends Event 1 to Object 1's authoritative node (Node 2).

Determining which objects are affected by an event is termed *event interest management*. Delivering the event to the authoritative nodes of the affected object is termed *event dissemination*. [1]

When Event 1 arrives at Node 2, containing Object 1, the event is placed in

[1]It should be noted that designs exist that use multiple authoritative objects, for example the event-based consistency model described in Section 2.4.1, but that more research focus has been placed on consistency architectures where every object only possesses a single authoritative object, such as the update-based consistency model described in Section 2.4.2.

a event queue in an order based on the time each event occurred in. The process of determining which order events have to be processed in is called *event ordering*, since many events will be arriving at an object's authoritative node from multiple agents simultaneously.

After event ordering has been performed, environment logic is applied to determine how a specific event will affect a specific object. To determine possible updates, the states of other objects might also have to be taken into account. An example is firing at an agent through a wall. If the thickness of the wall determines how much damage is delivered to the agent, the authoritative node processing the damage to the agent object will have to know the thickness of the wall.

When an event has been translated into a state update, the authoritative object on the authoritative node (Node 2) is updated. The new state of the object should also be communicated with all agents that can perceive the object in the virtual world. An object is perceived by an agent if the agent's area of interest (AoI) in the VE encompasses the object. Every agent that can perceive an object, requires a non-authoritative copy of that object. Determining which agents should receive object state updates is termed *update interest management*. In the figure, Object 1 is perceived by Agent 2. Node 2, therefore, sends an update to Node 3, the node containing Agent 2. Delivering state updates to the affected agent nodes is termed *update dissemination*.

When the update is delivered to Agent 2, the local object state is updated and the object's new state is displayed to Agent 2 to allow the agent to make decisions and generate events based on the new object state.

Object removal can be considered as a special case of object modification. An object modification request of type: "remove object" can be sent. An authoritative peer receiving this request can then remove the object. Is should be noted that it is difficult to remove objects from a distributed environment, since a node that contains an object might have left the network. For this purposes, object time-to-live is usually used in practice. This is described in more detail in Section 4.4.4.

2.3 Generic consistency architecture

In this section, we will discuss the literature where individual modules of the generic framework are implemented, but to our knowledge, no article has been published that presents a comprehensive view of a state consistency architecture. This generic architecture allows us to put the literature regarding state consistency into context and allows us to identify requirements of the authoritative object store from a consistency perspective.

2.3.1 Request-response consistency basis

Since the description of an overarching framework for state consistency was lacking in the literature, we decided to develop such a framework. To develop this framework, the main issues that state consistency has to solve for each of the scenarios discussed in Section 2.2, were identified:

- Determine the set of nodes that are interested in a generated request (object, query or event).
- How requests are sent to the set of interested nodes.
- Determining the correct order of received requests (events).
- Generating a response (updates or objects) in response to a request (queries or events).
- Determining the set of nodes that require the generated response.
- How responses are sent to the set of interested nodes.
- How the effect of a response is displayed to an agent.

Each identified consistency scenario is in the same basic request-response form. The items in parentheses show the possible forms that the specified request or response can take. Using this basic model, the generic consistency model was created that incorporated all scenarios. This model is shown in Figure 2.4.

2.3.2 Model

Three "flows" exist, that model each of the query, create and modification scenarios discussed earlier. A flow is initiated by an agent and terminated at an agent. Flows are symmetrical, which means any agent can initiate a flow or be the terminating agent for a flow. Each flow corresponds to the three scenarios discussed in Section 2.2.

2.3.2.1 Agents and generators

Agents generate three streams of requests, using a query, object or event generator. The requests are sent to the authoritative object hosts.

Agents may be able to build structures, make clothes or construct a variety of mechanisms in the virtual world. The objects that the agent creates in this way should be stored on an authoritative node.

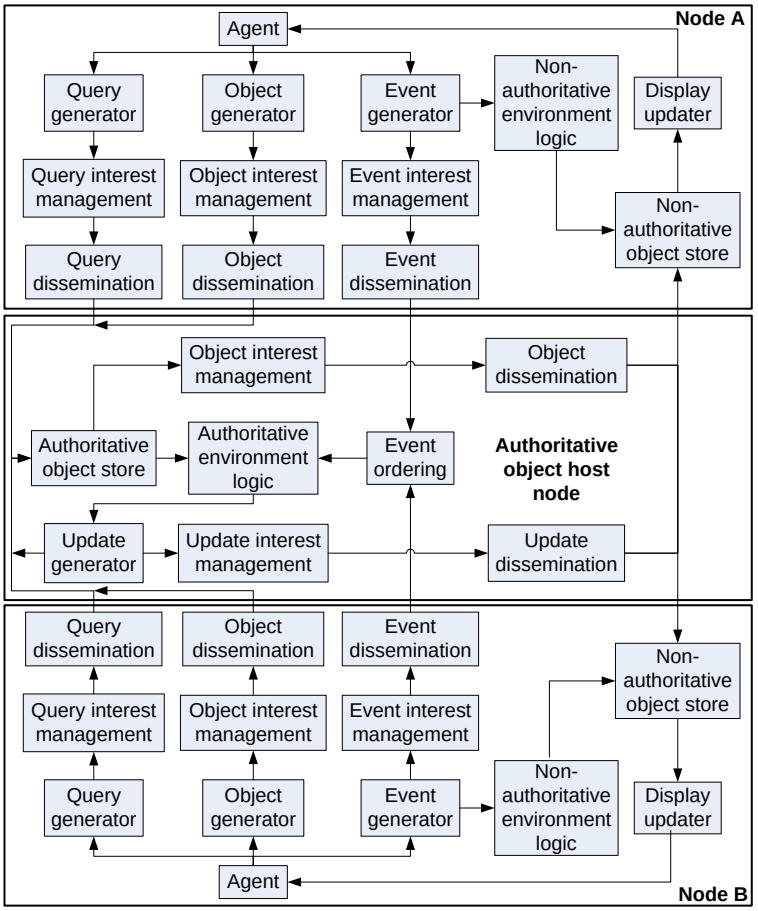


Figure 2.4: Event-update flow cycle and all steps required from generating an event to delivering an update, to ensuring object consistency.

Agents can also interact with the virtual environment and other agents, which are objects themselves. Agents interacting with objects generate events, which have to be sent to the authoritative node.

Lastly, as agents move around in the virtual world, they might query an authoritative node for any new objects that have entered their AoI. This is referred to as a range query and requires the retrieval of all objects within a certain area.

2.3.2.2 Display updater and non-authoritative environmental logic and object store

Some generated events may be of low importance where a malicious creation of the type of event will not give the generating agent any advantage and will not disadvantage other users. For these types of events, processing them locally using non-authoritative environment logic may be sufficient. Locally processing events reduces network traffic, but increases the security risk. Deciding which events will

be handled by non-authoritative environment logic and which will be handled by authoritative environment logic is a significant design decision during the MMVE design phase. It is sometimes difficult to know which types of events malicious users will be able to use to break the environment rules.

Some events may be handled by the non-authoritative environment logic module. This module outputs object updates directly to the non-authoritative object store, without generating any network traffic. The non-authoritative object store houses the local copies of objects that an agent uses to perceive its virtual environment.

The display updater module displays the non-authoritative object store objects to the agent. This may be in the form of interfacing with a graphics card to output the information on a computer screen for a user to see, or to make the objects accessible to an AI script.

2.3.2.3 Interest management and dissemination

Interest management deals with finding a set of nodes that might have an interest in a request being generated. Events, objects and queries require some form of interest management.

Event interest management determines which objects will be affected by a generated event and on which authoritative nodes these objects are housed. Object interest management determines on which authoritative node the generated object should be stored. Query interest management has to discover which authoritative nodes contain objects that exist within the specified range.

Event, object and query dissemination deal with how information is sent to nodes after interest management determines which information should be sent (and to which nodes).

2.3.2.4 Authoritative node

All requests generated at non-authoritative nodes are sent to authoritative nodes for processing. Objects received are stored in the authoritative object store.

Multiple nodes, controlling different areas, may receive the same range query and be expected to supply the objects in their respective authoritative object stores that have positions that intersect with the range of the range query.

After receiving events, they are ordered to maintain causality. Object updates are generated from received events. An object update can be considered as some change that should be applied to a known object. Generating an object update, might also require the state of other objects, retrieved from the authoritative object store. Therefore, after the combination of event, object states and environment logic, an update is generated.

The authoritative node, therefore, can create two reply streams: objects and updates. Objects are created in response to queries and sent to the node requesting the objects. Updates are created in response to received events and these updates have to be applied to the object in the authoritative object store, as well to all non-authoritative copies of the updated object.

Update interest management determines which nodes require the generated update, which typically are the nodes whose agents can perceive the object in the virtual environment, and therefore possess non-authoritative copies of the object. Object interest management has to determine which node received a generated object. With a range query, this will merely be the node that sent the query, but other situation might occur where one node requests objects on behalf of another.

Update dissemination delivers a generated update to all nodes that contain local copies of the object. Multiple object updates may be generated for a single event. Any generated update may be delivered to multiple nodes that contain local copies of the object. Object dissemination delivers generated objects to the nodes that requested them.

2.3.3 The event-logic-update scenario as the generic scenario

It should be noted that object queries and object creation can both be implemented as specialised forms of the event-update model. An object query can be implemented as a "query" event with an AoE equal to the generating agent's AoI. The environment logic determines which objects are within the event's AoE and state updates containing the complete object can then be sent to the node containing the originating agent.

Object creation can also be handled as an event, where an "object creation" event contains the information required to instantiate the required object. The object creation event can then be sent to the applicable authoritative node. The environment logic will process the object creation event, generate a state update containing the new object and respond with that update to the node of the agent generating the object. This is also considered the safer method by which an object can be generated, since the environment logic can ensure that the object is allowed within the confines of the environment rules.

It is therefore sufficient for the generic state consistency model to only contain the event-logic-update flow, but the other two flows were added for clarity. For the range query, it clarifies that the response is always sent to the node generating the query. For the object creation case, it clarifies that the object itself is generated on the non-authoritative node and then sent to the authoritative node, where it is then housed.

2.3.4 Discussion

The flow diagram in Figure 2.4 shows all steps that should be taken to ensure object consistency. We shall now demonstrate that the flow diagram of Figure 2.4 is applicable to all known state consistency models of C/S as well as P2P MMVE architectures.

It should be noted that the generic consistency model is developed to fit all consistency architectures. To this end, some modules might be trivial in some consistency architectures, but become non-trivial with others. This is especially true for the C/S consistency model, where only a single server or server cluster exists, as will be described in Section 2.4.

The differences between the various architectures is where the authoritative objects are housed, and how the authoritative nodes are distributed. In a C/S system, all authoritative objects are housed in the server. In a P2P architecture, authoritative objects can be distributed on a number of nodes, depending on the specific architecture. The various state consistency schemes for both C/S and P2P networks will be reviewed in the next sections.

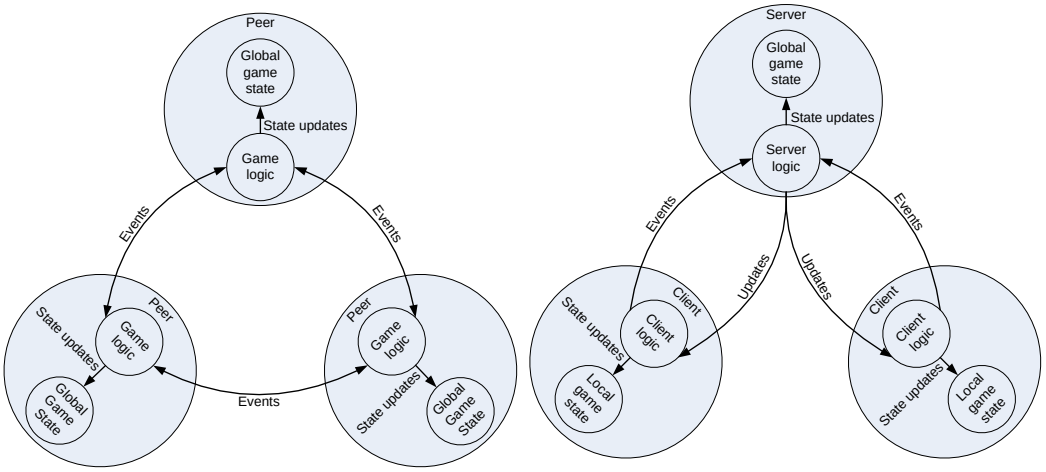
2.4 Basic consistency models

An overview of two basic models, currently used in MMVEs will be described. More complex models, for example ones that contain partitioning, exist and will be described in Section 2.5. The models used in P2P MMVEs are all permutations of the two basic models. The two models are based on the two different network models. These are the fully distributed model, also called the event-based model [11], and the C/S-based model, also called update-based model [108].

2.4.1 Event-based (Fully distributed)

Figure 2.5a shows the fully distributed model. Each node has a copy of the global VE state. Any event that a node generates is sent to all other nodes. Each node processes all events separately and updates its own environment state accordingly. This consistency model is one where there exists multiple authoritative environment states.

Figure 2.6 shows the event-based model in terms of the modules defined for the generic consistency model. The figure shows two nodes and how they may interact. For the event-based model, each node is both an authoritative and a non-authoritative node, for the purposes of which modules they contain. There also exists no non-authoritative storage, object interest management, object dissemination,



(a) Event-based (fully distributed) (b) Update-based (client/server)

Figure 2.5: Consistency models

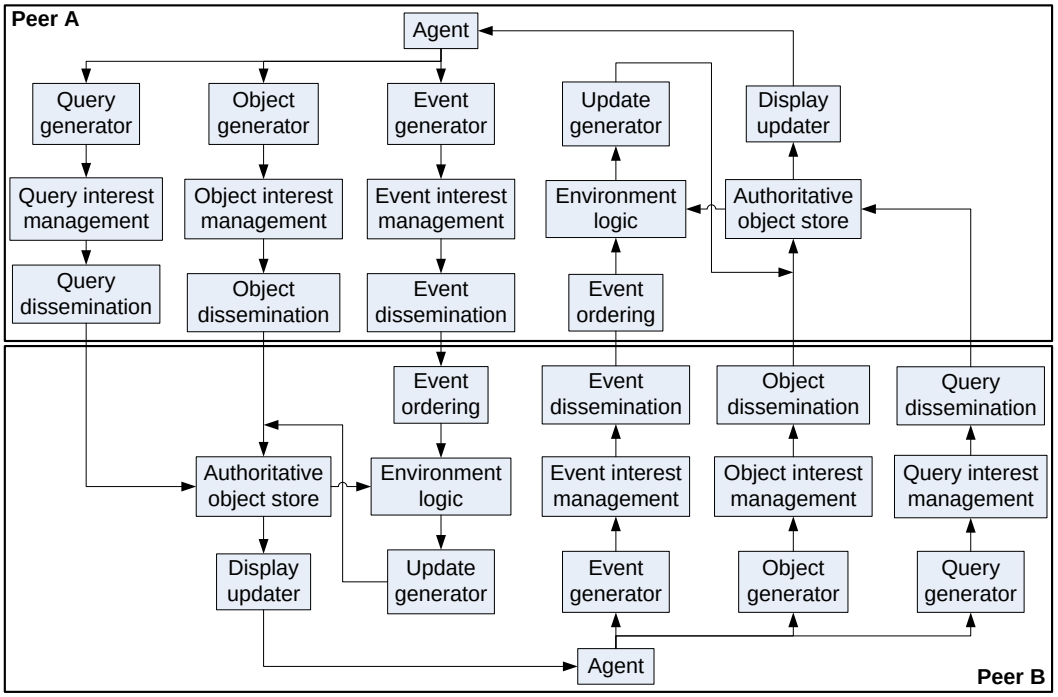


Figure 2.6: Event-based consistency model, showing the modules as defined for the generic consistency model. The diagram shows how the generic consistency model may be used to describe the event-based consistency model.

update interest management, or update dissemination modules. The authoritative storage module is directly connected to the display updater module.

The event-based model has fallen into disuse over the past few years, because of its lack of scalability and the degradation in responsiveness experienced when high latency nodes are connected due to the event ordering techniques used, as explained in Section 2.4.1.2.

2.4.1.1 Event generation and dissemination

With reference to the generic flow diagram in Figure 2.4, user agents access the VE using nodes in the fully connected network. Event generation occurs on nodes.

Event layer interest management is trivial, since all nodes receive all updates. Every node, therefore, requires an up-to-date list of all other nodes in the network. Event dissemination typically makes use of unicast to send all events to all nodes.

2.4.1.2 Event ordering

The order in which events are received is the same for all nodes, otherwise the environment states of different nodes may become inconsistent. Usually some kind of lockstep technique is used to solve this issue [5]. The issue with lockstep is that it reduces the latency to twice that of the node with the highest latency.

Various techniques have been proposed that improves the latency by introducing some deadline before which all events should be submitted. An example of this is the New Event Ordering (NEO) technique [47]. Event ordering using deadlines makes it difficult for a user with a high latency to access an MMVE, because if one user has a higher latency than the others and misses event deadlines, none of her actions will be processed by the other nodes which effectively excludes the user from the network.

An alternative to NEO is a P2P and C/S hybrid scheme, called the referee anti cheat scheme (RACS) [121]. The scheme proposes referees, that are similar to the C/S network model, which allow for greater resistance to more types of cheats.

2.4.1.3 Environment logic

Complete environment logic is housed on every node, since every node must be able to apply the logic to translate any event into a state update. The environment logic will determine a set of objects that will be affected by the received event and update the affected objects accordingly.

2.4.1.4 Update generation and dissemination

The environment logic housed on every node generates state updates. These state updates are only used to update the local copies of affected objects, since each node is responsible for keeping its environment state up-to-date.

Update layer interest management and update dissemination is not required, since all affected nodes will have received the event.

2.4.1.5 Authoritative object store

The complete environment state is stored on each node. This is why the event ordering technique employed is important. If an event is received in the incorrect order at one of the nodes, it may cause inconsistency for the authoritative state of that node. Such a situation is difficult to correct. A possible solution is for detection mechanisms to be implemented, where peers exchange object states at regular intervals for comparison. If a peer's state has diverged from the group, a quorum mechanism might be employed to correct the divergence.

2.4.1.6 Advantages

The event-based model works well for strategy games, and was implemented in Age of Empires [11] and Starcraft [32].

When latency issues are not present and all users possess reasonable latencies, the event-based model can provide for an high-degree of responsiveness, because no extra latency is added by a server and no extra hop required to send information to a server which then relays it to all clients.

2.4.1.7 Disadvantages

The event-based model is not scalable, since all nodes must connect to all other nodes and all events are transmitted to everyone. This means that as N, the number of nodes in the network increases, the traffic increases with a factor of N^2. The security issues of the P2P network model, on which this consistency model is based, are also present. A degradation of responsiveness is also experienced by all users if one user's latency is below par, since the lockstep mechanism has to wait for all events to be received for that round to conclude [47].

2.4.1.8 Partially connected event-based consistency

An event-based model that is not fully distributed might be considered, although such a model holds many challenges. Interest management techniques might be used to determine which nodes require updates. A challenge arises when nodes that were

not aware of each other come into contact in the virtual world. A merge operation will have to be performed on the two authoritative environment states to ensure consistency. Such a model has not been proposed in practice and significant research will have to be performed into defining a framework that establishes relationships between events to maintain consistency. Such a system might be developed by defining classes of events and their interactions. A linear temporal logic model might then be developed to ensure consistent merging of states.

A partially connected model is not a basic consistency model.

2.4.2 Update-based (C/S)

An alternative to the event-based model is the update-based model, shown in Figure 2.5b. This model is based on the C/S network model and based on the principle that only the server contains authoritative environment state.

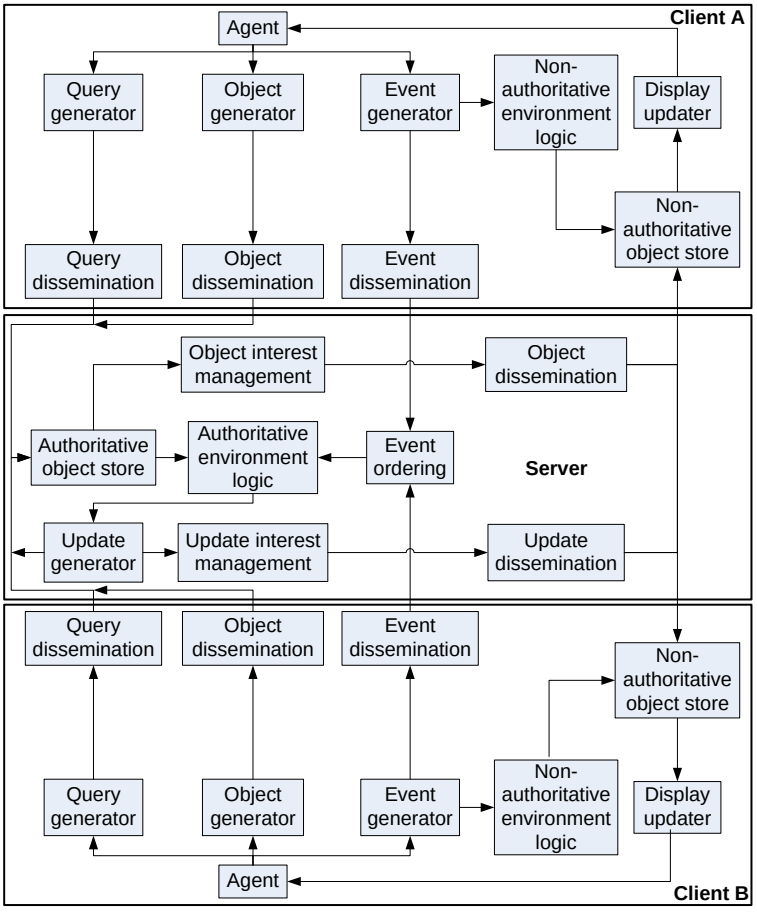


Figure 2.7: Update-based consistency model, showing the modules as defined for the generic consistency model. The diagram shows how the generic consistency model may be used to describe the update-based consistency model.

Figure 2.7 shows the update-based consistency model, showing the modules as defined for the generic consistency model. This model is similar to the generic consistency model itself, introduced in Figure 2.4. The only difference is that there exists a single authoritative node, containing all objects and that no interest management is required by the clients, since all queries, objects and events are only sent to the server.

2.4.2.1 Event generation and dissemination

Agents control clients that are able to generate events. As with basic event-based consistency, event layer interest management is trivial, since all events are sent to the server, usually making use of a unicast event dissemination technique. Interest management only becomes non-trivial when the P2P MMVE consistency model is considered in Section 2.5. The reasons why these sections exist in a discussion of the basic consistency models is to show that, although trivial, all sections relate to the generic consistency model.

2.4.2.2 Event ordering

Event ordering is performed server-side, although it is not as critical as for the fully distributed model. If events are processed in different orders by different clients in the fully distributed model, the environment states of the clients will start to diverge and no mechanism currently exists to merge the diverging states.

If event ordering is not performed in the update-based model, the server might execute some events out of the order in which they were generated. The environment state will, however, remain consistent, since the server informs all clients of the environment state once it calculates a new environment state.

2.4.2.3 Environment logic

The environment logic that deals with high risk and high importance events is housed on the server. This allows for simplicity of design and increases system security, since the server can verify any actions performed by a client. Environment logic housed on the clients deal with low risk events that don't require a high level on consistency, for example, movement updates.

2.4.2.4 Update generation and dissemination

The environment logic generates states updates, which is used to update the server's authoritative environment state and is also used to inform nodes able to perceive the updated objects of the object's new state.

Update level interest management is required to determine which clients can perceive the updated objects. Unicast (direct transmission using TCP or UDP) is then used as update dissemination technique to inform the affected clients of the updated object states.

2.4.2.5 Authoritative object store

An authoritative global environment state is housed on the server and a non-authoritative local environment state is housed on all clients for display purposes.

2.4.2.6 Advantages

- *Authority*: The server state is authoritative, because if there is a conflict, the server state is always the state to which the system is expected to return. Combining server authority with the fact that the MMVE operator has full control over the server allows the operator to control who may access the server and on what terms.
- *Security*: The update-based model is secure, since clients cannot influence the state of any other clients and every client's state depends on updates received from the server.
- *Hardware scalability*: Another reason why the update based model is successful is because it is currently more scalable then the fully distributed model. More hardware can be used to build a more powerful server, which can handle more clients. It should be noted that scaling using hardware is expensive, since large server farms are required to host the VE.

2.4.2.7 Disadvantages

All disadvantages of the C/S architecture, as discussed in Section 1.2.3.2, are relevant to the update-based model, since the model is based on the C/S network model.

- The main disadvantage is the high cost involved in obtaining and maintaining the computer clusters and server hardware, required to host the VE.
- When the server fails, the complete VE is off-line. Approaches exist to host the VE on multiple servers, but in this case a "zone crash" may still occur where a server hosting a specific region goes off-line. Additional redundancy is possible if backup servers are used, but this further increases costs.

- Sufficient hardware may be made available to support the VE environment, but the update-based system struggles with dynamic load balancing to cope with large numbers of users flocking to a single area in a short span of time [25].

2.5 P2P MMVE state consistency models

Various schemes have been proposed to address the various challenges of P2P MMVE state consistency, but these have mostly occurred in isolation. Each approach focuses on a few of the modules identified in the generic state consistency model presented in Section 2.3, but no approach addresses all modules.

What should be noted is that the various areas of state consistency, e.g. interest management, event ordering, state management, and the other areas discussed in Section 2.3, are quite modular, which might allow for the integration of the best schemes in each area into a comprehensive P2P MMVE state consistency model. Therefore, some of the proposed schemes were reviewed to determine the scheme most suitable for integration.

2.5.1 Region-based update-based consistency model

P2P MMVE schemes are based on one of the two basic consistency models described in Section 2.4. Although it might seem that the fully distributed model is a better fit for P2P MMVE state consistency, this model has received less attention than the C/S model. The reason for this is partly because of the difficulties in getting the fully distributed model to scale. Usually the fully distributed model makes use of rounds, where users submit events and all user events are executed at the end of a round. In an MMVE with tens of thousands to millions of users, designing the round mechanism to remain scalable is a challenge. Rounds are required, because of the event ordering mechanism that ensures state consistency.

The C/S consistency architecture is applied to P2P MMVEs by distributing authoritative objects. Every authoritative object is maintained by a single peer, that receives all events for the object, applies the environment logic and informs all peers of the generated updates. This effectively creates a C/S consistency model from the perspective of each authoritative object. An example of such a consistency model is the region-based model, which requires that some peers be selected as super peers. Super peers are specialised peers that perform specialised tasks within the P2P network. [2]

[2]The super peer is selected in some logical way, from the available set of peers and then promoted. Super peer selection in itself is a complex topic that has to deal with determining whether a peer has sufficient resources available and also whether the peer is trustworthy.

A region is usually created by segmenting the world, with super peers acting as regional servers to all peers in their region. Each super peer receives all events, handles all environment logic and distributes updates to all peers in its region. The super peer also handles state management and persistency for its region, containing NPCs, objects and persistent user data. Clients in the region only house copies of the regional objects and some client logic to update the local copies of objects.

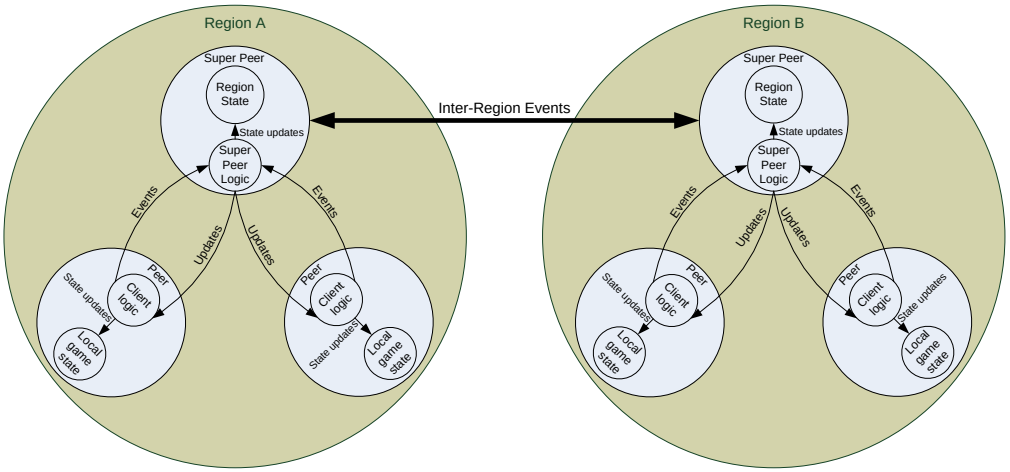


Figure 2.8: Region-based Client/Server consistency model

Figure 2.8 shows the region-based P2P consistency model. The model is based on the update-based model, but segmented into separate regions. The role of the server is here fulfilled by a super peer.

In the remainder of this section an overview is presented of the different techniques and general trends present in the various areas of P2P MMVEs. The overview does not presume to present an exhaustive list of papers in these areas, rather to place the topic of state management and state persistency in context.

2.5.2 Request generation and dissemination

In P2P networks, each peer contains an agent that represents a user (player). Each agent generates events, queries and objects that have to be transmitted to the required peers.

Classically, application level multicast (ALM) and unicast techniques are the two methods of request dissemination (classically called event dissemination). The specific technique used depends on the grain of the dissemination. ALM is used, instead of router level multicast, because of a lack of general support for this technology at the router level [34].

ALM is used for coarsely grained interest management techniques, while unicast is used for finely grained techniques. Unicast is not used for coarsely grained techniques, because it is not scalable. For a network containing N peers, N^2 messages are exchanges for every user action. ALM, however, significantly increases the message latency in the system, because messages first have to be routed over a structured overlay network. ALM is however preferred over unicast for large numbers of messages, because of the weak scalability of unicast.

An ALM scheme has recently been proposed, that sends updates to locations in the virtual world, rather than to specific users [49]. This removes the need to first determine which peers will be affected by a request, before the request may be transmitted. Another ALM scheme proposed sends messages to the neighbours in the AoI, instead of to all the users in the region [98]. This effectively implements a basic interest management scheme, where only neighbouring users can receive requests.

2.5.3 Event ordering

The presence and complexity of event ordering depends on the consistency architecture on which the P2P MMVE is based. In systems that use the update-based (C/S) model, no literature could be found that addresses the issue of event ordering, since events delivered out of order will not lead to an inconsistent state, as discussed in Section 2.4.2.2.

Where distributed architectures are proposed for P2P MMVEs, event ordering is of high importance. Designers of P2P MMVEs make use of the same event ordering mechanisms, as is used in the fully distributed model [45].

To support a large number of users using an event-based model, event ordering with the concept of interest has been developed using N-trees [46].

An active field of research is that of optimistic event ordering techniques, which is used in the field of synchronised parallel discrete event simulators [44]. An example of such a technique is the time warp algorithm [63]. The basis of time warp is that all events are executed optimistically, without regard for the causal order between events. If an event is later received that should have been processed before an event that has already been processed, the system is rolled back in time to process the correct event first.

The time warp algorithm is a complex one, with many possible transient errors and race conditions that can occur. Since a design of the algorithm, a formal verification effort has been working on showing the correctness of the algorithm and identifying issues that should be guarded against [43].

2.5.4 Environment logic

In P2P architectures, the environment logic for all objects have to be housed on all peers, since any peer can be selected to store any object. The environment logic itself is, therefore, usually also distributed as part of the client software. This does not mean that the environment logic cannot change, but any change in environment logic will be in the form of a client patch, which occurs while the application is off-line.

For distributed applications where the code base might have to be dynamically updated, dynamic code deployment might provide a partial solution [70]. Dynamic code deployment works by a deployment system being able to deploy code to a host system. The host system is able to verify the identity of the deployment system and the validity of the code. The deployment system is able to verify whether the code is successfully deployed.

2.5.5 Update generation and dissemination

In P2P MMVEs, update generation is the responsibility the peer containing the authoritative object. When the peer receives an event, environment logic is applied to generate an update. This update is then disseminated to all peers that have local non-authoritative copies of the object. This is similar to the update-based consistency model discussed in Section 2.4.2.

Hybrid C/S and P2P update dissemination schemes that use the concept of area of propagation (AoP) exist, to achieve high levels of responsiveness [106]. If updates that a peer generates only affect peers within its AoP, the peer directly transmits the updates to those peers. If a peer generates updates that affect peers outside of its AoP, it sends the updates to a server, which retransmits the updates to all affected peers. Preliminary results show the hybrid approach remains more scalable, i.t.o numbers of peers, than either P2P or C/S techniques in terms of responsiveness. The work uses the terms event and update interchangeably, so it is not always apparent what is being referred to.

2.5.6 Update and event layer interest management

Interest management occurs both in the update and the event layer. For the event layer, the set of peers housing the authoritative object, affected by the generated event, has to determined. For the update layer, the set of peers has to be found which house the agents that will perceive an object update. What is common to both layers is that a message has to be sent to a subset of peers with interest in the

message and the interest management mechanism must determine what that subset is.

Thus far, little attention has been paid in the P2P MMVE literature, to whether the interest management in question is developed for the event layer or the update layer. Most discussions implicitly assume a layer and all the descriptions of the algorithm are based around the terminology of that layer. That is not to say that the algorithm itself might not be traversable to the other layer, just that this is not discussed.

Interest management is used to determine the smallest amount of information that a peer requires, in order to present an accurate representation of the world to users. In consistency terms, it provides a means to determine which non-authoritative objects require updates of the authoritative object. The idea is not specific to P2P MMVEs and was already formally suggested in [79] and later with greater focus on a distributed environment in [119].

The main idea is that a user has a limited visual range and area around the user in which it can interact with objects. The user requires update information of all objects in this area, called the user's Area of Interest (AoI). AoI calculations also rely on the fact the a user's direction and velocity of movement cannot change instantaneously and are bounded in magnitude.

Extensive research has been done into solving AoI problems and a comparison of techniques can be found in [17] and [67]. The solutions range from aura/nimbus [9] to publish/subscribe [14] to Voronoi based models [57], [18] to hybrid models [85], [124], [38].

Generally, interest management solutions can be divided into coarsely or finely grained solutions.

2.5.6.1 Coarsely grained IM

Coarsely grained solutions usually divide the virtual environment into multiple regions and when a user enters a region, it subscribes to that region's events. This is called the region-based publish subscribe model [39]. All users in the region receive all events generated in the region.

2.5.6.2 Finely grained IM

Finely grained techniques create groups of agents based on their AoIs. Groups of interacting agents directly exchange information, so all agents only receive information that is relevant to them. This has been termed the spatial model [39]. The grain of the solution in turn determines the type of event dissemination that should be used, as described later in this section.

Another example of finely grained interest management, making use of frontier sets, was presented in [101] and implemented in Quake 3. By using Frontier Sets, it is possible to describe an area in which an agent may move, where no other users will require updates of that movement. Conversely, a set of other users that require movement updates from a specific user is also given by the frontier set.

2.5.6.3 Hybrid IM

Hybrid models, especially MOPAR, seem to have gained greater popularity because they seem to have all the benefits of the two solutions and little of the drawbacks [124]. MOPAR has been shown to perform better than either a finely grained technique or a coarsely grained technique.

MOPAR partitions the VE into hexagonal regions and appoints "home" peers to act as bootstrap peers for each region. A home peer of a region is that peer whose ID is the closest match the the region ID. This allows any peer to find the home peer for a region. A master peer is then selected for every region, whose function it is to inform slave peers of new neighbours. All slaves peers in a region register at their region's master peer.

Slave peers send direction and velocity updates to their masters. Masters communicate directly with other masters if a peer is about to enter their region. Masters inform their slaves of a new neighbour. Slaves communicate directly with each other, once identified by a master.

Interest management is a mature area of research, with many proposed schemes. It is one of the areas in P2P MMVE research that has received the most attention.

2.5.7 Authoritative object store

State persistency is treated as a sub domain of state consistency and is similar to state management, as discussed in Section 3.1. State models define where and how the authoritative objects are stored. It is assumed that non-authoritative objects are always stored in the primary memory of the clients that immediately require the information contained in the authoritative object.

The most mature state management model reviewed is that of P2P Second Life, which was discussed in Section 1.5 and will be further discussed in Section 3.4.1.5.

2.6 Conclusion

This chapter introduced the event-update consistency model that is mostly used in MMVE. From the literature reviewed, a generic process was developed that will ensure state consistency. Basic state consistency models were introduced and it was

shown how these models relate to the generic state consistency process. This is followed by a description of state consistency for P2P MMVEs. It was shown how the models used are all based on the basic consistency models. Some research into every area of P2P MMVE state consistency was discussed to enable the reader place the research of this work into perspective.

In this chapter it was shown that the field of P2P MMVE state consistency is a growing one, already containing many significant contributions. The purpose of this chapter was to place contributions in the field of P2P state consistency into perspective, to structure the literature concerning state consistency. It is believed that this structure will allow for more complex state consistency designs, by making use of the modular approach proposed in this chapter.

A thorough investigation into state consistency has now been performed. It was found that significant work still remains in the authoritative object store. Maintaining the authoritative object store is termed state management and state persistency. These techniques will be reviewed in the following chapter. P2P storage is classified into multiple types found in the literature and metrics are identified by which to judge the various storage schemes. This is done to work towards a point where our own novel state management and persistency architecture can be proposed.

Chapter 3

P2P MMVE state management and persistency

While the previous chapter broadly introduced all the concepts of state consistency, this chapter will focus solely on state management and state persistency in P2P MMVEs. State management and persistency in P2P MMVEs respectively allow for the short- and long term storage of environment data. The fact that environment data are now distributed amongst various peers in the network creates challenges not usually present in classic C/S MMVEs.

This chapter classifies the techniques used in documented P2P MMVE architectures and discusses the advantages and disadvantages of the different types of storage. The chapter also shows that no current storage type is well suited to P2P MMVEs.

3.1 A definition of state management and state persistency

A discussion of state storage requires an understanding of the difference between state management and state persistency. These terms are regularly confused in the literature and sometimes one is used where the other is more appropriate. Authors describe both mechanisms as if it is all part of the same model.

Storing authoritative data requires state management and state persistency. State management is the storage of objects in primary storage, while state persistency is the storage of the VE on secondary storage. State persistency does not have such rigorous responsiveness requirements as state management, as explained in Section 3.3.4.

In this work, state persistency is defined to mean long term data storage. This

means that if a server or client goes off-line, the state persistency mechanism will ensure the safe storage of the state for when the server or client starts up again. State persistency is, therefore, storage in secondary persistent memory. All servers require state persistency to back-up the VE state in case of some outage or in case a server restart is required. This storage type does not require low latency, but needs to be reliable. State persistency in P2P MMVEs is therefore taken to mean the same. It does not have to be fast, but when a client logs out of the VE and logs back in again, the stored state should remain.

State management is defined to mean the storage of the VE state in primary memory. This is the state with which the server and clients will interact. It is the state that will be displayed to the agent on the client and it is the state that will require fast retrieval and storage as a result of received events or updates. State management is used to:

1. apply generated or received object updates to authoritative objects on the authoritative node,

2. apply received object updates to non-authoritative objects on non-authoritative nodes, and to

3. supply information of other objects needed by the environment logic, in order to generate authoritative object updates for a received event.

Although state management and state persistency are two different types of storage, with different storage requirements, there are also similar aspects between state management and persistency. Since state management and persistency are similar (as discussed in Section 3.3), the term “storage” will be used when referring to both.

Many “storage types”, which can be used for either state management or persistency are discussed in this chapter. The reference of the storage type to state management and/or persistency will be discussed where applicable.

Before discussing storage techniques that have been developed for P2P networks, a review of classic state management and persistency architectures used in C/S based systems will be reviewed.

3.2 Client/Multi-Server state management and persistency models

This section introduces the classic state management and persistency architectures used in server cluster or client/muli-server based systems [57].

3.2.1 Sharding

A storage model where the VE state is duplicated over multiple servers, with users connecting to one of these servers is termed "sharding". For all practical purposes, this approach is still merely a C/S approach, with users forced into a specific C/S environment. Each shard hosts the global authoritative state and thus state management and persistency is performed centrally by the shard server, where all objects are stored.

Project Darkstar was created as middleware for a shard-based system to provide a networking architecture for MMOGs [16].

3.2.1.1 Advantages

Sharding has the following advantages:

- It allows for greater scalability than the single server approach, since the maximum load is fixed and more shards can be added as the environment becomes popular.
- It is relatively simple to implement, because no inter-server communication is required and no server migration is supported.
- It allows for a relatively small environment to support many users because of the duplication of the worlds. This reduces the load on level designers and content designers, who now have to populate a much smaller world with content.

3.2.1.2 Disadvantages

Sharding has the following disadvantages:

- Clients are not able to interact or communicate with users on other shards, which reduces game immersion.
- Users are not able to enter a shard if that shard has reached its capacity. In the past, this has caused unhappiness amongst users, since popular shards could be difficult to log in to. Users are also reluctant to move to a new shard, because a lot of time is invested in their characters in their "home" shard.
- Inefficient resource allocation occurs, as one shard may be overpopulated while another is underpopulated.

3.2.2 Replication-based

The replication-based model is similar to sharding, with the difference that all servers share the same duplicated environment state. Each server contains the global environment state and clients connect to any one of these servers (mirror-servers [30]) or through a load distribution algorithm to a server (proxy-servers [80]). Each server handles all actions from clients and updates its own database. The servers in turn send updates to each other over a high quality link, such as fibre, to maintain database consistency at high speeds.

3.2.2.1 Advantages

The replication-based scheme improved upon sharding in that users are now part of the same virtual world. In order to enable the virtual world to host more users, more servers may be added.

3.2.2.2 Disadvantages

The problem with this system is that two users standing next to each other in the virtual world, might be on different servers and, therefore, experience two slightly different worlds. Users are more likely to interact with other users who are close to them, than users who are far away. For this interaction to occur, the messages might now have to traverse servers. Sending a message to another server is a much more expensive operation than transferring information between two users within the same server. This required a message to be sent to another server, as opposed to an internal signal within the same server.

The single shared database still limits the number of concurrent users that can connect to the world.

3.2.3 Zone or region-based

The zone-based (also called region-based) storage model divides the virtual world into zones or regions, which are hosted on different servers [1], [26]. Region-based models can be divided into static region and dynamic region approaches. In static region approaches busy regions are hosted on their own servers, while multiple quiet regions are hosted on a single server. Dynamic regions are being investigated, where regions can be dynamically shifted from one server to another, in order to balance load [26].

3.2.3.1 Advantages

The advantage of region-based state management is that all users who are close to each other are likely to be hosted on the same server. Users interaction on the same server are more responsive and require less overhead than user interactions between servers.

3.2.3.2 Disadvantages

The disadvantage of the static region approach is that it does not scale well when one region is suddenly populated with users. This type of behaviour happens quite regularly and is known as flocking [25]. When users find something of interest in a region, many users will flock to that region. The solution to flocking has been over provisioning of resources to handle peak loads. When peak loads do not occur, the over provisioned hardware is, however, wasted.

If servers become overloaded, while others have resources to spare, some servers may have to be brought off-line to balance region load amongst the over-loaded and under-loaded servers. This will cause the VE to be unavailable during that time.

Dynamic regioning solves the issues related to flocking and transient loads, but this approach adds overhead and significant complexity with regards to the migration of the data and the handling of user actions while the data are in transit. No current, commercially successful, VE is known to make use of dynamic regioning.

3.2.4 Object-based

The object-based storage model equally distributes all virtual objects amongst the servers [71], [73], [78]. For an MMVE, most of these objects are expected to be user objects.

3.2.4.1 Advantages

The advantage of this method is that the system load is fixed for a certain user population and that the load is equally distributed amongst all servers. This allows for more accurate prediction and provisioning of resources, but still does not handle transient loads well.

3.2.4.2 Disadvantages

The inter-server communications are random and also more than the inter-server communications for a region based system. The reason for this is that the number of user interactions increase with a decrease in the distance between users in the virtual environment. Users playing together move together, chat and interact with

NPCs together. For a region based model, all user-neighbour interactions remain local to the server.

3.2.5 Conclusion

Examining the disadvantages and advantages of each classic storage method, the two methods that seem most promising are the region-based and object-based approaches. Sharding does not allow for a single world or any form of load balancing and replication-based schemes still have the single database as a bottleneck.

Because of the advantages of region-based storage, it has become a popular storage architecture to employ for virtual environments. Most virtual environments can be divided into regions and the virtual environment can be designed in such a way that region boundaries are difficult to cross and that users can only cross these boundaries when moving from one distinct area into the next. This can be done by placing high mountain ranges on region boundaries, with a single pass through the range. This prevents users from switching between regions if they stand on a region boundary. It also ensures that most users that interact, do so within the same server, which decreases inter-server communication.

An issue with the region-based approach, compared to the object-based approach is that each region contains many users and, therefore, requires a large server to host a region. Some region-based servers are themselves server clusters to handle the client load. An object-based server on the other has a fixed load, which can be determined based on the number of objects each server stores. Because of the success of the object-based and region-based storage schemes, P2P storage schemes are also based on either of the two. Because peers are less powerful than servers, the object-based approach seems to have gained more traction, since server load is fixed. If a region-based approach is used, peers are required to host large regions, which might overload them. If a the region is made too small, the amount of inter-peer communication might become prohibitive.

3.3 P2P MMVE storage requirements

In order to evaluate the suitability of the existing P2P MMVE storage architectures we will first identify and define a number of requirements for storage. These requirements are based on the reviewed literature and is a combination of the various issues that each of the existing architectures identified. In the following section we will review the existing architecture using the metrics defined in this section and show that no one architecture meets all the requirements.

The key challenges related to P2P MMVE storage models are: fairness, overhead, reliability, responsiveness, scalability and security. All storage models will be reviewed with these characteristics in mind. The identified metrics allow for different storage models to be compared and provide a measure of the applicability of any storage model to P2P MMVEs.

3.3.1 Scalability

Scalability underpins all evaluation criteria. This implies that for a system to be scalable, all other evaluation criteria should be satisfied for large numbers of peers and data. For this reason, scalability will not be explicitly reviewed in the following storage types. Rather, all other evaluation criteria will be evaluated for a large number of peers, thereby taking into account scalability.

The question of what constitutes a large number of peers arises. To establish what an adequate number of peers is, current MMVE architectures can be used for inspiration. *It is proposed that to classify a system as sufficiently scalable, 2500 peers is the smallest number of peers that a system must accommodate.* This is the number of users per server (realm), currently supported by most active C/S MMVEs. For a system to be classified as *truly scalable*, it is believed that the architecture should support 60,000 concurrent users, twenty times more than a sufficiently scalable system. This is the number of peak concurrent users (PCUs) currently supported by the super computer used to host Eve Online [21]. These two measures will ensure that a system is as scalable as other currently available architectures.

For systems that will support the MMVEs of the future, it is believed that a target of 1 million peers should be used. This number is sixteen times that of a truly scalable system and the peak concurrent user count of World of Warcraft in China in 2008 [110]. Systems that will support these numbers can be classified as *highly scalable*. It is important to note that no current systems support such a large PCU count on a single server cluster. The PCU count presented for WoW is the PCU count over all server clusters hosting WoW in China [110].

3.3.2 Fairness

Ensuring fairness in the system means distributing load evenly according to the abilities of individual nodes. This ensures that a small number of nodes do not provide all system resources required for the system to function, but that all nodes contribute what they can, in order to support the system.

Fairness does not exclude heterogeneity of nodes. It is accepted that nodes have different amounts of available resources. It also does not exclude node specialisation. It is accepted that some P2P architectures require specialised nodes or require a set

of different types of super peers to function. Fairness merely states that when all node contributions are added, the sum of all node contributions should be equal.

Fairness is based on the assumption that nodes are required to donate resources to the network for which their users might have to pay. One cannot expect that a small percentage of users pay for the network resources, while other users use it for free. For a storage system, nodes that store more data will have to serve more data to other nodes. This means that nodes that store more data will also have to contribute more bandwidth. The extra bandwidth means that the users of the nodes will either have to pay more for the extra bandwidth. Where users do not pay for bandwidth, they will still have that much less bandwidth to use for their personal use.

If objects are distributed across more nodes, it also means that each node will have to store fewer objects, than if only a small number of nodes stored objects. Where fewer nodes store objects, the nodes that store objects will each have to store more objects than if the load was more evenly spread.

Fairness can be evaluated by evaluating the distribution of virtual objects amongst all nodes in the P2P network. This is, however, a measure of fairness that implies equal distribution and does not take into account the heterogeneity of nodes. The distribution can be measured at a file level, i.e. what is the standard deviation of the number of files contained on each node, or on byte level, i.e. what is the standard deviation of the number of bytes stored on each node. A lower standard deviation will point to a fairer data storage scheme, if it is assumed that all nodes should contribute an equal amount of storage.

3.3.3 Reliability

For the storage to be reliable, it must be impossible for data to be lost, and stored data should always be available when a node requests it. Reliability encompasses both robustness and availability. Robustness means that the data should be resilient to network churn and availability means that data should be available to any node in the network, if the node has permission to access the data.

Reliability can be measured as the percentage of successful storage and retrieval requests, compared to the total number of storage and retrieval requests.

3.3.4 Responsiveness

Responsiveness is the only requirement that is of lower priority for state persistency, compared to state management. Since it is assumed that all low-latency data requests will be handled by state management, state persistency is only used to archive environment state, and thus responsiveness is less of a priority. Highly responsive

state persistency will still improve data throughput and ensure that less data will be lost in the event of a failure, but highly responsive state management is required for the correct functioning of the state consistency mechanism.

To ensure system responsiveness, objects must be stored or retrieved in real-time. With real-time, it is meant that data should be available within a certain time frame that would ensure correct functionality of the MMVE requiring it. For classic MMORPGs, acceptable latency is anything under a second [99], but preferably under 500ms. The variance in data retrieval times should be small. *Responsiveness can be measured by the time it takes to read or write data to the storage network.*

3.3.5 Security

The storage system should store data securely. It should not be possible for data to be altered in ways that are inconsistent with the environment logic. It should also be possible to identify nodes that alter the data in a malicious way. This also adds the requirement that nodes should be authenticated in the storage system and that only authorised nodes should be able to alter data. According to [8], security is the combination of a number of objectives: Authentication, Authorisation, Data Integrity, Confidentiality, Availability, Trust, Privacy and Identity Management.

For a storage model to address the above mentioned security objectives, a certification scheme with public and private key encryption is required. Such a scheme allows for the identification of users, and by having users sign any storage interactions, every change made to the storage system can be tracked. This is a major differentiating factor from classic distributed storage systems such as Freenet, where a primary objective is anonymity. If all operations are logged and all users have to be identifiable for a secure system, no user can be anonymous.

It is difficult to define a single measurement criteria for security. In this work we take the same approach as the approach taken with scalability. Where the scalability criteria specifies that all requirements should be met for large numbers of nodes, the security criteria is defined to specify that all requirements should be met for various percentages of malicious nodes present in the network. *In other words, security specifies that the system should remain reliable, responsive and fair for various percentages of malicious users in the network.*

3.3.6 Overhead

The amount of bandwidth required by the storage system, compared to the amount of objet data stored, should be low. Any networked communications protocol should attempt to limit its overhead. A system with low overhead has a faster effective data

transfer rate, compared to a system with high overhead if the same bandwidth is available.

For a given storage type, or any network layer, overhead can be measured as the factor of data received from or delivered to the higher layer, compared to the bandwidth required to handle the higher layer requests in the storage layer.

For the storage systems discussed below, little empirical evidence is available as to the overhead that the various systems require. Overhead is also directly proportional to the number of redundancy in the storage systems, i.e. the number of replicas used. This is usually a configuration parameter of the storage system and it is difficult to judge what the required number of object replicas are to achieve desired levels of reliability. Due to these issues, overhead will not be reviewed in this literature study. Overhead will, however, be evaluated when our own storage system is evaluated in Chapter 6, and the overhead measured will be compared with the overhead of a currently implemented storage system (overlay storage, as discussed in Section 3.4.2).

3.4 Overview of P2P MMVE storage types

In this section four approaches to state persistency in P2P MMVEs is discussed: *super peer storage*, *overlay storage*, *distance-based storage* and *hybrid storage*. Each storage type is evaluated in terms of the requirements identified in Section 3.3.

Two other types of storage sometimes described in P2P MMVEs papers are *centralised storage* and *individual storage*. Centralised storage is storage in a centralised database, the same as for a C/S MMVE [68], [93], [62]. Centralised storage for an MMVE requires the same large expensive servers and high bandwidth as required by a classic C/S architecture and therefore does not fit into the P2P MMVE paradigm. For this reason, centralised storage will not be evaluated in this chapter.

With individual storage, user data are stored on a user's own computer [24], [4]. These architectures do not address the storage of NPC state or mutable objects. These objects cannot be as easily mapped to a single peer in the network as user state can and, therefore, require a mapping mechanism to decide where to host these objects. Although individual storage will not be explicitly discussed in this chapter, it can be regarded as a subset of distance-based storage, which will be discussed in detail in Section 3.4.4.

Table 3.1 presents our characterisation of current storage systems according to the characteristics defined in Section 3.3. In the rest of this section, we shall motivate our characterisation of the various storage systems.

Storage type	Reliability	Responsiveness	Security	Fairness	Overhead
Super Peer	Medium	High	Low	Low	Low
Overlay	High	Low	Medium	High	High
Hybrid	High	High	Medium	Low	High
Distance-based	Medium	High	Low	Medium	High

Table 3.1: Differences between storage mechanisms

3.4.1 Super peer storage

Super peer storage makes use of the region-based P2P MMVE consistency model described in Section 2.5. Super peer storage relies on the super peer storing all information that is in its region.

A mechanism is required for selecting a super peer for the region as well as migrating the super peer when the existing super peer leaves the network.

3.4.1.1 Fairness

The super peer storage model has many potential issues, of which overloading of the super peer is one. A super peer could be relatively easily overloaded if a region becomes too crowded, since a super peer is usually a personal computer of some user in the virtual environment and not a specialised server.

Super peer storage is not fair, since a peer with extra resources is expected to act as super peer to the region whilst other users do not contribute storage. This is contrary to the principle of P2P, where all peers contribute resources.

3.4.1.2 Reliability

In a P2P network, with a high rate of churn, users are expected to constantly leave and join the network. This requires that redundancy mechanisms have to be developed that would ensure state data are always available, even when a super peer leaves the network. A possible solution is having redundant super peers in each region that take over hosting responsibility when the main super peer leaves.

It is important that the main and backup super peers always possess consistent states, even during a transition from main to backup. Other schemes to support improved reliability deal with reputation mechanisms for super peers. Super peers that have more resources and stay in the network longer are preferred during super peer selection, using reputation mechanisms [38].

3.4.1.3 Security

Super peer storage is not secure, since a single peer stores all environment state of the region. If a single peer is allowed to house the user information of a large group

of users, it might become possible for such a peer to maliciously modify the data. The issue is not only that modification of the data might be possible, but also that it would be impossible for the cheating to be detected, because of no centralised logging. Locally obtaining access to data will circumvent the protections created by a certification system, which would then pose a threat to all security objectives protected by the certification system as mentioned in Section 3.3.5.

A scheme that would improve the security of this systems has been proposed, where every event is also sent to the backup super peer of the region [54]. The main super peer responds with the update and the backup super peer responds with a hash of the update. A peer can then check whether the hashes match to determine whether the data has been received correctly. A hash is not the state update itself, so will be smaller, but the events that have to be sent to all super peers will increase traffic in the network and bandwidth usage by peers.

3.4.1.4 Responsiveness

The advantage of super peer storage is its responsiveness. All data are stored on the super peer, which means that storing data is a low latency operation. The regional state can be stored and retrieved at high speeds, making the system responsive. Data retrieval from such a storage is as fast as data retrieval from a server. Peers can request data from a super peer and the data can be returned to the peer in one hop after transmission of the request. Super peers may, however, become overloaded with requests and thereby increase the latency of the system.

3.4.1.5 Existing architectures

Knuttson et al. [66] employ regional super peers called “coordinators”, to host all shared object states. The coordinator is chosen as the peer whose ID is closest to that of the region ID. The region ID is a SHA-1 hash of the region’s textual name [81]. This mapping makes it unlikely that the coordinator will be a member of the region. The advantages of such a selection scheme is that the opportunities for cheating are reduced, because the data are hosted on a peer that has no or little interest in the data, as described in Section 3.4.4.3.

Coordinator hand-offs also occur less than if the coordinator was an elected member of the region. If this is the case, a new coordinator has to be chosen every time the current coordinator leaves the region. In this scheme, hand-offs only occur as a result of network churn, which is far lower than the number of users moving from one region to another.

Reliability is achieved by maintaining backup coordinators as the Distributed Hash Table (DHT) neighbours of each region coordinator. One method by which

redundant region coordinators are maintained in [66], is to create backup coordinators on peers with IDs closest to the current coordinator. This means that if the main coordinator fails, all data will automatically be routed to the backup, because of the feature of DHTs. This method is similar to how reliability is achieved in overlay storage as described in Section 3.4.2.1.

The most mature form of super peer storage is found in P2P Second Life [115] in the form of Walkad [114]. Walkad assigns multiple coordinators to a single cell. For every object stored in a cell, a copy of the object is stored on every communicator. A peer requiring an object selects a single communicator from the set of available coordinators to retrieve the object from. Walkad also employs dynamic regioning, which means that there exists some threshold of objects per cell which aids in scalability and prevents overloading of the super peer.

Increasing the number of replicas maintained per cell, increases the number of nodes that participate in the storage system, thereby increasing the fairness of the storage system. There will, theoretically, be some limit where increasing the number of replicas further will have a negligible effect on overall reliability. The question then becomes, for this situation, what is the percentage of peers that contribute to the storage system. Another issue that might occur is excessive bandwidth usage by the system when 20 or more replicas are maintained per object per cell.

It should also be noted that the required number of replicas will be a function of network churn to maintain a specific reliability. The number of users present should not affect reliability. The question then arises: how is the fairness and bandwidth requirements of the system influenced when tens of thousands of users enter the virtual environment.

3.4.2 Overlay storage

Overlay storage is classified as using any type of structured P2P overlay to store data in a distributed fashion. This is a broad definition, which basically encompasses any P2P distributed storage currently in use. Some examples and a comparison of different distributed storage techniques can be found in [56]. The reasoning is that any P2P distributed storage can be used to only store virtual objects. Therefore, distributed storage is used as a distributed database for virtual objects.

3.4.2.1 Reliability

Overlay storage can be made reliable, using redundancy. One method used to achieve high reliability in structured overlay storage is to store R replicas for any file stored, in order to ensure the availability of the file. These replicas are stored at the neighbouring peers of the peer containing the original file. Neighbours are the R peers

whose IDs are closest to that of the root peer. By the characteristics of DHT distance-based routing, if the peer with the original data leaves the network, packets will automatically be routed to the neighbouring peer, which stores a duplicate. This technique ensures high availability of data and the number of duplicates can be chosen according to the reliability of the network.

3.4.2.2 Responsiveness

The most significant issue with overlay storage is the delay incurred when storing and retrieving data. As data can be stored anywhere on the network and the network is not fully connected, an average of $\mathrm{O}(\log(N))$ hops are required to retrieve or store a data item [94]. Although this is a sufficient order complexity for a routing algorithm in a large network, it is not sufficient to support a real-time application. For responsive MMVEs, a distributed file system is required that allows for real-time file storage and retrieval.

The mechanism by which churn is handled, described in Section 3.4.2.1, also improves responsiveness. This is achieved, because IDs created by the random hash function ensures that a peers's neighbours are distributed randomly throughout the P2P overlay. This random distribution ensures that file replicas, stored at a peer's neighbours, are uniformly distributed throughout the P2P overlay network.

3.4.2.3 Security

The overlay storage model is more secure than the super peer storage model as data are distributed amongst all peers and redundancy and quorum techniques can be implemented to ensure that files are retrieved with a high level of security.

To ensure a secure system, copies of files have to be saved at different locations. If a file is retrieved, all copies must be queried and received. All received copies then have to be compared to ensure that the contents are correct. This is usually referred to as a quorum mechanism. Additional network overhead is introduced, as well as additional load on peers to serve as file copies.

The network overhead can, however, be reduced by having file replica peers only send hashes of the files, which may then be compared at the requesting peer. Hashes require less bandwidth, while still allowing a requesting peer to check update validity by hashing the received update and comparing with the received hashes.

3.4.2.4 Fairness

Overlay storage is fair, as all peers share file data and requests equally. The system might be made fairer by taking into account the heterogeneity of peers. Peers do

not all possess the same resources, something which a truly fair system should take into account. The difficulty with using such a scheme is that peers can be made to report incorrect resource information in order to reduce their resource donation requirement. This is where incentive mechanisms have to be investigated as well as ways to ensure correct resource reporting.

3.4.2.5 Existing architectures

Douglas et al. designed a P2P MMVE architecture in 2005 [35] and implemented state persistency using a distributed storage implementation, which they developed in 2003 [55]. The storage system allows for the manipulation of spatial data, while also implementing range queries. This enables the system to store and retrieve data that exist in a certain area of the virtual environment. In the MMVE architecture they developed, state persistency is implemented by the "Spatial Data Service" (SDS), which is a distributed storage architecture that uses the Chord P2P overlay for routing [104].

PAST has become a popular way to implement state persistency. This is the approach proposed by both Hampel et al. in 2006 [54] as well as Fan in 2009 [37]. In these publications, it is said that PAST is used to store the global virtual environment state, but never is detailed what is stored and how regularly it is stored. User information is supposed to be stored as virtual environment state, but from the papers it is unclear how position updates are handled. It is not clear whether the last position of a user is stored at all or how regularly it is stored. It is important to know how position updates are handled in the virtual environment, since position updates are the most common type of update [66].

PAST has not gained much commercial adoption, because of the lack of support for keyword searches, which is a requirement of most distributed storage networks, where users constantly search for content. Keyword searches are, however, not required by the storage mechanism of an MMVE. The IDs of the items stored in the network will be known. A player's inventory can, for example, always be called `(player_name)_inventory`. A hash of this file name will find the correct file. This, and the use of Scribe, is why PAST has become popular with researchers of P2P MMVEs.

PAST [36] uses Pastry to implement a distributed storage system. Files that have to be stored are given IDs, by using some hash function, for example SHA-1 [81]. The file, along with the ID are sent as a message over the overlay. The messages is then routed to the peer whose ID is a closest match of the file ID, where the file is stored. If any peers wish to retrieve the file again, it only requires the file hash. A "get" message can be sent to the overlay, where the overlay will route the

message to where the file is situated and retrieve the file.

In an effort to increase responsiveness, PAST also employs caching techniques [94]. If a peer forwards many queries for a file, that peer can elect to cache the file to improve the responsiveness of the system. This caching can only occur if the peer has space available. If a peer has cached a file and another file is explicitly inserted into the peer, it can elect to remove the cached file in order to free up space. This means that the success of the caching mechanism is directly related to the level of storage utilisation. Higher utilisation will prevent files from being cached.

The main difference between the implementation by Douglas et al. and the PAST implementations, is that the former supports range queries on spatial data, which allows for a set of objects to be returned, queried by their virtual position. With PAST, the exact ID of an object is required before it may be retrieved. Using PAST is the simplest means by which environmental state persistency may be implemented, but PAST is not necessarily the application best suited to environmental state persistency. There exists a need for more research into appropriate state persistency mechanisms for P2P MMVEs.

3.4.3 Hybrid region-based storage

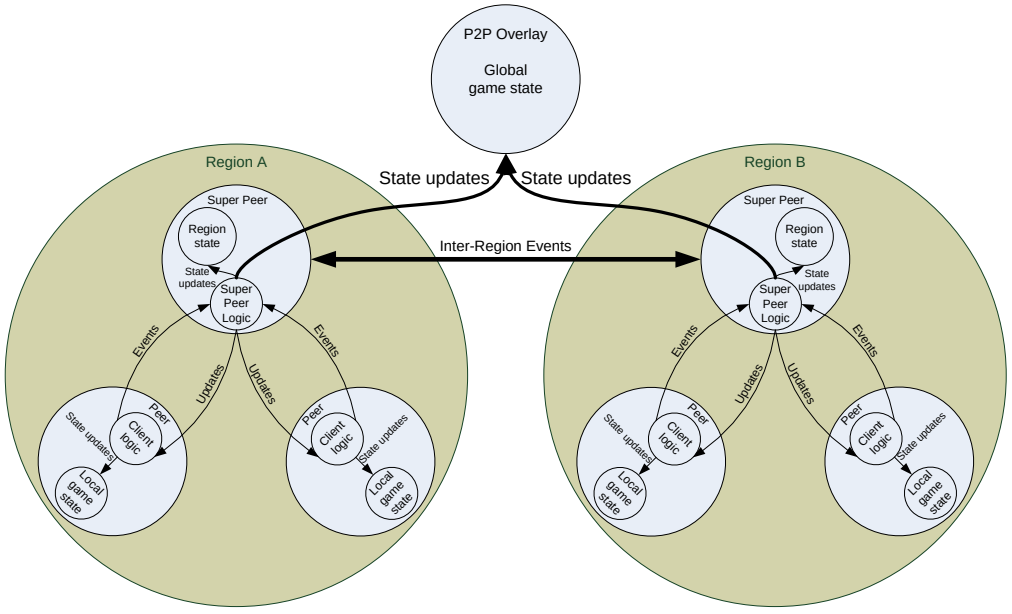


Figure 3.1: Hybrid region-based super peer storage with backup overlay storage consistency model

Figure 3.1 shows a type of Super Peer/Overlay hybrid storage implemented in [61]. The model depicted in Figure 3.1 uses overlay storage, managed by regional

super peers. The world is divided into regions, with each region controlled by a super peer. The complete region state is cached at every super peer, the same as with super peer storage. There also exists a backup overlay storage architecture, to which data may be backed up for long term, redundant and secure storage. The hybrid region-based overlay storage contains many improvements over pure overlay storage as will be discussed in the following sections.

3.4.3.1 Reliability

Because of the use of overlay storage for backup, the hybrid region-based storage is almost as reliable as a pure overlay storage. It is classified as almost as reliable, because there is a delay between when data changes and when it is updated in the overlay. If a super peer fails during this time and the data was not backed-up to the overlay, that data could be lost. Backup super peers can, however, be implemented as described in Section 3.4.1.2 to improve the reliability of the hybrid storage model.

3.4.3.2 Responsiveness

Because all regional files are cached at super peers, the system is as responsive as super peer storage.

3.4.3.3 Security

Security in hybrid storage is still an issue, because of the inherent problems of the super peer storage model. Although it is more difficult for peers to access and manipulate data stored in the overlay, a malicious peer promoted to super peer status may manipulate the region data it controls.

It does seem possible to achieve higher levels of security, by checking data received from super peers, against the data stored in the overlay. Care should be taken with such a scheme, because the rate of change of data at the super peers might be higher than the rate that data are submitted to the overlay. Another issue is the time delay between data received from a super peer and data received from the overlay.

3.4.3.4 Fairness

The issue of fairness is also still present in hybrid storage. The system is fairer than super peer storage, since all peers share the load of the overlay storage, but there still exists the unfairness of the super peer storage. As all data exists in both the overlay storage as well as the super peer storage, the system is as unfair as super

peer storage, because the same quantity of data as in super peer storage is not being distributed evenly amongst all peers.

3.4.3.5 Existing architectures

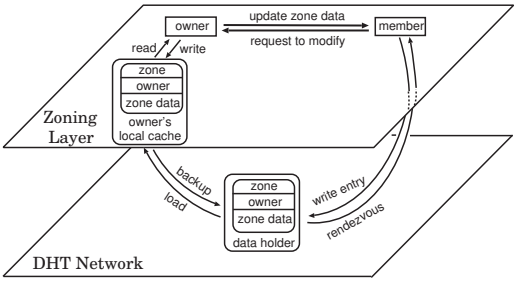


Figure 3.2: Zoned Federation Model [61]

The first hybrid state persistency model for P2P MMVEs was proposed by Iimura et al. in 2004 [61] and called the "Zoned Federation Model", shown in Figure 3.2. The regional super peers are called "Zone Owners", which handle all events by clients in their zone or region. In the Zoned Federation model, a Zone Owner acts as the primary storage medium for all object states in the zone or region. As shown in Figure 3.1, this is analogous to an update based model, divided into zones. The difference here is that the environmental state of all zone owners are regularly backed-up to overlay storage. The zoned federation model can thus be seen as a super peer/overlay storage hybrid. The super peers storage provides for low latency data storage and the overlay storage provides security and reliability.

An extended abstract, published by GauthierDickey et al. in 2004 [45], proposed to distinguish between permanent and ephemeral data. Permanent data are described as data that should exist at all times and ephemeral data are described as data that need only exist for as long as its owner is in the virtual environment. An item, being dropped by a dispatched NPC, can be considered as ephemeral. When the peer on which the data is hosted leaves the area, that item can disappear. An example of permanent data is a player's inventory contents, which can further be classified as participatory data or a player's house, which can be classified as existential data. Participatory data are data that need only be available when a specific player is in the virtual environment and existential data are data that should be available, even when a certain user is not present in the virtual environment.

Categorising data by how long and under which circumstances the data should exist, may assist in the design of the storage model. Since ephemeral data does not have to exist after the player has left the virtual environment, it may be stored in

primary memory. Participatory data might also be stored on the player's computer, but security issues will have to be kept in mind. Existential data will have to be stored somewhere other than on the player's computer, since other players will require the data, even in the absence of the player that might have left the virtual environment. GauthierDickey et al. did not explore how their data classification scheme might be translated into a state persistency model.

In 2004, Merabti and El Rhalibi describe the implementation of the "Time Prisoners" MMOG [76]. They propose a hybrid network topology, where the network architecture starts out as client/server, but as players are added to the world, a P2P architecture is created. There exists a single central database, that acts as an initial entry point into the network. Peers can also become super peers in the form of region servers. Region servers maintain all the state in the region and clients only interact with a region server, after having been assigned one by the central server.

Merabti and El Rhalibi also discuss the "data storage" requirement and propose the use of a data storage architecture based on the Freenet project [28]. Freenet is a distributed storage facility that uses a Darknet to ensure user anonymity when distributing files [15]. A system such as Freenet is designed for general file sharing, which means that no focus is placed on achieving the high levels of responsiveness required for MMVEs.

Without defining it as such, Merabti and El Rhalibi propose a form of super peer storage for state management and overlay storage for state persistency. Where super peer storage is the storage on the regional servers and overlay storage is the storage in Freenet. Freenet is a an overlay storage, which is not as responsive as direct connections. The work by Merabti and El Rhalibi is one of the more mature hybrid region-based storage systems encountered.

3.4.4 Distance-based storage

Distance based approaches, such as the Voronoi storage approaches [18], [57] and some more general approaches [13], [42], store object data on the peer closest to the object in the virtual world. Some distance metric is used to determine on which peer an object should be stored.

3.4.4.1 Responsiveness

An issue with the above reasoning is that multiple peers are usually interacting with a single object. The examples of the NPC monster and trader are again relevant. Usually many users interact with a trader NPC and usually users attack monster NPCs in groups.

Multiple user interactions are, however, not as big an issue as others have suggested [39]. In the best case, the object being used by a user is also hosted on that user's peer. If another user requires use of a remotely hosted object, that user may still interact with the object, where the host peer is simply acting as a server to that user. This essentially means that every user hosting an object becomes a server for that object. In the case where a user interacts with an object hosted locally, there is no object latency. In the case where a user accesses a remotely hosted object, there is only one hop latency, the same as with a C/S or super peer application.

Issues with this approach stem from the fact the users are constantly moving. When users move, the objects in their regions change. Authoritative objects, therefore, have to be constantly transferred from one peer to another, which might cause significant network traffic. An object in transit might also delay interaction with that object. Because object transfer introduces overhead into the system, how regularly an object has to be transferred and whether the number of transferrals produce sufficiently low overhead to implement a real-time virtual environment, still have to be investigated.

Voronoi-based storage schemes also become unresponsive when communications are no longer between neighbours, but between two arbitrary peers in the Voronoi overlay. When such communications occur, the average required time to route a message is $\mathrm{O}\left(N^{1/2}\right)$ for a two-dimensional configuration. Advancements have been made that suggest augmenting the Voronoi overlay with additional links to far off peers to create a small world network. This reduces the average routing time to $\mathrm{O}\left(\log\left(N\right)\right)$, the same as for overlay storage [102].

3.4.4.2 Reliability

Reliability, because of network churn, is still an issue. Peers will leave the network whenever a users leaves the virtual environment, which makes it a common occurrence. When peers leave the network, the objects that are stored on that peer should still be accessible to other users. This will require transferring all objects contained in the leaving peer to another object, still present in the network. No literature could be found that dealt with the issue of reliability in a distance-based storage network.

The same solution that is used for overlay storage, namely the presence of redundant peers, might also be implemented for distance storage. Another structured overlay might be used to implement this redundancy in exactly the same way it is done with hybrid storage (mentioned in Section 3.4.3.1).

3.4.4.3 Security

The main issue with the distance based scheme is security. Peers that have the most interest in an object also have the most interest to manipulate that object in ways inconsistent with the environment rules. When objects are hosted on peers that have the most interest in them, there will be a strong temptation to try and manipulate these objects. Because these modification are all local, it is also not possible to log the alterations and detect cheating. Means by which local objects can be secured have to be found or distance based algorithms with quorum need to be investigated.

This security issue is similar to that of super peer storage, in that local data can be accessed, thus circumventing the certification system. With this circumvention, the objectives of authentication, authorisation, confidentiality, trust, privacy and identity management are all compromised.

3.4.4.4 Fairness

Distance-based storage is relatively fair as all objects are distributed amongst all peers. Where the system becomes somewhat unfair is when a peer is nearest to a large number of objects. For Voronoi regioning approaches, this is when a peer's Voronoi region contains many more objects than the average number of objects hosted by other peers. This might not be a major issue, depending on how long the objects have to be hosted on the overloaded peer. This will depend on the movement of the overloaded peer as well as that of neighbouring peers. The use of aggregators as a proposed solution to this problem is discussed in Section 3.4.4.5.

3.4.4.5 Existing architectures

Bharambe et al. created the Colyseus architecture in 2006 [13]. The architecture is designed to support First Person Shooter (FPS) games and implemented to function with Quake II. Mutable virtual objects are stored on the peer that is nearest to the object in the virtual environment. An "object placer" component is mentioned, but the details of the placement algorithm are left for future work. The architecture also does not implement non-volatile state persistency, since this is not required for normal FPS games, where object states need only exist to the end of a round and where players generally do not leave before the end of the round. This means that object states are only stored in primary memory, until the end of a game round.

The Solipsis architecture was created by Frey et al. in 2008 [42]. The architecture uses Voronoi diagrams to create virtual regions. Stationary objects are maintained by site peers until a user picks up an object. When an object is picked up, control of that object is transferred to the user that picked up the object. The Solipsis

architecture focusses on distributed physics computation and when a user gains control of an object, that user is responsible for the object's physics computations. That user should also save all object state until a new user takes the object, at which time control is transferred to her. Control can also be transferred back to a site peer if an object remains stationary for some time.

What differentiates the Solipsis distance-based storage from the other architectures presented in this section, is that object states are only handled by peers as long as those peers directly use an object. At other times, those objects are handled by site peers. This differs from other distance-based storage techniques, where all objects that are nearest to a user in the virtual world are controlled by that user.

In 2008, papers were published by Buyukkaya and Abdallah [18], and Hu et al. [57], proposing to use Voronoi diagrams [2] to implement distance-based storage. Voronoi state management schemes host the mutable objects on the peer in which region the object exists. As peers move around in the virtual world, the Voronoi diagram has to be constantly recalculated and objects have to be moved to new owner peers as the regions in which they fall change. The significant advantage of the Voronoi approach is that peers only require connections with their neighbours, and peers within their AoI.

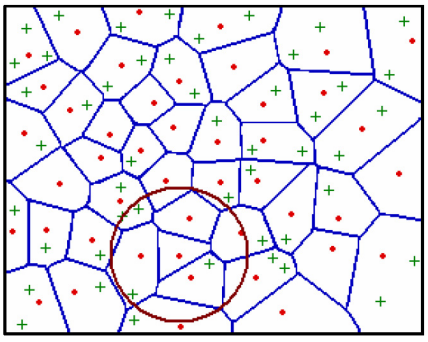


Figure 3.3: Voronoi Diagram [18]

Voronoi storage approaches use Voronoi diagrams to determine on which peers objects should be stored. Given a set of points, the Voronoi diagram of the set of points is the partition of the plane, which associates a region around every point in such a way that all other points contained in the region are closer to the centre point than any other point in the set. Figure 3.3 shows a Voronoi diagram, where the lines define the region boundaries, the dots define the users, which make up the set of points for which the diagram was calculated, the plus signs represent mutable objects and the circle represents the AoI of a central point in the set.

For the Voronoi approaches a peer controls and hosts all objects within its Voronoi region. The reasoning is that there is a high probability that the user closest to the object is also the user using the object. Examples of this are where a player is trading or fighting with an NPC.

A thesis by Chang also describes the Voronoi approach in more detail and how to achieve state consistency amongst all peers in a Voronoi network [22]. What distinguishes this work from the others is the implementation of a load balancing mechanism. When peers get overloaded, another, more powerful peer is chosen as an Aggregator. The Aggregator assumes responsibility for a larger area that encompasses multiple peers. This scheme will reduce the load on peers with minimal resources, but it is uncertain how this would reduce load when peers with an average number of resources in an area become overloaded.

The works by Buyukkaya and Abdallah, Hu et al., and Chang form part of the VAST project, which is being created to be a fully functional P2P overlay architecture, using Voronoi diagrams as its basis [117]. The reason why state persistency in Voronoi-based P2P MMVEs architectures are mostly distance-based approaches, is because the Voronoi diagram immediately identifies which objects are closest to a particular peer, and therefore, which objects that peer has to host. Architectures not making use of Voronoi diagrams still require some other mechanism to allocate object hosting to peers.

3.5 Conclusion

After providing an overview of the classic C/S and C/MS state persistency techniques, the chapter classified P2P MMVE state persistency techniques into super peer based, overlay based, distance based and hybrid storage. The advantages and disadvantages of each method were discussed after identifying key challenges that state persistency techniques have to solve. These challenges are: fairness, overhead, reliability, responsiveness, scalability and security.

Super peer storage can be seen as a C/S type storage implementation, where every region has a super peer that stores all data for that region. Overlay storage is a fully distributed, P2P approach to storage, where every peer stores data as part of a P2P overlay network. Hybrid region based storage combines super peer and overlay storage to improve the overall storage performance. Distance based storage is a different paradigm that stores virtual objects on the nearest peer to that virtual object. This type of storage is usually characterised by the use of Voronoi diagrams to determine which peers are nearest to which objects.

Super peer storage is characterised by its high level of responsiveness and ease

of implementation. Overlay storage is characterised by its high level of fairness and reliability. Hybrid region-based storage combines the two previous schemes and has high levels of reliability and responsiveness. Distance based storage is also responsive and can be made both reliable and fair.

Security is still an issue for all storage types. Super peer storage has the issues that are usually present in a centralised system, namely low fairness, security and also not being reliable and not resistant to failure. The main issue with overlay storage is its unresponsiveness, because of the routing delay in the P2P overlay. Hybrid storage suffers from the disadvantages of super peer storage, namely low fairness and security, due to all files still stored on super peers. The main issue of distance based storage is security, because users that have the most incentive to alter object states own those objects.

Distance based storage is still fairly immature, but shows a lot of promise. No publicly available implementation could be found for this type of storage. Where this type of storage was used, it was explained as distance based storage, but most of the details were left out. Distance-based storage holds the most promise to fulfill all P2P MMVE requirements.

We conclude that there exists no single state persistency architecture, currently in use, that is suited to P2P MMVEs. None of the storage techniques reviewed meet the requirements of a real-time distributed application, such as an MMVE. What is required is a state persistency architecture, specifically geared towards data persistency in P2P MMVEs, that meet all the challenges of the application.

To compensate for the deficiencies of the existing architecture, we combine overlay storage, distance-based storage and to some degree, super peer storage to propose a hybrid architecture, which meets all storage requirements. The design of this novel architecture, called Pithos, is discussed in the next chapter.

Chapter 4

Pithos Design

The generic state consistency model was presented in Section 2.3, but one of the key challenges that still remain was determined to be the design of an authoritative storage module specifically tailored to P2P MMVEs. The authoritative storage module identified in the generic consistency model state management and state persistency for the authoritative objects.

In the previous chapter, we've identified the main requirements of P2P MMVE state management and persistency as: fairness, overhead, reliability, responsiveness, scalability and security. It is argued that none of the current approaches to state persistency satisfy all identified requirements. This chapter proposes a novel design, called Pithos, that satisfies all the identified requirements for P2P MMVEs.

The novelty of Pithos lies in its group-based distance-based approach that supports both a responsive and a fair storage system, while also taking into account security aspects of scalable distributed storage. There are some storage systems that provide responsive or fair storage, but none that provide both. No storage system, designed specifically for P2P MMVEs, have taken security into account.

If Pithos is incorporated into an existing P2P MMVE consistency architecture, it will add the ability to robustly handle both state management and state persistency. The addition of a robust state management and persistency mechanism, specifically designed for P2P MMVEs, will bring us one step closer to the creation of a complete P2P MMVE architecture.

4.1 Use cases

The purpose of Pithos is to allow for efficient object storage and retrieval that satisfies all identified requirements. As is evident from authoritative storage in the flow diagram in Figure 2.4, Pithos will interface directly with the VE logic, as well

as receive updates from the update generator. For the purposes of this discussion, update generation is assumed to be part of VE logic.

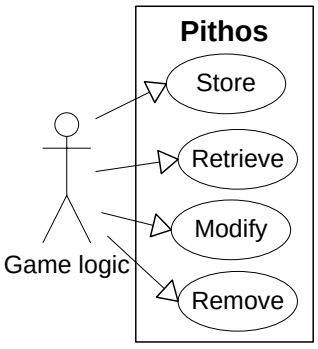


Figure 4.1: Use case diagram of Pithos

As shown in the use case diagram in Figure 4.1, the VE logic should be able to use Pithos in four ways: store, retrieve, modify and remove. These are the use cases generally required of any storage system.

4.1.1 Store

The VE logic will store data when a new object is added to the VE state. This can happen as a consequence of an event leading to the generation of a new object. An example of this is a rocket firing at a target. This event might generate a missile object to be sent towards the target.

4.1.2 Retrieve

Object retrieval will be required every time an event is received. The VE logic will retrieve the object state from memory, which is part of state management. Object states, other than the one being altered, might also be required by the logic to determine the effect an event will have, as discussed in Section 2.2.3.

4.1.3 Modify

Object modification occurs every time an object update is generated. An object update, by definition, required a modification of the object state.

4.1.4 Remove

Object removal might also be required to save storage space, although this is not essential to the correct functioning of the storage system.

4.2 Designing for the storage requirements

Pithos is designed to fulfill all use case requirements as well as the storage requirements for P2P MMVE storage architectures as set out in Section 3.3. To achieve both these goals, an architecture was first designed to fulfill all the storage requirements and the use case requirements were then implemented on the designed network.

The inspiration for Pithos come from two observations:

1. One can combine multiple storage models and arrive at a model which possesses fewer disadvantages than any of the models used.

2. Responsiveness is greatly increased in a fully distributed model, where there is no intermediate server that relays all information.

Fully distributed architectures are, however, not scalable because the number of messages scaling by $O(N^2)$, where N is the number of nodes in the network. We, therefore, use groups of peers, where all the peers in a group are fully connected.

This section will describe how the design of Pithos satisfies the identified storage requirements. For each of the requirements of fairness, reliability, responsiveness, scalability and security, the design decisions made to achieve the specific requirement will be discussed. Some design decisions satisfy multiple requirements, therefore different requirements might discuss the same design decision, with a focus on how the decision satisfies the specific requirement.

4.2.1 Responsiveness

In Pithos, responsiveness is achieved by grouping users into fully connected groups, using group-based distance-based storage to distribute objects and using replication to enable the parallel retrieval of objects.

4.2.1.1 Group storage

In a fully connected network, where all nodes are connected to all other nodes, every node is one hop away from every other node. This solution is, however not scalable. In order to achieve a scalable single hop architecture, it was decided to group users in the virtual world. User groups are then fully connected, which allows for highly responsive group interactions. Group sizes are smaller than the overall network size and allows for the fully connected groups to remain scalable.

In virtual worlds, users already group themselves into explicit groups, including questing groups, raid groups and guilds. In Section 3.1 of [115], it was found that most users in Second Life form small groups. Implicit groups are also formed by

flocking, where multiple users congregate in areas of the virtual environment that are of interest to them [25].

4.2.1.2 Distance-based storage

Grouping users is not sufficient to achieve a responsive system. Users should also require data stored in the group more than data stored outside the group. To this end, distance based storage is used on a group level. This means that objects are stored in the group that is closest to them. The assumption is that users interact more with objects that are closer to them and, therefore, have more interest in closer objects. If a user has interest in a far off object, she will likely move closer to that object.

4.2.1.3 Replication

Replication improves retrieval responsiveness, if multiple copies of an object may be requested in parallel and the object that arrives fastest is used. This is termed parallel retrieval in the Pithos design and is evaluated in Chapter 6.

Depending on how storage is implemented, replication can also improve storage responsiveness if the first object that is successfully stored signals success to the higher layer. This is termed "fast storage" in the Pithos design, the suitability of which is evaluated in Chapter 6.

4.2.2 Reliability

Reliability is achieved by storing all objects on overlay storage (DHT), in addition to group storage, replicating all stored objects and repairing object replicas as they are removed due to network churn.

4.2.2.1 Overlay storage

Distance-based group storage attempts to maximize the number of requests for objects within a group. The number of actual intra-group vs. inter-group requests will still depend on the grouping algorithm and it might not be possible to ensure that all requests remain in the group.

In order for peers to be able to requests objects stored in other groups, overlay storage is used. Peers belong to both a group and the overlay. Peers can therefore request data from the group and from the overlay. Overlay storage adds reliability by ensuring that out-of-group object request can also be served.

Every retrieve request is requested from both the group and the overlay storage. If any of the storage types fail, the other might still succeed, improving reliability.

4.2.2.2 Replication

Replication allows for multiple peers to leave the network before an object becomes unavailable, improving reliability. Because of network churn, replication is essential to ensuring objects survive network churn. If only a single object is stored, a node unexpectedly leaving the network destroys an object. There is no chance of an object being recovered if no replicas are used.

It is, therefore, important that a sufficiently large number of replicas and repair rate be chosen, while taking the network parameters such as expected node lifetime and required TTL into account. The interaction between these parameters are explored in Chapter 7.

4.2.2.3 Repair

Repair further improves reliability by maintaining sufficient object replicas. When it is discovered that a peer has left the network, peers storing the objects of the peer that left can replicate those objects, thereby maintaining a sufficient number of replicas.

If only replication is used, the number of replicas will steadily decline from the moment the object is stored. A repair mechanism is needed to replace missing replicas, ensuring long term object survivability.

4.2.3 Security

Security cannot exist in a single system layer and has to be ensured on multiple layers, as discussed in Section 3.3.5. Some security concepts implemented in Pithos are the use of a certification authority, replication and the use of quorum mechanisms.

4.2.3.1 Certification

A difference between Pithos and other P2P systems is that it requires all peers to be uniquely identifiable. A main requirement of classic P2P storage architectures is that peers remain anonymous to ensure user privacy. In Pithos, all users are unique identifiable. An object records which user created it and which users modified it. This allows for the identification of users that maliciously alter objects.

4.2.3.2 Replication

Storing multiple replicas attempts to limit the effect that a malicious user might have on the storage system. If a malicious user alters an object, the original unaltered object is still stored on multiple peers.

4.2.3.3 Quorum

When security is a concern, multiple parallel retrieval requests can be performed. A quorum mechanism is then used at the receiving peer to compare the different received objects. The object that is returned by most of the peers is then considered to be the correct object. A peer modifying an object has no effect on the system, since the receiving peer selects the object returned by the other peers.

The quorum mechanism will fail if the majority of peers queried colludes to send the same altered object. This can be detected by comparing received objects to objects received from the overlay storage. Ideally, it should only be necessary to perform this comparison if it is suspected that some of the peers in the group are colluding to maliciously alter objects.

4.2.4 Fairness

Fairness is achieved by using both overlay storage and group storage. Both of these schemes require all objects in the network to participate in storage.

4.2.4.1 Group storage

Objects are uniformly distributed on group peers, ensuring that all group peers share the load of group objects stored.

4.2.4.2 Overlay storage

Overlay storage maps objects and peers to the same uniform key space and stores an object on a peer that is the closest match for the object's key. This uniformly random mapping ensures a uniform distribution of objects in the overlay.

4.3 Key modules and mechanisms

In this section we identify and discuss the key modules and mechanisms that Pithos requires to implement in order to achieve its design requirements.

Figure 4.2 shows the Pithos network topology. The figure shows groups of fully connected peers (light blue) and super peers (dark red). All peers are further connected to a single overlay. Pithos groups peers to form a two tiered storage model. The first tier is a storage model at group level (group storage) and the second is a storage model over all groups (overlay storage). Pithos can be considered as a multi-tiered structured overlay, with group storage an implementation of an $O(1)$ overlay and overlay storage an implementation of an $O(\log(N))$ overlay.

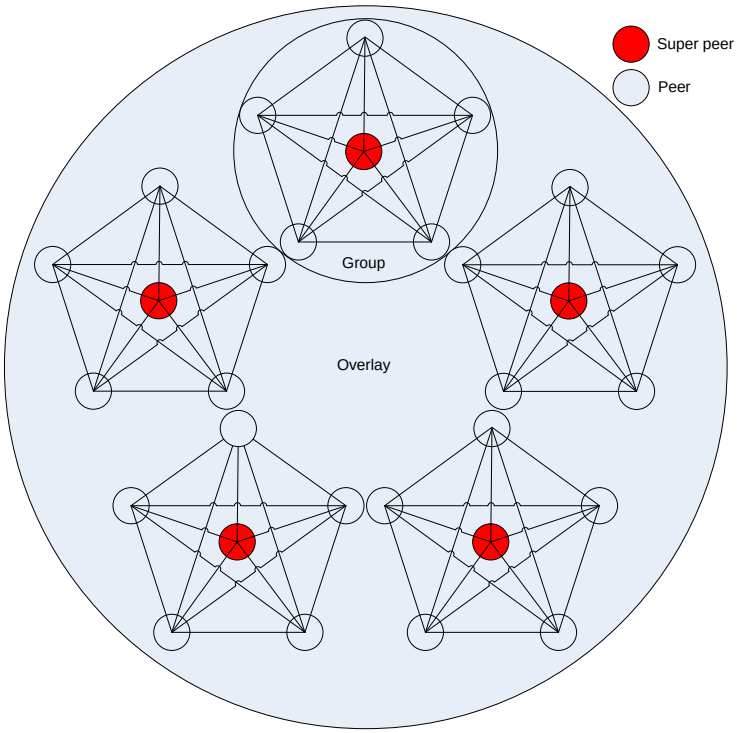


Figure 4.2: Layout of the Pithos storage architecture

When referring to state management and persistency as defined in Section 3.1, Pithos is designed to serve both requirements. Group storage is designed for state management and overlay storage for state persistency. Since state persistency does not require responsive storage, the overlay section of Pithos is well suited to this task, while group storage, with its high responsiveness is well suited to state management.

4.3.1 Group storage module

On a network level, group storage consists out of two node types: super peers and peers. It is possible for a single Pithos terminal to be both a peer and a super peer.

Since group storage can be seen as an $O(1)$ structured overlay, it requires the features identified in Section 1.3.4: network topology, routing, network join and leave and bootstrapping. Additionally to the required routing features, group migration should also be possible (moving from one group to another) and records of all objects stored and all peers present are also required. Lastly, some method of determining group membership is required.

4.3.1.1 Network topology and routing

The network topology is fully connected and routing in such a network topology is trivial. A peer's routing table contains a list of all peers in the group. To route a message to a destination peer, the peer address is retrieved from the routing table and a message is sent directly to that peer.

4.3.1.2 Super peers and peers

In group storage, peers handle requests from the higher application layers. Peers also represent users in group storage, which means that peers are the originators of all store, retrieve, modify and remove requests from a group perspective. The peers themselves receive those requests from the higher layer (such as a game client).

Super peers are responsible for representing a group, managing group membership and managing object repair. The design of Pithos attempted to minimise super peer load, to prevent the functionality of the super peer to adversely affect the peer. The super peer does, however, require extra resources from the underlying system. It would, therefore, be preferable to select peers that have sufficient additional resources to become super peers.

The question of super peer selection has not yet received sufficient research attention. The authors in [105] discuss the need for a utility function that is able to calculate the appropriateness of a peer to become a super peer. The utility function takes certain system parameters into account. The issue is that many of the identified parameters cannot currently be reliably measured, including for example: user reputation, bandwidth and delay. The super peer selection problem is, therefore, still an open research problem.

For the current Pithos design, it is assumed that some utility function exists to select the most appropriate peer and to promote that peer to super peer status.

4.3.1.3 Join and bootstrapping mechanisms

Peers should be able to join the P2P network and join a group in the P2P network. The Pithos design specifies that some well known entry point into the P2P network should exist that a peer can use for bootstrapping. For an MMVE, it is recommended that this be a peer or a set of peers that is controlled by the VE operator. It is understood that this design breaks away from a pure P2P design, but a pure P2P design is not always the best design in practice.

It is assumed that VE operators want to control access to the VE. The operator controlled bootstrapping mechanism will aid in that goal. If the bootstrapping mechanism is fully distributed, operators might have no way to control access to

their VEs, which would simplify the task of pirating the VE and creating a separate free VE.

A well known entry point into the network also simplifies the network design.

For a peer to join a group, it is assumed that some distributed grouping mechanism would exist. Super peer might form part of their own virtual location aware overlay network. A joining peer can then be passed from super peer to neighbouring super peer, until an appropriate peer accepts the peer's join request.

Peer joining will also depend on the specific grouping algorithm used.

4.3.1.4 Grouping mechanism

In order for Pithos to build groups, a clustering mechanism is required. There already exists a wealth of clustering algorithms in literature and a few will be reviewed here.

Distributed peer clustering techniques: use the distance between users in the virtual environment to dynamically group users. The main idea of flocking is that users move around in groups, rather than randomly on their own. It is desirable that user density within groups should remain constant, because a fully distributed architecture is not scalable. This means that groups should merge or split as the user density within them change. It might also be required to have mobile groups, where a group has a velocity as well as a position, to be able to uniquely identify groups, even with users joining and leaving and the group moving through the virtual world.

Affinity propagation clusters nodes using a similarity matrix to find similar nodes [41]. The similarity matrix may contain user positions. In this case, affinity propagation will group nodes depending on their location in a virtual world. This algorithm might be well suited to P2P applications, since it is a distributed clustering algorithm based on message passing.

Dynamic Voronoi regioning: divides the virtual world into regions that can be resized or further divided to maintain constant user densities across regions. VSO [60], [59] creates dynamic regions based on a Voronoi overlay network [2]. Near constant user density is achieved by increasing and decreasing the area sizes. This system is based on VON, a distributed Voronoi overlay network designed for MMVEs [58].

Clustering is also used in mobile ad-hoc networks (MANETs) for routing purposes. Mobile-aware clustering algorithms used in MANETs, might also be applicable to P2P MMVEs [125]. Mobile-aware clustering uses relative velocities of mobile nodes to define clusters containing nodes with low relative velocity to each other.

Any of the above mentioned techniques might be used to achieve distributed clustering in P2P MMVEs. The focus of this work is, however, not on developing

a novel clustering algorithm for P2P MMVEs, therefore we have chosen to use a super peer centered distance-based grouping approach for group clustering in our implementation. The implemented joining and grouping algorithms will be discussed in more detail in Section 5.3.1.

4.3.1.5 Leave mechanism

If a peer leaves the network unexpectedly, it will not have the opportunity to inform its group. With no mechanism to detect a peer unexpectedly leaving, group inconsistency will occur. The peer that left might be selected to store an object or might be required to provide an object from storage. These requests will all fail.

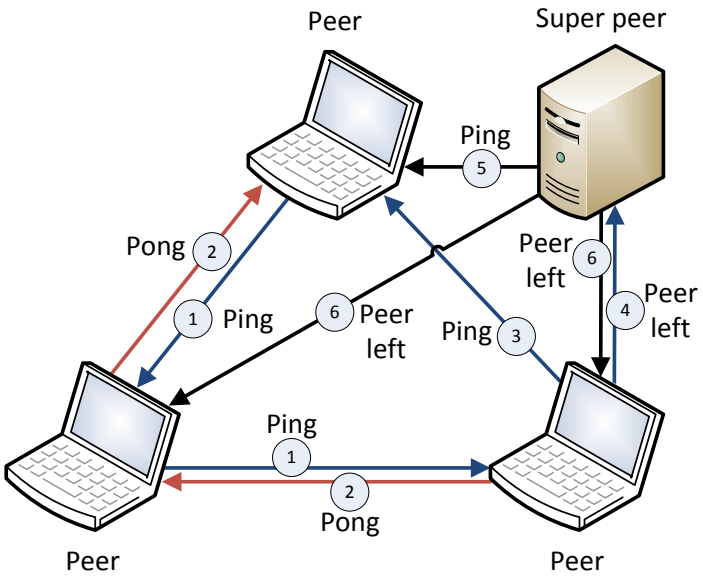


Figure 4.3: Pithos group leave mechanism. Two of the three peers respond to a ping request, while a single peer does not. That peer is removed from the group by the super peer.

In Pithos, two methods exist to ensure group consistency and handle peers leaving the group. The first method, shown in Figure 4.3, uses periodic keep-alive messages as follows:

1. At regular intervals, each group peer uniformly randomly selects another peer in the group to send a keep-alive (Ping) message to.

2. If a peer received a keep-alive message, it responds with an acknowledge (Pong) message.

3. A peer that does not respond with an acknowledge message within a sufficiently large amount of time is considered to have left the network.

4. The originating peer of the keep alive message informs the super peer that the target of the keep-alive message is thought to have left the group.

5. The super peer then also sends a keep-alive message to the target peer, to ensure that there does not exist some communications issue between the two group peers, and that the target peer has actually left the network.

6. If the target peer does not respond to the super peer, the super peer informs all group peers that the target peer has left the group.

The super peer verifies that the peer has left, because by design the super peer is selected to be the most available peer in the group on the network. The super peer does not transmit all keep-alive messages in order to prevent overloading of the super peer.

It is accepted that not all group peers might receive keep-alive messages every round, because of the random selection mechanisms, but this is not required. Only that in a sufficiently small number of keep-alive intervals, the peer is found to have left.

The second mechanism uses requests to identify peers that have left the group. Whenever a request is sent to another peer, a timeout is also started. If a peer does not respond to a request within a given amount of time, the same leave mechanism will be activated as if the target peer has not responded to a keep-alive message.

It should be noted that either of these methods are sufficient to ensure group consistency over some time period, but using both minimises the time it takes to detect a peer leaving the group.

For graceful group leaves, the same leave mechanisms as used in group migration is employed, which is to be discussed in the next section.

4.3.1.6 Migration mechanism

Users will constantly be moving in the virtual environment, leading to changes in position. These position changes may lead to users moving from one group to another. It is imperative that all objects stored in Pithos remain available even with users traversing groups.

Group migration involves informing the group that the peer is leaving and initiating the group join mechanism. Informing the group that the peer is leaving ensures that this peer will not be picked for any storage or retrieval requests, while the group is unaware that the peer has left. If this is not done, the group will only

know that a peer has left after a timeout for a request to the peer expires, which reduces request success rates.

There is a time window between a peer leaving a group and all peers being informed of the migration. During this time, a peer that has migrated might receive requests from its original group. The peer will serve these requests if able and include a group ID in its response. If a peer receives a response from another group, the requesting peer removes the responding peer from its group list. Inter-group requests are not currently allowed due to scalability concerns as time passes and peers continue to migrate.

The current implementation of the group migration mechanism is based on a centralised approach discussed in Section 5.3.2.

4.3.1.7 Ledgers and object store modules

The previous sections have focussed on group storage, purely as an $O(1)$ structured overlay. To enable the storage and retrieval of objects within the group, an object store is also required, as well as a way to track which objects are stored on which peers and conversely, which peers contain which objects.

Each peer in the group contains a local authoritative object store, containing all objects stored on the peer by group storage. Each object is stored with an associated time-to-life (TTL) to prevent stale object copies.

Each peer and super peer in the network contain a group ledger to keep track of which objects are stored on which peers. The group ledger is required whenever an object is retrieved within the group to know from which peer an object can be requested. The ledger is also used by the super peer, to determine which objects should be repaired.

4.3.1.8 Distance-based storage mechanism

For Pithos to succeed as an MMVE storage architecture, intra-group data requests should be preferred to inter-group data requests. This requirement, combined with the fact that the grouping algorithm geographically groups users in the virtual world, lends Pithos to a storage system based on distance-based storage. Similar to interest management, the assumption is that users have a limited area of interest and require interaction with a limited number of objects within range.

Distance-based storage is implemented on a group level rather than an individual level, which means that objects are stored on the nearest group of users, rather than the nearest user. It is assumed that such an approach will alleviate the security and reliability challenges present in distance-based storage [51].

With group-based distance-based storage, it is assumed that because peers now store objects closest to the group, the objects that they are interested in will most likely be stored within their own group. Therefore, most data requests should be intra-group requests. The overlay storage component ensures that nodes that require data, which are not stored within their group, are still able to access requested data.

4.3.2 Overlay storage (DHT) module

The function of overlay storage is similar to that of group storage. It provides storage capabilities as a backup to group storage to improve reliability. Various overlay storage architectures exist, as discussed in Section 3.4.2. Any existing overlay storage architecture can be used in Pithos.

The requirements of overlay storage is that is should be reliable, bandwidth efficient and able to handle network churn. It is assumed that a $O(\log(N))$ overlay is used. For the overlay storage, responsiveness is unimportant, since group storage ensures responsiveness. Reliability is the most important aspect of overlay storage.

An example of an overly storage that might be used in Pithos is PAST [36], based on the Pastry overlay [90].

4.3.3 Certification mechanism

In order to design a secure distributed storage system, one requirement for the P2P overlay is that peers should not be able to select their own IDs or it will not be possible to secure the system against attack. Peer IDs should rather be assigned securely by some certification authority [20].

To meet this requirement, Pithos implements its own certification authority to assign peer IDs securely and promote security in the P2P overlay. A certification server exists that handle ID requests from peers. The server assigns IDs to peers and provides the peer with a signed certificate that it may use to store data.

Whenever an object is stored or updated in the storage network, peers have to sign the object to enable the tracking of object changes throughout the life of the object. This system is different from classic distributed file storage designs that advocate anonymity in storage. The fact that all changes can be tracked to a specific peer will simplify the task of eliminating user cheating.

4.3.4 Quorum mechanism

A security mechanism that performs object verification is also used. When this mechanism is activated, the objects received from multiple retrieve responses are compared to ensure object correctness.

Object verification is an attempt to make Pithos more resistant to malicious peers that attempt to alter data in a way that is not consistent with the environmental logic. When multiple retrieve responses are received, all objects are compared and the object that occurs the most from all the responses is sent to the higher layer.

4.3.5 Replication mechanism

When storing objects in Pithos, replication is used to increase object availability under network churn and for security in the presence of malicious peers [94]. For every object that is stored in Pithos, R object replicas are also stored. The number of replicas (R) depends on the degree of network churn as well as the number of expected malicious users in the network. If the network churn is high, more replicas are required to avoid the situation where all R peers hosting an object leaves the network before any object migration can be done.

If a peer leaves the network and stops to transmit "keep alive" messages, the repair mechanism will detect this and replicate the file on another peer. Replication exists on group as well as overlay storage and is required for ensuring that if a peers leaves the network, the objects that it stores are not lost.

In overlay storage, when the target of a retrieval request leaves the network, the request is routed to the peer with the next closest ID, because of the distance-based routing scheme used in overlay storage. This feature of overlay routing can be exploited by storing object replicas at peers with IDs that are close to the main peer selected to store the object. If a peer leaves the overlay, the new destination peers will possess the stored files, since overlay storage stores overlay replicas at overlay neighbours.

Group replication will be described in detail when the storage mechanism is described in Section 4.4.1.

Another reason to replicate objects is to make the system more secure. If it is known that a certain percentage of users are malicious, it is beneficial to have more replicas than malicious users. This will allow for a secure system where object hashes can be compared to determine which peers are malicious and what version of an object is accurate.

4.3.6 Repair mechanism

When objects are stored, as described in Section 4.4.1, multiple replicas of the same object is stored within a group. This form of redundancy extends the lifetime of an object, but with the presence of constant network churn, all objects will eventually cease to exist.

The solution is to use a repair mechanism to ensure that missing object replicas are constantly replaced. Two types of repair mechanisms were implemented: periodic repair and leaving repair.

With periodic repair, the super peer periodically checks the number of available replicas of every object in its group ledger. If an object contains less than the required number of replicas and there are peers in the group that do not already store the object, that object is replicated. Because the super peer itself contains no objects, it requests that an object be replicated from a peer that does contain the object. That peer will receive a repair request for a specific object, select a group peer in a uniformly random fashion, that does not already contain the object, and initiate a store request to that peer.

Object repair is not lossless. A peer can be selected to store a new object replica and that peer can leave the group before it stores the object. Because many object replicas exist and the loss of a single object will, therefore, not endanger the life of an object, the extra bandwidth and maintenance requirements could not be justified. Sometimes there are multiple object replicas missing, which is solved by having the super peer request multiple object repairs.

Since objects are only destroyed when a peer leaves the group, with leaving repair, the super peer repairs all objects of a peer that leaves the group. This mechanism can be seen as an extension of the group leave mechanism, depicted in Figure 4.3. Figure 4.4 shows how this mechanisms is extended to include repair. Steps 1 to 6 are the same as for the leave mechanism.

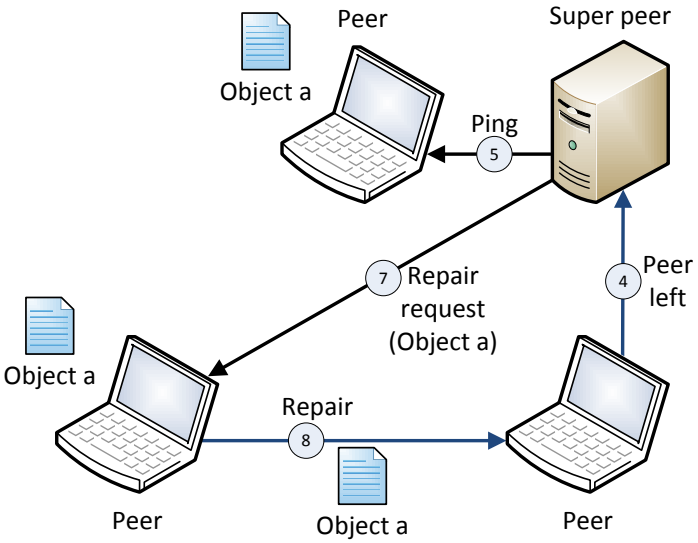


Figure 4.4: Pithos repair mechanism. The super peer requests that Object a be replicated on the only peer in the group that does not yet store it.

7. After a the super peer has detected that a peer has left and has informed all other peers, the super peer selects a peer that contains an object that was destroyed by the leaving peer and requests that that object be replicated.

8. The peer receiving the replication request from the super peer selects another peer in the group that does not already contain the object and replicates the object on the selected peer.

Leaving repair attempts to minimise the super peer load by only repairing objects when a node leaves the network. This is as opposed to periodic repair, where the super peer has to constantly cycle though its list of object replicas and verify the number of replicas currently stored in the system.

4.4 Satisfying the use cases

All Pithos modules and mechanisms that ensure adherence to the storage requirements have now been described. What remains to be described is how the use cases presented in Section 4.1 are designed on the architecture that adheres to the storage requirements.

It is assumed that some higher layer exists above Pithos, which generates requests. According to our generic consistency model, these requests will arrive from the environment logic or object updater. It should be noted that a request can be a storage, retrieval, modification or removal request and does not only imply a request for data, i.e. a retrieval request.

4.4.1 Store

Referring to Figure 4.5, Pithos storage works as follows:

1. The higher layer is expected to send a storage request to the Pithos module on the source node, containing the object to be stored.

2. Pithos contains logic that relays the storage request to peers in both group storage and overlay storage.

3. The group storage module will store the object in its local authoritative object store and acknowledge the reception of the object by sending a response message to the source node.

4. Each peer successfully storing a received object will inform all group peers, as well as the super peer, of the new object stored. The peers and super peer can add this information to their group ledger, used to service retrieval requests.

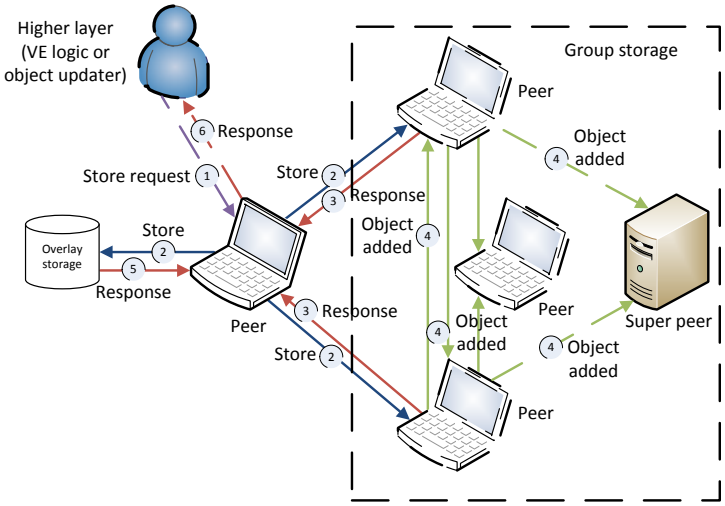


Figure 4.5: Pithos storage. The higher layer requests that an object be stored. The peer stores the object both in overlay storage and group storage.

5. Overlay storage also completes the request and informs the source node of the success or failure of the request, by sending a response message.

6. The Pithos module on the source node monitors all responses and records the success or failure of each request. When a sufficient number of responses have been recorded, Pithos informs the higher layer of either a successful or failed store.

Overlay storage implements an existing DHT and it is assumed that overlay storage correctly handles the storage request.

The number of nodes selected from the group ledger for storage relates to the number of required object replicas. Nodes are selected in a uniformly random fashion, making sure to never select the same node more than once. If a node is selected more than once, it reduces the effective number of replicas in the network, which reduces the expected object lifetime. [1]

The question arises as to what constitutes a sufficient number of responses. Two types of storage mechanisms have been implemented in Pithos: "safe" and "fast". When safe storage is used, Pithos waits for all responses to be received. Only after all responses are received is a decision made whether the store was successful or not. If the majority of group stores as well as the overlay store are successful, is the request considered to be a successful store. The higher layer is then informed of the status of the store request.

[1] A detailed discussion of object lifetime is presented in Chapter 7.

When fast storage is used, Pithos only waits for the first successful response, before informing the higher layer of success. Only if all responses are failures is the higher layer informed of a failure.

Because safe storage ensures that most of the replications are successfully stored, there is a smaller chance of it reporting a false positive than for fast storage. The disadvantage is that from the point of view of the higher layer, and by extension any system that uses Pithos, safe storage requests have a lower responsiveness. Fast storage on the other hand is no less reliable than safe storage, but the chances of a false positive being reported to the higher layer is greater. The advantage of fast storage is that the from the perspective of the module using Pithos, it is performed faster than safe storage. A comparison of fast and safe storage is performed in Chapter 6.

4.4.2 Retrieve

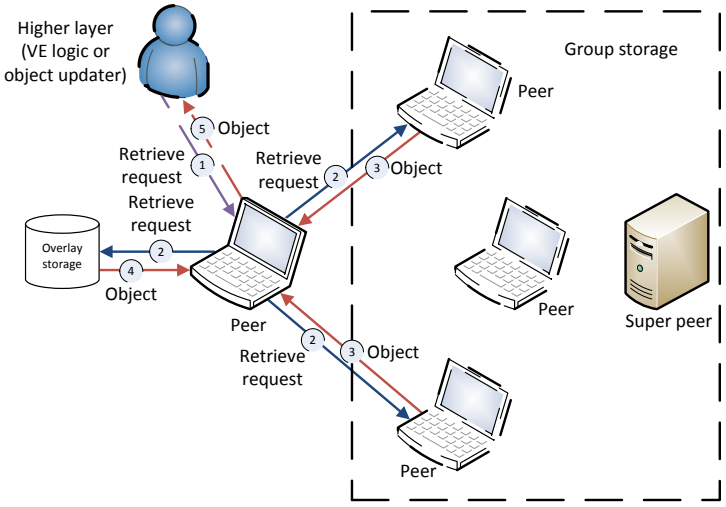


Figure 4.6: Pithos retrieval. The higher layer requests that an object be retrieved. The peer requests the object both from overlay storage and group storage.

The process of retrieving object data is similar to that of storing object data. With reference to Figure 4.6, retrieval works as follows:

1. The retrieve request, received from the higher layer, contains the hash of the object name which the higher layer requires.

2. Pithos first checks whether the object that was requested exists within the group by checking its group ledger. This is required, because the higher layer might have requested an object that is stored in another group. Out-of-group

requests are only serviced by overlay storage. If the object is housed in the Pithos peer's local store, the object is sent to the higher layer. If the object is not housed locally, Pithos requests the object from both overlay and group storage.

The group ledger is used to determine which group peers contain which objects. The number of peers selected depends on the retrieval type. Three retrieval types exist: fast, parallel and safe retrieval. Fast retrieval selects a single peer for object retrieval. Parallel and safe retrieval select multiple peers to concurrently retrieve an object from. More requests sent in parallel improves the probability of the request succeeding as well as reducing the time it takes to serve the request.

3. When the retrieval request is received at the destination peer, the object is retrieved from local storage, attached to a response message and sent to the requesting peer.

4. Sometime after group retrieval has completed, overlay retrieval will also complete and send the object, stored in the overlay, to Pithos.

5. After Pithos has received a sufficient number of responses, it informs the higher layer of either a successful or failed retrieval. The object itself is attached to a successful request response.

Retrieve requests mainly fail because of nodes leaving a group. If a node leaves before it can serve an object, the object request fails. If multiple requests are sent with parallel retrieval, it improves the chances of contacting a node that is not about to leave, which improves the probability of retrieval success. More requests also improve lookup times, since some peers might be geographically closer to the requesting peer, thereby possessing a lower latency.

Sending multiple requests improves the chances of contacting a peer that is closer, which improves the responsiveness of the systems. Fast and parallel retrieval sends the first successful response, and therefore, the first successful object, to the higher layer. It replies with failure if all requests failed. The disadvantage is that it is not resistant to malicious peers.

The safe retrieval request is more resistent to malicious nodes. For safe retrieval, Pithos waits for a specified number of responses in order to use the quorum mechanism described in Section 4.3.4.

As shown in Figure 4.6, the super peer is not used for retrieval requests. This is not necessary, since all peers contain lists of all objects. The super peer should, however, still contain its own group ledger, to inform peers joining the group of the

peers and objects already in the group. Not requiring the super peer to manage object requests prevents the super peer from becoming overloaded.

4.4.3 Modify

Modifying data is similar to storing data. A request to modify a specific object is received from the higher layer. The request specifies the object ID, the parameter that has to be modified and the new value of the parameter. The update is delivered to Pithos, which forwards the request to both group and overlay storage. The overlay handles the modify request and responds with success or failure. Group storage received the modify request and checks the group ledger whether the object exists within the group. If the object does not exist in the group, it is only modified in the overlay. To keep track of modified objects, each object has a version number that is incremented every time an object is modified. This allows retrieve requests to select the object with the latest version number to be sent to the higher layer.

If the object exists within the group, all peers that contain the object are identified from the group ledger and modify requests are sent to them. The modify request is sent to the destination peer, where the object in the local store is updated and its version number incremented. If a retrieve request receives objects with multiple version numbers, it selects the set of objects with the latest version number for "safe" comparison.

4.4.4 Remove

Removal is handled by the object TTL. It is not possible to define an explicit remove operation for overlay storage, since a peer may be off-line when the removal request is sent. If the peer rejoins the network, the object becomes available again. Object TTL ensures that the storage space requirements does not increase indefinitely during the lifetime of the storage system.

4.5 Conclusion

This section describes the Pithos conceptual design. Pithos is a hierarchical distributed storage system that uses grouping to reduce latencies of store and retrieve requests, as is required by massively multi-user virtual environments (MMVEs). The use cases are defined from the perspective of the higher layer that wishes to make use of Pithos's services.

The conceptual design is first described in terms of how it ensures the storage requirements are met. Each module and mechanism that contributes to the achievement of the identified requirements are then discussed. The implementation

of the use cases is described with a discussion of how they relate to the modules and mechanisms required to ensure the consistency requirements are met.

Pithos is designed as a multi-tiered distributed storage system, containing multiple $O(1)$ group stores and a single $O(\log(N)$ overlay store. Each node is connected to the overlay storage and grouped into group storage. A certification mechanisms ensures store and retrieve requests can be tracked and that malicious users can be identified. It contains a quorum mechanism that rejects maliciously altered objects. To maintain reliability, a replication redundance mechanism with repair is used.

Group storage has a fully connected network topology, containing peers and super peers. Ledgers allow each peer in a group to keep track of all other group peers and all objects available on the group peers. Distance-based storage is used to ensure that retrieval requests for objects in the VE are mostly within the same group.

Every storage, retrieval and modification request is sent to both overlay and group storage. Group storage makes use of the group ledgers to find object locations for retrieval and possible peers for storage. Removal is ensured by the association of a TTL with all objects.

Chapter 5

Pithos Implementation

The previous chapter discussed the conceptual design of Pithos. Pithos has been implemented in Oversim [7], a P2P simulation environment based in Omnet++ [113], which allows for the measurement of identified requirements. Furthermore, it allows for the comparison of the current model with other state persistency models available in Oversim.

5.1 Oversim

Oversim is a peer-to-peer and overlay simulation framework for the Omnet++ simulator. Oversim allows for the simulation of many well known structured or unstructured overlay protocols. It also allows for the development of applications that can use the already implemented overlays in a well defined architecture.

5.1.1 Motivation

To allow for greater control of the environment, as well as greater scale, it was decided to implement the first version of Pithos as a large scale network simulation. A simulation allows for careful selection of the environment parameters and precise control of the parameter values. This in turns allows one to keep all but one parameter constant and evaluate the effect of a design decision on the system for varying values of the free parameter.

The underlying Omnet++ simulation environment is a powerful network simulator in its own right. It allows for robust message and module definitions and contains many tools to assist with simulation measurement and monitoring, for example global statistics gathering tools and built-in plotting tools.

Simulation also allows for greater scalability and simulating on a network with global Internet characteristics. Greater scalability is achieved, because thousands of

nodes can easily be created, where all the nodes have global scale latency characteristics.

Implementing a large scale network application in the real-world would require thousands of computers spread across the globe. Such an environment will contain many variables out of the designer's control. For example, issues such as router congestion vary from day to day, which will produce varying latency results.

Because of these reasons, it was decided to first implement Pithos in simulation. This allows for the perfection of the design, under laboratory conditions, before the real world implementation is completed.

Pithos is intended for future real world implementation, however, so all modules created were written in such a manner to allow for easy porting. No tasks that are performed in Pithos make use of any simulation "short cuts", such as accessing the global peer list. Only statistics gathering modules are allowed to make use of these simulation features.

5.1.2 Underlay network

At the base of an Oversim simulation is the underlay network. The underlay network determines the types of nodes in the Oversim simulation. Three underlay types exist: the "simple", "INET" and "SingleHost" underlays [6].

A node type is determined by the protocols executing on every layer of the Oversim architecture. As discussed in Section 4.3, the Pithos node types are the peer, super peer and directory server.

For the Pithos simulation, reliable communication was assumed. It was also assumed that nodes have sufficiently large receive buffers to buffer all required messages until they are handled. No message ordering was assumed. Messages are allowed to be received in a different order than the order they were sent in. This, in fact, always happens when a larger message is sent before a smaller message.

The time that a message is scheduled to be received at the destination is the time when the message would have been fully received by the destination.

5.1.2.1 Latency profiles

In the simple underlay, node latencies are determined by the distance between nodes placed in a two dimensional Euclidean space. The positions of the nodes are chosen to match the latencies of the CAIDA/Skitter Internet mapping project [40]. Different nodes are also assigned different bandwidth and jitter parameters to simulate a heterogenous network. The simple underlay, therefore, captures the delay characteristics of a global scale network in an abstract way. The simple underlay network

is ideal for simulating large scale overlay networks because of its simplicity and high accuracy [6].

The INET underlay is based on the Omnet++ INET underlay and allows for the simulation of the complete IP level stack. This includes backbone routers and gateways. It also contains many implemented MAC layer protocols. The INET underlay is well suited to simulating lower level communication protocols or wireless protocols such as IEEE 802.11 (Wi-Fi).

The SingleHost underlay allows for interaction with a physical network. It allows for the implementation of physical nodes, running on networks computers. These physical nodes can then connect to a simulated network and receive packets with bandwidth and delay characteristics as if they were situated on a network with the simulation's specific characteristics.

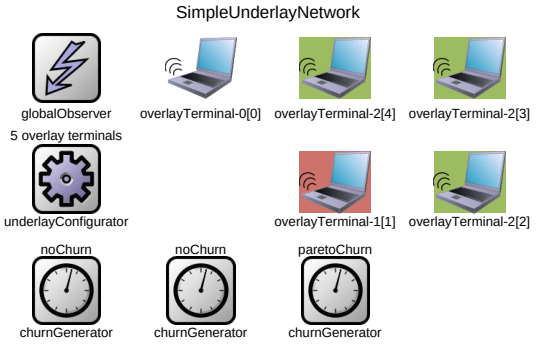


Figure 5.1: The Oversim simple underlay network with the three Pithos terminal types

Figure 5.1 shows the Pithos simple underlay network after the first five nodes have been generated. Each overlay terminal is an Oversim node containing its own protocol stack. The underlay configurator is responsible for setting up the network. The global observer contains the global node list, global statistics and parameters modules. It is used to generate global statistics and by the lower levels to keep track of all nodes.

5.1.2.2 Churn generation

Three churn generators also exist, one for generating each of the three Pithos node types. Churn generators model users joining and leaving the network. There are three types of churn generators: "no churn", "lifetime churn" and "Pareto churn". The "no churn" churn generator creates nodes every specified number of seconds until a specified number of nodes are reached and does nothing further. This generator can be used to test simple networks, where churn is not an issue, or for initial testing of the first prototype.

Lifetime churn creates nodes with specified average lifetimes, sampled from an exponential distribution. This is a distribution regularly used to model lifetimes in reliability engineering. Pareto churn samples node lifetimes from a Pareto distribution (type of heavy-tailed distribution), instead of an exponential distribution. Pareto churn requires mean node lifetime as well as mean node dead time to be specified.

5.1.2.3 Exponential versus Pareto lifetime distributions

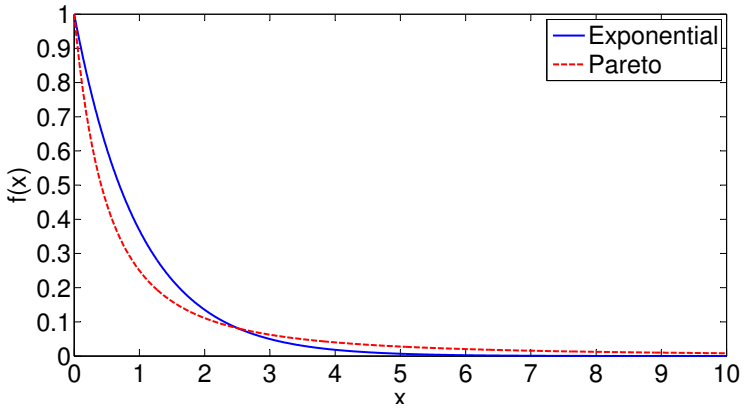


Figure 5.2: Pareto distribution with shape parameter $K = 1$ and scale parameter $\sigma = 1$ and exponential distribution with rate parameter $\lambda = 1$.

Figure 5.2 shows the probability density functions (PDFs) of the two types of lifetime distributions that are regularly used in reliability engineering: the exponential and the Pareto distribution [89]. The Pareto distribution has a shape similar to the exponential distribution, but with a heavy tail. For a shape parameter of $K = 0$, the Pareto distribution is equivalent to the exponential distribution.

The advantage of the exponential distribution is its constant failure rate (hazard rate), which for node lifetimes is equivalent to a constant departure rate. The constant departure rate simplifies statistical models making use of the node failure rate parameter, as will be seen in Chapter 7.

Even though the exponential lifetime distribution is simpler to work with, the Pareto distribution has been shown to more closely match user session times in online games [69]. During the Pithos evaluation, the results from both distributions are compared. The disadvantage of the Pareto distribution is that its failure rate is a function of node lifetime.

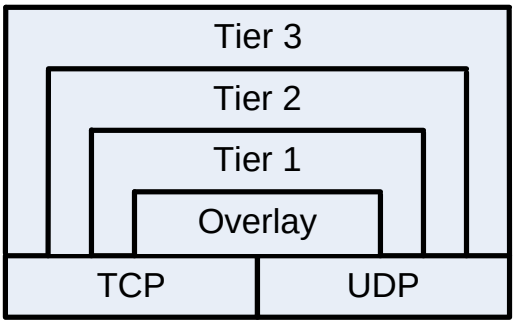


Figure 5.3: Oversim node architecture layers

5.1.3 Node architecture

Every node in Oversim contains a layered architecture, as shown in Figure 5.3. The Oversim node architecture contains the layers: TCP, UDP, Overlay, Tier 1, Tier 2 and Tier 3.

The transport control protocol (TCP) and user datagram protocol (UDP) are both transport level protocols, according to the OSI protocol stack specification. These two layers are the lowest communications layers in Oversim and allows message passing to other nodes.

The overlay layer is where the P2P overlay is housed. This can be any of the well known structured or unstructured overlays, such as Pastry, Chord or Gia [23]. The three tiers above the overlay layer are the application level layers and is where application that use an overlay may be implemented. Three application layers are allowed.

Tier 1 can communicate with the Tier 2, overlay, TCP and UDP layers. Tier 2 can communicate with the Tier 1, Tier 3, TCP and UDP layers. Tier 3 can communicate with the Tier 2, TCP and UDP layers. Internal communications are usually between tiers and TCP and UDP communication occurs when a layer wishes to send a message to the same layer on a different node.

Figure 5.4 shows the complete Oversim node architecture. The tiers, udp, tcp and overlay are connected as explained earlier. UDP is also connected to a bootstrap list module, which assists nodes with joining the UDP network. The neighbourhood cache, interface table, crypto module and notification board modules are also present, but these modules are unused in Pithos.

5.1.4 Generic consistency extension

To allow for the evaluation of Pithos as a module of a larger generic state consistency architecture, we extended the Oversim framework itself to simulate a complete

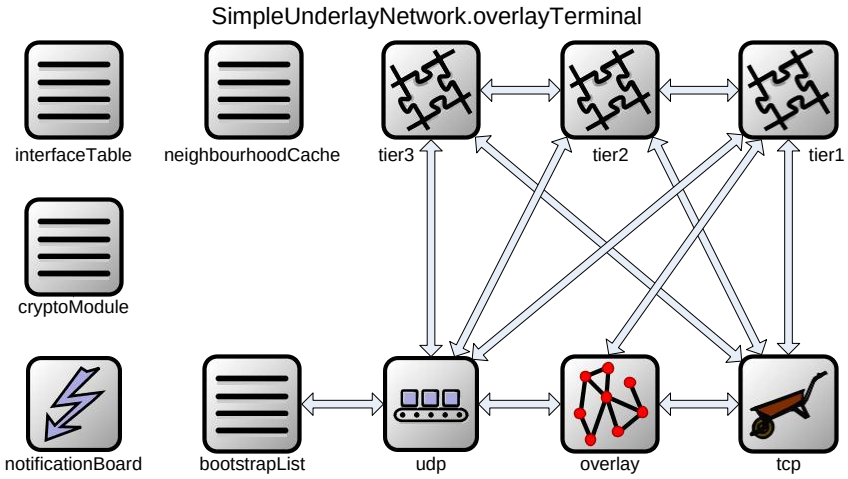


Figure 5.4: Components of an Oversim terminal, including the tiered node architecture.

request stream. The three tiers were replaced with all the elements identified in the generic consistency model in Section 2.3 for a single generic stream. Practically, this required the implementation of a new type of overlay terminal type, replacing the tiers with the layers shown in Figure 5.5.

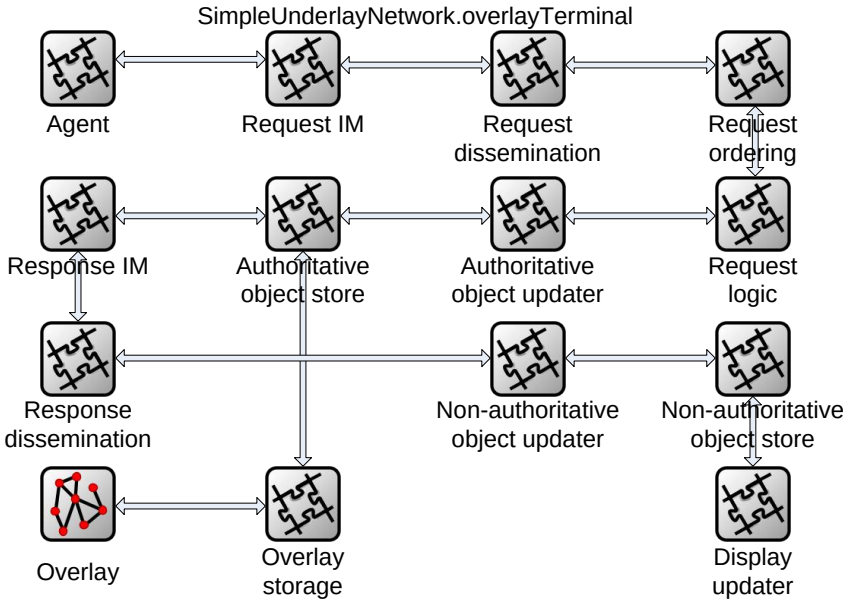


Figure 5.5: Components of the expanded Oversim terminal, including all modules required for the generic consistency model.

Apart from the modules shown in Figure 5.5, all other Oversim terminal modules that were shown in Figure 5.4 are still present (apart from the three tiers) and all of

the modules shown in Figure 5.5 are also connected to the UDP and TCP modules, as with the tiers in the original Oversim module.

Every new layer can communicate with the layer above and below it, and can also communicate with its counterpart on another node using UDP or TCP. This allows for the consistency model to be distributed amongst any number of nodes. A module can either send its information to the layer below it within the same node, or it can send it to the layer below it on another node.

A difference between the generic model and the Oversim implementation is the two modules: "overlay storage" and "overlay". The "overlay storage" module houses the third-party overlay storage implementation and the implementation requires a separate overlay routing module. The separation is, therefore, for practical reasons. Conceptually, both the overlay storage and overlay modules form part of the authoritative object store.

Every module can contain a collection of sub-modules which implements the layer itself. If the layer is not implemented, a dummy layer is specified. The Pithos implementation is located within the authoritative object store module. An application that provides Pithos with test inputs called "PithosTestApp" is situated within the authoritative object updater layer. Pithos also makes use of overlay storage, which is implemented in the overlay storage module. Overlay storage requires a P2P overlay for routing purposes, which is why it is connected to the overlay layer.

5.2 Module types

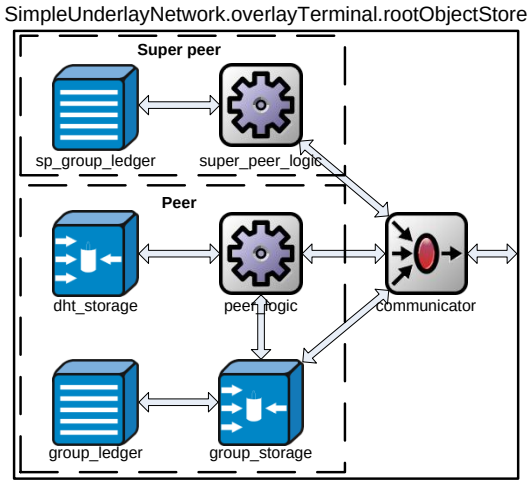


Figure 5.6: Pithos super peer and peer modules in Oversim

A simple module in Omnet++ is a module that implements a C++ class. All modules in Figure 5.6 are simple modules. Compound modules also exist that are combinations of simple modules, where the simple modules are connected to each other and external gates in some way.

Oversim requires a single simple module to be of type: "BaseApp". Oversim delivers messages from the layers above and below a layer to the module extending the BaseApp module. This means that because of Oversim's architecture, all messages must traverse a single module. When Pithos was designed, it was decided that placing all functionality in a single module would be a bad design decision, going against the philosophy of modular design. A single module would not allow for a modular implementation and would make future extensions more difficult.

Therefore, a communicator module was implemented to meet Oversim's requirement of all messages traversing a single module. The super peer and peer functionality is then implemented in seperate modules as shown in Figure 5.6.

5.2.1 Communicator

The communicator module receives and sends all communications to the upper and lower layers on behalf of the other Pithos modules. All messages received or sent are, therefore, routed through the communicator module. The design philosophy of the communicator module was to make it as transparent as possible. Another function of the communicator module was outbound message verification. The communicator module ensures that all sent messages contain a destination IP address, source IP address and packet type information.

Transparency was achieved by designating certain gates for certain types of traffic. If a message is received on a gate with a specific purpose, that message is processed in a way consistent with that purpose and then transmitted. This design allows other modules to be expanded with new message types. Those message then only have to be sent to the correct gate to ensure correct handling of the message. The addition of a new message type, therefore, rarely required alterations to the communicator module.

5.2.2 Peer node

The peer module is responsible for group and overlay storage. It therefore requires a group ledger to keep track of objects in the group. It also needs access to the overlay storage and to implement local group storage. The group peer logic implements all the required group peer mechanisms, which include joining the group, leaving the group, repairing objects and handling store, retrieve and modify requests for the group.

5.2.2.1 Peer logic

Peer logic

1. receives all store, retrieve and modify requests from the higher application layer,
2. forwards those requests to the required modules,
3. forwards responses received from modules to the higher layer,
4. keeps track of all outstanding requests (Section 5.3.3), and
5. implements the quorum mechanism.

5.2.2.2 DHT storage

The DHT storage module forms part of the peer node. The DHT storage module interfaces with the overlay storage module on behalf of the peer. The DHT storage module received requests from the peer logic modules for overlay storage services. The DHT storage module also keeps track of all outstanding overlay storage requests.

In the Oversim simulation, the DHT storage module interfaces with Oversim's DHT implementation, written by Gregoire Menuel and Ingmar Baumgart. The DHT implementation itself is located in the overlay storage module of the extended Oversim node architecture that represents the generic consistency model, as described in Section 5.1.4.

The Oversim DHT implementation contains data structures to store data, route requests via the overlay to other DHT modules and allows peers to join a network-wide overlay. It supports store, retrieve and modify requests. The Oversim DHT implementation meets all Pithos's overlay storage requirements and is able to run on any of the overlay implemented in Pithos.

5.2.2.3 Group storage

The group storage module forms part of the peer node. It receives requests from the peer logic module for group storage services. The group storage module

1. handles group store, retrieve and modify requests from the peer logic,
2. maintains a view of all peers currently in the group,
3. maintains an object store where it stores some subset of the group objects,
4. is responsible for the administration of group membership on behalf of the peer,

5. monitors the availability of group peers and informs the the super peer of peers that left the group,

6. informs the group of new object stored, and

7. repairs objects if requested to do so by the super peer.

5.2.2.4 Group ledgers

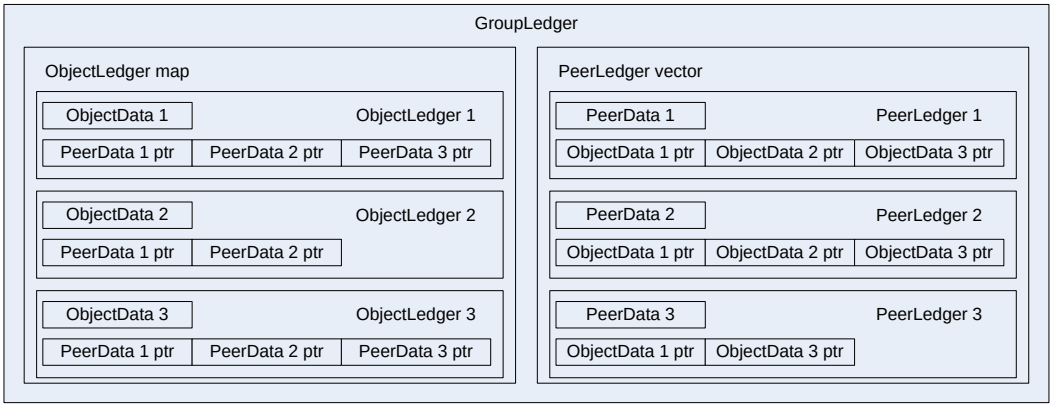


Figure 5.7: Structure of the group ledger in the Pithos implementation.

Figure 5.7 shows the structure of the group ledger implementation. The group ledger is an abstract data type that maintains lists of all objects stored in the group, as well as all peers that are part of the group. The group ledger is used by peers and super peers to keep track of all group objects and peers.

The group ledger was designed for efficient storage, while still allowing high speed access to data. In order for the ledger to be efficient, it implements a map of object ledgers, where each object ledger contains object information and a list of peers where that object is stored on. The group ledger also contains a peer ledger list, where each peer ledger contains a list of objects that are housed on that peer.

In the object ledger, each entry in the peer list is a reference to peer information contained in the peer ledger. In the peer ledger, each entry in the object list is a reference to peer information contained in the object ledger. This means that peer and object information are only stored once, and always referred to by references. This dually linked structure allows for searching in any direction. Objects are stored in a map, since there might be thousands of objects in a group, compared to tens of peers. Efficient object lookup is, therefore, a priority, whereas sequential peer search is sufficient.

The ledger abstract data type is implemented using reference counting pointers (smart pointers), to avoid any memory leaks that might have occurred due to incorrect memory management of the large number of pointers present.

5.2.3 Super peer node

Since super peers are responsible for managing group membership and object repair, the super peer module requires its own super peer ledger to keep track of peers and objects in the group. The super peer logic sub-module implements all the required super peer mechanisms.

5.2.3.1 Super peer logic

The super peer logic module implements all the functionality of the super peer node, except for keeping track of group objects. The tasks of the super peer logic include:

- handling joining peers,
- facilitating peer migration,
- ensuring group consistency (Section 5.3.5),
- initiating object repair, and
- maintaining a list of group objects and a list of peers on which those objects are stored (Section 5.2.2.4).

5.2.3.2 Super peer ledger

The super peer ledger is the same group ledger object used by all peers. The only difference is that it collects group-wide statistics on behalf of all group peers. This is possible, since the information of any group ledger in a specific group is the same.

5.3 Implementation issues relating to key mechanisms

While the previous chapter describes all key mechanisms, some adjustments had to be made to allow for correct functionality under Oversim. The key mechanisms that have simulation specific implementations are also discussed in this section.

5.3.1 Joining the network and grouping

It was decided that the work on Pithos would focus on improving the performance aspects of distributed storage systems, rather than on distributed joining and group-

ing mechanisms. It is accepted that this is work that has to be done, but it will be done as future research.

For the Pithos implementation, a single directory server was used. The directory server places joining peers into groups represented by super peers. The first task of a Pithos peer is to join the network and a group. This is done using the directory server, which possesses a static IP address and port that is known to every new peer and super peer.

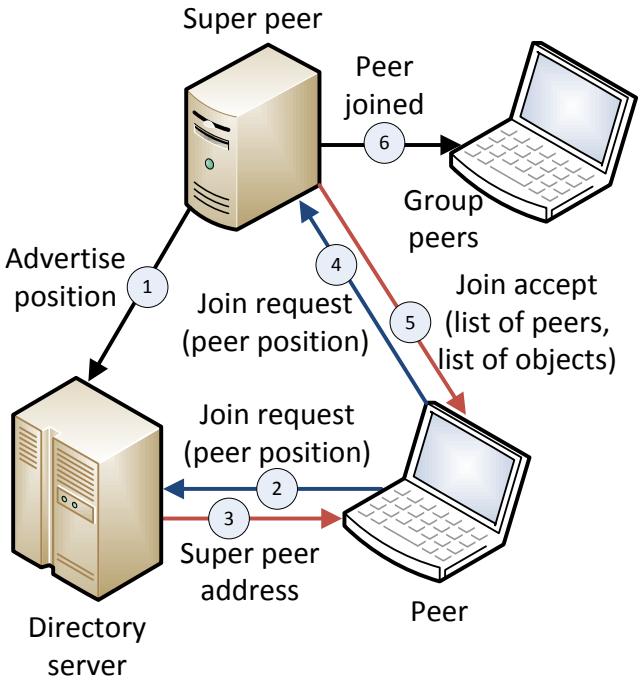


Figure 5.8: Pithos group join mechanism. The super peer publishes its VE location at the directory server, which allows the joining peer to be matched with a super peer and to join that super peer's group.

With reference to Figure 5.8, the join and bootstrapping mechanism works as follows:

1. Whenever a super peer is created or potentially selected, it advertises its information with the directory server. This information includes its IP address, port and virtual location of the super peer in the VE.

2. When a peer wishes to join the network, it sends its current VE location to the directory server.

3. The directory server then responds with the address of a super peer, representing an initial group which the peer can join.

4. The joining peer then sends the same type of join request to the supplied super peer address.

5. If the super peer accepts the joining peer, it sends that peer a list of all peers currently in the group, as well as a list of objects currently available in the group.

6. The super peer also informs all other group peers of the newly joined peer.

The super peer can elect to accept or reject the joining peer. This extra step is intended to prevent groups from becoming overloaded. The super peer can supply an alternative address for the joining peer to contact. This extra step is also intended to compensate for the movement of super peers. If a super peer moves, and by the time a joining peer contacts a super peer there is another super peer in closer range, the super peer receiving the join request can supply the joining peer with the address of the closer super peer.

Because Pithos is not functioning in an actual VE, peers have no locations. For simulation purposes, each super peer is assigned a random location created by sampling both an x and a y coordinate from a uniform random distribution in a fixed range (0 to 100). Joining peers are assigned positions in the same uniform random manner when they are initialised.

These positions are used by the super peers and peers when contacting the directory server. The directory server supplies the peer with the address of the closest super peer to join. This is an implementation of a simple centralised grouping algorithm. Super peers effectively control certain areas in the VE, based on their and their neighbours' locations.

Although the distributed grouping techniques discussed in Section 4.3.1.4 are exchanged for a simple centralised scheme, it serves its purpose to group all peers into groups controlled by super peers for the simulation. In a practical implementation, this solution is still applicable, but the directory server might become overloaded. The distributed grouping and joining mechanism is, therefore, recommended.

5.3.2 Group migration

To simulate the effect that user movement will have on objects stored in Pithos, PithosTestApp is designed to generate position updates.

PithosTestApp simulates a user moving around in a virtual world. This is done by having PithosTestApp generate a new random position chosen in the same way as when a node joined a group: by selected x and y coordinates in a uniformly random fashion. A position update timer determines when a node's position should

change. When a new position is generated by PithosTestApp, this is sent to the group storage module, which then initiates the group migration mechanism.

As avatars move around in the virtual world and PithosTestApp sends position updates to Pithos, Pithos queries the directory server at regular intervals with new avatar positions, using the same mechanism described for joining a group in Section 5.3.1. Directory server query intervals are long enough to prevent overloading of the directory server, but short enough to ensure peers have an up-to-date view of the group. The peer then receives a new super peer address from the directory server, if this address is different from the address of the currently known super peer. When a new address is received, Pithos initiates group migration.

It is accepted that the movement scheme does not match the way users move in a virtual world. Modelling accurate user movement is, however, not considered important for the current Pithos simulation. Storing and retrieving objects in Pithos only depends on which group a user occupies, the group density and how Pithos handles a user that moves to another group. The order in which a user might enter groups or, in fact, the specific group that a user moves to, is not considered important. All that is important is how objects are handled when any group migration occurs and how group density affect the performance.

The effect of user movement traces on the performance of Pithos will be explored as future research.

5.3.3 Tracking requests

When sending a request to a destination peer, the destination peer might leave the group before it can receive the request or respond to it. No response will, therefore, be sent to the source peer, which might cause some requests to remain unresolved. For example, if safe storage is used and the system waits for all responses before informing the higher layer of the request status, the higher layer will never be informed if a destination node has left. A mechanism is required to handle nodes leaving the group.

The solution is for the super peer logic, peer logic and DHT modules to track all requests sent, and link every outstanding request with a timeout. Every one of those modules contains a pending requests list. Each pending request is uniquely identified by the RPC ID received from PithosTestApp, which uniquely identifies a specific request. Each pending request from the higher layer usually generates multiple messages to DHT storage, and multiple nodes in the group via group storage.

Every request sent out by the DHT storage or group storage modules contain the RPC ID of the original request from PithosTestApp. The RPC ID is copied into the response at the destination location. This allows every response to be linked to

the original request that generated it. When a response is received from a peer, the pending request for the RPC ID for that peer is removed from the pending requests list.

Every pending request is also linked to a timeout. If the timeout expires, the pending request is removed and a failure response is generated and sent to the higher layer. This mechanisms ensures that all requests will always receive all expected responses. The timeout are the same timeouts use to detect whether a peer has left the group, as discussed in Section 4.3.1.5.

It is accepted that this mechanism may cause false negatives if there is a delay due to congestion in the network. This situation can be avoided by making the timeout sufficiently long, as will be shown in Section 6.3.2.

5.3.4 Quorum

To test the quorum mechanism, the idea of malicious nodes was added to Pithos. When a node starts, there is some percentage change that it will be designated as a malicious node. Whenever a malicious node receives a retrieve request, it retrieves the requested object, but alters it in some random way. This altered object is then sent to the requesting peer. A flag saying that the object has been altered is also sent, for statistics gathering.

The reliability of Pithos could be investigated for various levels of malicious nodes in the network, as discussed in Section 6.6.

5.3.5 Group consistency

With peers constantly joining and leaving the network and transferring between groups, an issue that arises is group consistency. It is important that all peers within a group share the same view of that group. If this is not the case, some peers might never be chosen to store data and this will decrease the fairness in the system. Another reason is that for the sake of scalability, peers do not service requests from peers outside of their groups. Group inconsistencies might also lead to nodes becoming isolated from any group, which basically disconnects that node from the P2P network. A primary source of inconsistencies was the fact that it takes time to inform all group peers of a peer that left and that it takes time to inform all group peers of a peer that joined.

To ensure group consistency, the following improvements were made to Pithos:

1. The super peer stores the information of the last peer that left and the last peer that joined.

2. Peers store information of the last peer that left.

3. Every time a new peer joins, the super peer informs the joining peer of the last peer that left.

4. Every peer ignores messages from the last peer that left.

5. Timeouts are adjusted as a function of available network bandwidth.

6. Group identification information was added to every request, identifying the group where a messages originates from.

7. If a peer received a response from a peer situated in another group, the peer making the request removes the information of the peer that responded to the request from its group ledger.

For a detailed discussion on group consistency and the reasons for the various solutions, the reader is referred to Appendix A. The appendix is, however, not required for an understanding of the rest of this work.

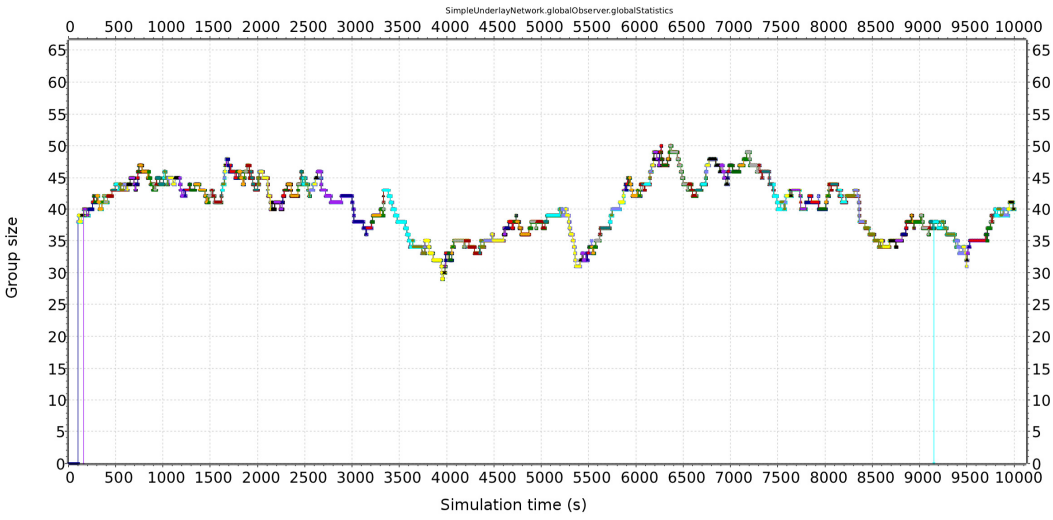


Figure 5.9: Group consistency after all improvements

Figure 5.9 shows the view of group containing 40 peers on average as a function of time. Every peer reports its view of the group size at 1s intervals and those views are overlayed and plotted.

To calculate group consistency, each vector was compared with the super peer's view of the group. For each of the 251 nodes that joined and left the group over the 10,000s simulation time, error vectors were calculated by calculating the difference between the super peer group size vector and the group size vector of every other peer in the group for the duration that a peer was part of the group.

The mean and standard deviations were then calculated for each of these vectors to form two new mean error and standard deviation error vectors. The mean and standard deviation of the mean error vector is 0.0096 and 0.0104 respectively. The mean and standard deviation of the standard deviation vector is 0.0941 and 0.1338.

This data shows that on average there is a 0.0096 peer difference between the number of peers that the super per perceives and the number of peers that the rest of the group peers perceive within every 1s interval. This is considered a sufficiently high level of consistency for the purpose of the Pithos evaluation.

5.4 PithosTestApp implementation

Some application was required that would generate requests for Pithos and collect statistics on how well Pithos services those request. The application that was implemented to perform this duty is PithosTestApp.

PithosTestApp is located one layer above Pithos in the generic consistency architecture, called the object updater. It should be noted that PithosTestApp has no purpose other than to test Pithos. It is therefore supposed to present an aggregation of all the higher layer modules to Pithos.

PithosTestApp has four functions:

1. Generate store, retrieve and modify requests.
2. Correctly handle all responses received form Pithos.
3. Generate position updates.
4. Collect statistics to determine how successfully Pithos handles the requests.

5.4.1 Store requests

PithosTestApp periodically generates store requests of random sizes. The sizes of these requests are based on the literature on how large game objects usually are. This includes an evaluation that was performed on the size of game objects in Ultima Online and the measurements acquired from the Donnybrook architecture [12].

All store requests are sent to Pithos in the lower layer. When a response is received that an object has been successfully stored, the object information is added to a global map, containing all generated objects.

5.4.2 Retrieve requests

The global object map is not available to Pithos and is only used by PithosTestApp to enable object retrieval and for retrieved objects to be verified. Objects in the global object map are grouped by the group they are stored in.

PithosTestApp periodically generates retrieve requests for random objects. An object is selected from the global object map and a request for this object is made to Pithos. The object itself is never supplied to Pithos, other than when it is stored.

PithosTestApp can generate in-group object requests with a given probability. This is to test how well Pithos will function when distance-based storage is implemented to store objects. The assumption is that, with distance-based storage, a high number of retrieval requests will be for nodes within the same group, as opposed to out-of-group nodes.

When a response is received, statistics are recorded on whether the request was a success or failure as well as the reason. If the retrieval is a success, the returned object is compared with the object in the global object store to determine whether the retrieved object's contents physically matches the object contents in the global object store. Verification is mostly used when malicious peers exist and there is a chance that Pithos might send the information of a corrupted object to the higher layer.

5.4.3 Modify requests

PithosTestApp periodically generates modify requests. It selects a random object from the global object map, selects a random variable of that object to modify and assigns a new random value to that parameter. The new parameter value is then encapsulated in a modify request sent to Pithos.

5.4.4 Position updates

When group migration is enabled, PithosTestApp generates regular position updates to allow Pithos to test group migration.

5.5 Conclusion

An overview of the Oversim simulation framework was presented and it was also explained how Oversim was extended to support the generic consistency architecture. Some implementation issues encountered during the Pithos implementation were also discussed. The main purpose of this chapter is to link the conceptual

design with the Oversim simulation implementation of Pithos; to provide concrete explanations on how some of the key mechanisms were implemented.

Chapter 6

Pithos Evaluation

The purpose of this chapter is to verify that Pithos meets the design requirements set out in Chapter 4 and to compare Pithos to overlay storage. Following the discussion of the Pithos design and implementation, the performance of Pithos will now be presented, along with a comparison of Pithos against overlay storage.

The overhead (bandwidth usage), reliability and responsiveness of Pithos will be evaluated together, since the one influences the other as will be shown. The fairness, scalability and security of Pithos will then be evaluated separately.

6.1 Simulation setup

Since Pithos is designed to be a storage solution for P2P MMVEs, it is important that the simulation conditions reflect the network and storage conditions that a typical MMVE experiences. Specifically, the following simulation parameters are important:

- Size of the P2P network (number of peers).
- Length of simulation.
- Underlying physical network.
- Object lifetime distribution (network churn)
- Storage and retrieval rate.
- Choice of P2P overlay.
- Size and time-to-live (TTL) of objects being stored.

6.1.1 Size of the P2P network

For the results shown, 2500 peers, 100 super peers and a single directory server are created at the start of the simulation, simulating a *sufficiently scalable* MMVE, as discussed in Section 3.3.1.

6.1.2 Length of the simulation and storage and retrieval rate

Each simulation runs for 10,000 seconds, which generates a total of 5 million storage and retrieval requests and an average of 600,000 objects. This is thought to be a sufficient number of requests and objects to generate results with high accuracy.

After a node has joined the network, PithosTestApp starts to generate store and retrieve requests at a rate of one objects every 5 seconds. Storage requests are limited to a total window of 20s. The limit of 20s storage request generation time for every new peer joining the network is to reduce the memory required to store all objects. With the current setup, every simulation run requires 6 gigabytes of memory.

The simulation was rerun between 10 and 20 times per generated result to verify and find accurate means for the results. Means between simulation were usually found to match up to three decimal places, which is believed to be as a result of the high number of statistically independent storage and retrieval requests per simulation run.

6.1.3 Underlying physical network

To be able to simulate Pithos for the large numbers of nodes required, as discussed in Section 3.3.1, it runs on the simple underlay. Pithos is dependant on the delay characteristics of the lower level protocols, but the CAIDA and Skitter measurements capture these quantities in their measurements. The reasons why the underlying network shows certain delay characteristics are not important. A focus is also not currently placed on the effects of transient background traffic flows in the Internet architecture on which Pithos executes.

The simulation uses a channel bandwidth of 10 Mbps. Pithos has also been successfully tested for a 1 Mbps link. Results of the 1 Mbps link are similar to the 10 Mbps results. The bandwidth requirements of Pithos is also evaluated in this chapter.

6.1.4 Object lifetime distribution

For each node in the simulation, a lifetime is sampled from some statistical distribution. The two distributions considered were the exponential and Pareto dis-

tributions, which are two distributions that are regularly used in reliability engineering [89]. For most experiments, exponential peer lifetimes were used, since this distribution places a heavier load on the storage system, because of the heavy tail of the Pareto distribution. The load is increased, because if more peers have shorter lifetimes, objects will have shorter expected lifetimes, which decreases the reliability of the system. A heavy-tailed distribution contain more peers that live longer, which increases object lifetime and, therefore reliability.

The Pithos simulation has, however, been successfully evaluated using Pareto lifetime distributions and an experiment is performed in Section 6.5, to show the differences in performance between the two distributions.

Node lifetimes represent how long users spend in the VE. It has been found that these times vary greatly between the type of VE. Mean playing time in WoW, for example, has been measured as 168 min (10080s) [109], with players staying for at least an hour and usually not longer than five hours per session. In Second life, 50% of users stay connected for less than 10 min (600s), 15% stay connected for 100 min or more and 5% of users stay connected for more than 10 hours [116].

Unless otherwise states, peers with 1800s mean lifetimes are used, which was chosen to be somewhere in the mid range of the two games evaluated.

6.1.5 Overlay

In the results shown, Chord was used as the P2P overlay, mainly due to its faster simulation time, compared to Pastry. On a quad core Intel i5 2.66 GHz processor, Pithos using Chord takes 4 hours to simulate and Pithos using Pastry takes 10 hours to simulate. Pithos has, however, also been successfully tested with Pastry and Kademlia and results are similar, as shown in Section 6.2.1.

The specific overlay used is not as important when evaluating the performance of Pithos, since Section 6.4 shows that the performance of overlay storage and group storage are independent and that there exists a linear relationship between overlay storage and group storage as a function of how many in-group requests are sent, when evaluating the overall Pithos performance.

6.1.6 Object size, TTL and replication

An object size of 1024 bytes was chosen to be much larger than Quake 3 game objects without delta encoding, as seen during the development of Donnybrook [12], since Pithos is designed for the low latency storage of small game objects.

Objects have a TTL of 300s and group storage stores six object replicas and retrieves either one, four or six depending on the storage method and experiment. The number of replicas and object TTL was chosen to ensure that objects will remain

in the system with a high probability, without the need for repair. The theoretical work presented in Chapter 7, allowed us to predict expected object lifetimes under the conditions presented and to know that with a TTL of 300s and six replicas, there is a high probability that the objects will be in the system for the full 300s.

The overlay storage module stores 4 object replicas and always requests 4 replicas in parallel, similar to Pithos's parallel retrieval. Overlay storage expects more than half of the requests to be identical, before success is reported.

6.1.7 Measurement of metrics

Some key metrics that will be used to show performance in this chapter are: bandwidth usage, reliability and responsiveness. How these metrics are measured will now be described.

6.1.7.1 Reliability

Reliability is measured as the ratio of successful responses received, to the total number of requests sent. Storage reliability measures the ratio of successful storage requests. Retrieval reliability measures the reliability that an object that was successfully stored can be successfully retrieved. Reliability is, therefore, given by

$$\text{reliability} = \frac{\text{successful responses}}{\text{total requests}} \tag{6.1.1}$$

6.1.7.2 Responsiveness (latency)

Responsiveness is measured as the round trip latency of a request in seconds. In other words, the time it takes, since a request is generated, until a response is received. For the latencies measured in all the results shown, the widest 95% confidence interval for all means measured is 0.3ms, which gives an idea of the precision of the results presented.

6.1.7.3 Bandwidth usage

Bandwidth is measured in bytes per second (Bps) and measures the amount of data transferred per unit of time. In the simulation, a distinction can be made between the data used for group storage and the data used for overlay storage. These two entities are two different modules in the Oversim simulation. Every module measures the amount of data that it sends and received over UDP. This data is used when measuring group and overlay storage bandwidth. It, therefore, includes both the object data as well as any overhead.

To compare overhead, the sizes of objects received from PithosTestApp is used as a metric of usable data entering and exiting the system. This definition of overhead

ignores any packet headers or response message sent to the higher layer that does not contain an object. Overhead is then given by:

$$\text{overhead} = \frac{\text{Pithos UDP bps - object bps}}{\text{Pithos UDP bps}} \tag{6.1.2}$$

6.2 Reliability, responsiveness and bandwidth

Firstly, the reliability and responsiveness of storage and retrieval performance of Pithos and overlay storage are evaluated. The purpose of this section is to establish the baseline of Pithos performance under a basic simulation setup. Therefore, no object repair is performed and no malicious users exist. The network, however, still contains churn.

It is expected that this experiment will show the high reliability and low storage and retrieval responsiveness of Pithos. The reliability of Pithos should be at least the same as overlay storage, while the latency should be significantly lower. This experiment is also used to compare the different storage and retrieval methods implemented in Pithos. The best performing methods will be used in subsequent experiments.

6.2.1 Overlay storage

In this evaluation chapter, Pithos will mainly be compared with pure overlay storage, as well as the different implemented methods of group storage. Overlay storage was chosen as a comparison, because as discussed in Section 3.4.2, many P2P implementations mention using overlay storage as a solution for state management and state persistency.

Overlay storage requires a routing overlay. Three types are overlays are now evaluated: Chord, Pastry and Kademlia. For Chord, three different configurations are evaluated to show how an overlay's performance may be altered by altering its configuration parameters, especially the time between routing table maintenance. For a brief description and comparison of Chord, Pastry and Kademlia, kindly refer to [72].

For the Chord overlay, the three configurations represent high, medium and low levels of reliability and bandwidth requirements respectively. The main parameter modified to achieve these varying levels of responsiveness and reliability is the Chord finger (routing) table update time. The update time determines how up-to-date the Chord finger tables are and by extension, how up to date Chord's view of the network is.

6.2.1.1 Configuration parameters

The configuration parameters of the Chord, Pastry and Kademlia overlays used, are given in Appendix B. For Chord medium, Pastry and Kademlia, the Oversim default values were used.

6.2.1.2 Performance

Table 6.1 shows the reliability, responsiveness and bandwidth of overlay storage for various types of routing overlays. Because of how the settings were defined, high has the highest storage reliability of 98.31%, medium has a storage reliability of 96.9% and low has the lowest storage reliability of 79.08%. High also possesses the higher retrieval reliability of 93.65%, medium has a retrieval reliability of 93.20% and low has the lowest retrieval reliability of 62.16%.

Entity	Reliability (%)		Responsiveness (s)		Bandwidth (Bps)	
	store	retrieve	store	retrieve	in	out
Chord high	98.31	93.65	1.217	1.745	2175	2189
Chord medium	96.9	93.20	1.214	1.582	1183	1197
Chord low	79.08	62.16	1.245	2.071	301	314
Pastry	98.97	94.90	0.625	1.182	1979	2088
Kademlia	45.53	35.13	0.908	4.604	512	498

Table 6.1: Evaluation of overlay storage responsiveness and reliability for various Chord parameter settings.

What is also clear from Table 6.1 is that for low reliability storage, it takes a longer time to retrieve an item, compared to medium or high storage, which further highlights the importance of high overlay reliability.

When comparing Pastry to Chord, the performance of Pastry is higher than that of Chord. This may be seen when comparing the Chord high configuration with Pastry. Pastry has a higher reliability and improved responsiveness, using less bandwidth than Chord. Kademlia has reduced performance, but also required less bandwidth than the two other overlays.

The reliability of the overlays presented here can be adjusted at the cost of bandwidth. An overlay can be made more reliable, but will then require more bandwidth, because increased reliability is achieved by in increasing the rate at which the routing tables are updated.

6.2.2 Pithos performance

The results presented in Tables 6.2 and 6.3 are for a medium overlay storage configuration as described in Section 6.2.1 and where all requests were objects in the group.

Tables 6.2 and 6.3 present the reliability and responsiveness results for the various Pithos storage and retrieval methods discussed in Sections 4.4.1 and 4.4.2. Both safe and fast storage and fast and parallel retrieval are compared. It should be noted that because Pithos is a group/overlay hybrid storage, its performance depends on the underlying group and overlay performance.

It does not make sense to evaluate retrieval performance apart from the storage method used, since the storage method determines whether objects are stored successfully. Therefore, for each storage method in table 6.2, it firstly shows the performance of only the specific storage method. The retrieval performance is then evaluated, taking into account the storage method used.

What will also be shown in this section is that overlay storage requires significantly more bandwidth than Pithos group storage.

6.2.2.1 Storage

Storage method	Reliability (%)	Responsiveness mean (var.) (s)
Safe	97.05	1.554 (3.178)
Fast	100	0.0665 (0.00593)

Table 6.2: Responsiveness and reliability of Pithos's safe and fast storage.

Table 6.2 shows that when fast storage is used, a reliability of 100% is achieved, which is higher then the 97.0% reliability of safe storage. The real reliability is not actually higher, but because fast storage only waits for a single successful response before reporting success, it reports more successful storages. Safe storage, on the other hand, ensures that a majority of replicas were successfully stored before reporting success.

Fast storage will still report success, even if the majority of files were not successfully stored. Fast storage is thus trading reliability for responsiveness. If the underlying network is sufficiently reliable that there is a low probability of any request failing, fast storage is more appropriate than safe storage.

Fast storage can also be extended to retry a failed storage request and to select a new destination for the request. If the failure probability is low, the extension should have a negligible effect on responsiveness, while increasing reliability.

The advantage of fast storage over safe storage is its higher responsiveness of 0.0488s, compared to safe storage's 1.576 s. This means that data are available for retrieval 32 times faster with fast storage compared to safe storage.

6.2.2.2 Retrieval

Storage method	Retrieval method	Reliability (%)	Responsiveness mean (var.) (s)	Group bandwidth (Bps)	
				in	out
Fast	Fast	99.70	0.192 (0.181)	187	183
Safe	Fast	99.77	0.189 (0.160)	183	180
Fast	Parallel	99.98	0.0846 (0.051)	784	735
Safe	Parallel	99.98	0.0859 (0.053)	798	749

Table 6.3: Responsiveness and reliability of fast and parallel retrieval for safe and fast storage.

Table 6.3 shows the results of fast and parallel retrieval for fast and safe storage respectively. Interestingly, fast retrieval for fast storage performs only marginally worse (99.70%) than fast retrieval for safe storage (99.77%). It performs worse, because objects that were incorrectly reported as successfully stored by fast storage, was in fact not stored correctly. Fewer replicas than required might have been stored, which makes the object more sensitive to network churn. If fewer replicas than required are stored, the object will not live as long as other objects that have more replicas.

When fast retrieval attempts to retrieve an object stored with fewer replicas, there is a greater chance that the object has been destroyed, compared to an object that had more replicas initially stored. The request is then unsuccessful. The same is true for parallel retrieval for fast storage, compared to parallel retrieval for safe storage. Safe storage leads to an increase in retrieval reliability at the cost of longer delay before storage success is reported.

Comparing parallel retrieval to fast retrieval (both using fast storage) leads to similar results as the comparison of parallel retrieval to fast retrieval (both using safe storage). Parallel retrieval is both more reliable and responsive than fast retrieval. Parallel retrieval (99.98%) is more reliable than fast retrieval (99.70%), because of more requests being sent, which increases the probability that the request arrives at a node that is not about the leave the network.

The responsiveness of parallel retrieval (0.0876 s) is also higher than fast retrieval (0.196s), because fast retrieval uses the first received response. The responsiveness of a retrieval request depends only on the fastest received response. If more retrieval requests are sent, the expected response time decreases because more links are now

used. In other words, if more links are used, there is a higher probability of at least one link being faster than the others.

It should be noted that the Pithos responsiveness is highly dependant on the underlying network performance. The fast storage, fast retrieval performance of 192ms might seem slow, but is a function of the connectedness and responsiveness of the underlying physical links. Oversim models a global scale network as the underlying physical network, which means that the Pithos responsiveness should be seen in the context of operating in a network, distributed across the world. In a later experiment (Section 6.3.4), the Pithos performance for fixed network responsiveness in a local area network (LAN) environment with 1ms latencies is presented. The experiment shows the average Pithos retrieve latency reduced to 1.6ms.

For subsequent experiments, fast storage has been adopted. The factor 32 increase in responsiveness is seen to outweigh the factor 1.029 decrease in performance.

The question whether fast or parallel retrieval is preferred depends on the implementation environment. The reliability of fast retrieval will suffer if malicious nodes are present in the network, since fast retrieval sends the first received object to the higher layer. This leaves no time to compare objects received from multiple nodes. A fairer comparison of fast retrieval to parallel retrieval must also take bandwidth requirements into consideration.

6.2.3 Bandwidth requirements

Bandwidth requirements have to be taken into account to enable a fair comparison of the different Pithos storage and retrieval methods as well as to compare Pithos with overlay storage. Table 6.3 shows the required bandwidth of Pithos's group storage architecture. Because Pithos is a hybrid storage scheme, its properties depend on the underlying properties of both group and overlay storage modules.

To place the bandwidth usage into perspective, the data sent to Pithos by PithosTestApp and received from Pithos by PithosTestApp should be considered. For the simulation setup as described, 4 bytes per second (Bps) is sent to Pithos and 157 Bps is received from Pithos. The reason why there is less data sent to Pithos, is because of nodes only generating objects for 20s, as discussed in Section 6.1 and retrieval requests being generated during the complete lifetime of a node.

It was found that there is no interaction of group bandwidth requirements with overlay bandwidth requirements and that the two can be evaluated separately. To calculate the total bandwidth required by Pithos, first requires a choice of group storage and retrieval method and then a choice of overlay settings. For the storage and retrieval results shown in Section 6.2.2 a medium overlay storage was used in Pithos. It presented an acceptable reliability, for an acceptable runtime. The

more reliable an overlay, the more routing table updated have to be exchanged, which increased simulation time. Simulating Pithos with the medium overlay takes approximately 4 hours, whereas the high overlay takes approximately 12 hours.

It can be seen that "high" overlay storage still has a lower reliability than group storage, but for higher reliabilities, overlay storage requires prohibitively large amounts of bandwidth, as shown in Section 6.2.3.

6.2.3.1 Group storage

Table 6.3 shows that the bandwidth requirements in bytes per second (Bps) for fast and safe storage are similar and that bandwidth usage largely depends on whether fast of parallel retrieval is used. The reason being that parallel retrieval requires multiple retrieve requests be sent out, which has multiple returned object as an effect. In the results shown, six parallel requests were used to request all six replicas stored.

Six parallel requests lead to a factor four increase in required bandwidth. What is interesting is that it does not lead to a factor six increase in bandwidth as expected. This is a combination of two factors. The first is that a peer might retrieve an object locally and therefore require no bandwidth for that retrieval. The second has to do with the average number of object replicas under network churn. Initially, when an object is stored, six replicas are stored in the group. During the 300s lifetime of the object, some replicas are removed due to nodes leaving the network. Objects in a group, therefore, have an average number of replicas that are less than what was originally stored. Therefore, even though six retrievals are requested, only four replicas on average exist in the network and therefore only four replicas can be retrieved.

It should be noted that this does not influence group reliability, since all peers are aware only of the objects that do exist in the group. Failures can only occur when a peer sends a retrieval request to a target peer, as the target peer leaves the network.

6.2.3.2 Overlay storage

Table 6.1 also shows the large bandwidth requirement of overlay storage, compared to group storage, requiring 1183 Bps inbound and 1197 Bps outbound to function correctly. The low setting required a lot less bandwidth, but also only has a reliability of 62.16% and a responsiveness of 2.071 s as shown in Section 6.2.1.

6.2.4 Conclusion

In this section, it was shown that safe storage only has marginally higher reliability, when compared to fast storage, but that it is much slower when compared to fast storage. It was also shown that parallel retrieval leads to higher reliability and responsiveness at a cost of higher required bandwidth. The higher required bandwidth is still less than that required by the medium or high overlay storage configurations.

Group storage was found to be efficient in terms of higher layer data. Since storage and retrieval are separate data streams, PithosTestApp transfers a total of 161 Bps to and from Pithos. In response to those two data streams, group storage generates approximately 183 Bps traffic. This means that 88% of data in group storage is received from or destined to PithosTestApp, with 12% overhead. Overlay storage on the other hand generates approximately 86% overhead. A reduction in overlay storage overhead will, therefore, significantly decrease the amount of overall Pithos overhead.

To place bandwidth values into perspective, the total required bandwidth for Pithos when using parallel group storage and medium overlay storage, still only requires 15.4 kilo bits per second (kbps) inbound and 15 kbps outbound. This is well within the limits of most modern Internet connections that are capable of providing 1 to 40 Mbps bandwidth.

6.3 Responsiveness distributions

To evaluate responsiveness, it is not sufficient to only evaluate mean responsiveness, since the standard deviation also plays a role. The time it takes for the authoritative storage to retrieve an object will influence the latency users experience when interacting with the virtual environment (VE). Real-time VE interactions are sensitive to both latency and jitter. It is thus important to also review the range that latencies might have.

To calculate the responsiveness, the Oversim SimpleUnderlayNetwork was used for the physical network for all experiment but one.

The percentiles shown on all distributions indicate the percentage of requests serviced with a latency less than the individual time.

This sections presents all responsiveness distributions in the Pithos simulation. With the exception of the LAN experiment, the experimental setup is as in Section 6.1:

- Network of 2500 peers and 100 super peers.
- Simulation length of 10,000s.

- Exponential object lifetime with 1800s mean and 300s TTL.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- Storing six object replicas.
- All requests are for local group objects.

6.3.1 Overlay storage and retrieval

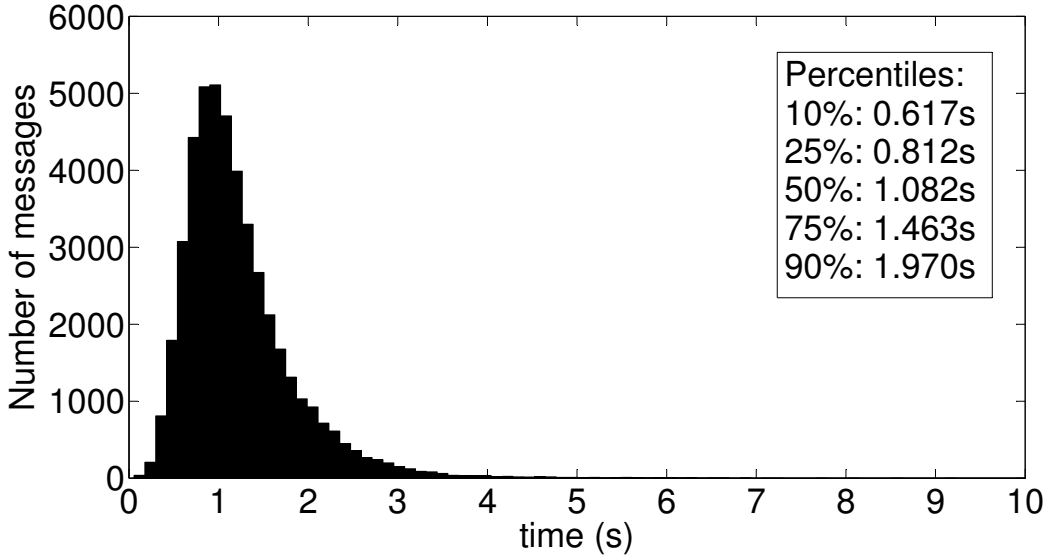


Figure 6.1: Overlay storage responsiveness

Figure 6.1 shows that to store an object in the overlay can take anywhere from 0.1 to 4 seconds, but 80% of storage requests are less than 1.569s.

Figure 6.2 shows that to retrieve an object from the overlay can take anywhere from 0.1 to 6 seconds, but 80% of requests are less than 2.322s. Figure 6.2 also shows a spike at 3 seconds, when overlay retrieval is performed. 0.55% of requests take 3 seconds to complete. When evaluating the CDF, it was found that a spike exists every 1.5s, with the magnitude of the spike decreasing for longer times.

The spikes are thought to be an artifact of the third-party DHT implementation, since this spike was found when simulating both Pastry and Chord so cannot be an artifact of any specific routing overlay. It was also found when simulating for the coordinate-based underlay as well as a fixed delay underlay, so it cannot be an

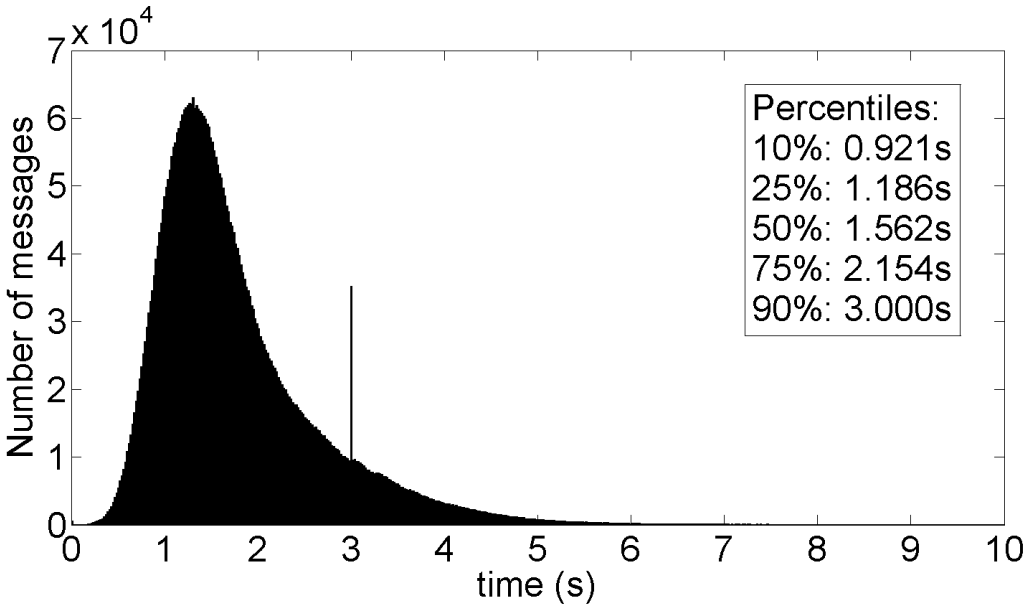


Figure 6.2: Overlay retrieval responsiveness

artifact of the type of underlay used. Because of the low percentage of requests that exhibit this behaviour, a rigorous study into this phenomenon was not performed.

6.3.2 Group storage

6.3.2.1 Fast storage

Figure 6.3 shows the distribution of group storage for fast storage.

Figure 6.3 shows a spike at zero seconds, with 1.82% of group storage requests having zero network latency. These are group requests that were made when no objects were available in a group and the request immediately returned a failure response. Apart from the spike, Figure 6.3 also shows that the responsiveness is distributed over a small range: with 86% of requests taking less than 0.1s.

6.3.2.2 Safe storage

Figure 6.4 shows the responsiveness distribution for safe storage. Because safe storage is slower than fast storage, the mean response time is longer. The maximum and minimum responsiveness values are also greater than for fast storage: from zero to 0.8 seconds, with 35% of requests taking less than 0.1s. This is still smaller than the range of overlay storage values.

Figure 6.4 also contains a spike at 10s. 0.68% of requests take 10s, which is due to the storage timeout being set to 10s. What should be noted is that these responsiveness graphs show both success and failure responsiveness. In other words,

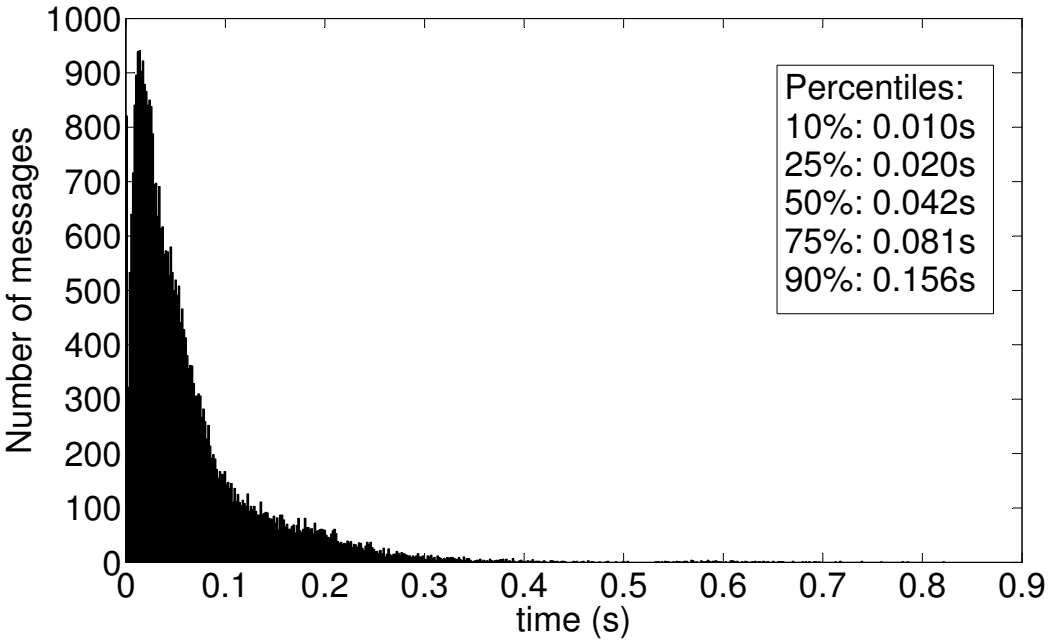


Figure 6.3: Group storage responsiveness for fast storage

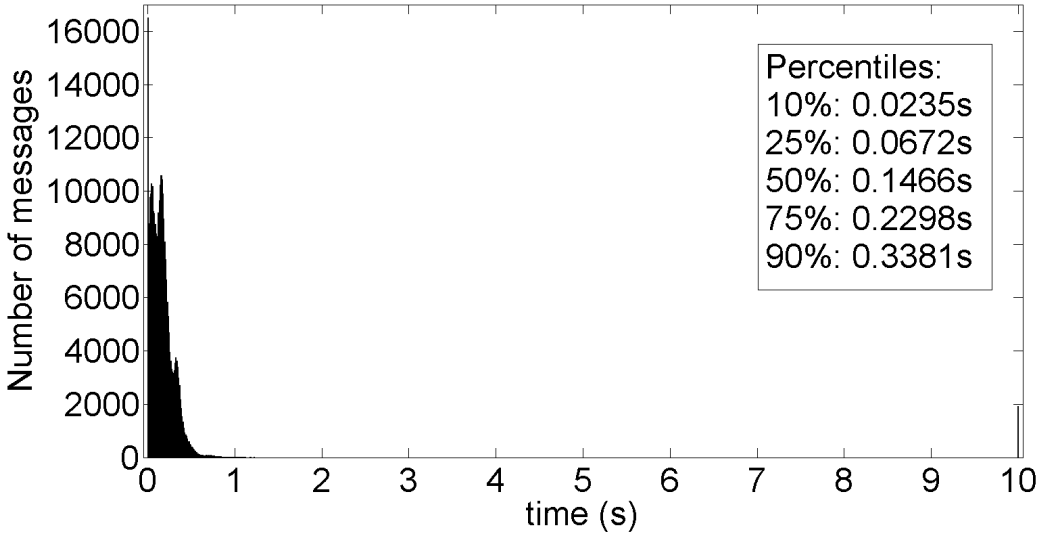


Figure 6.4: Group storage responsiveness for safe storage

it shows how quickly a response is received, even if that response is a failure. The responses received from 10s are all failure responses, where no response was received from the destination node and an internal timeout occurred.

6.3.3 Group retrieval

6.3.3.1 Fast retrieval

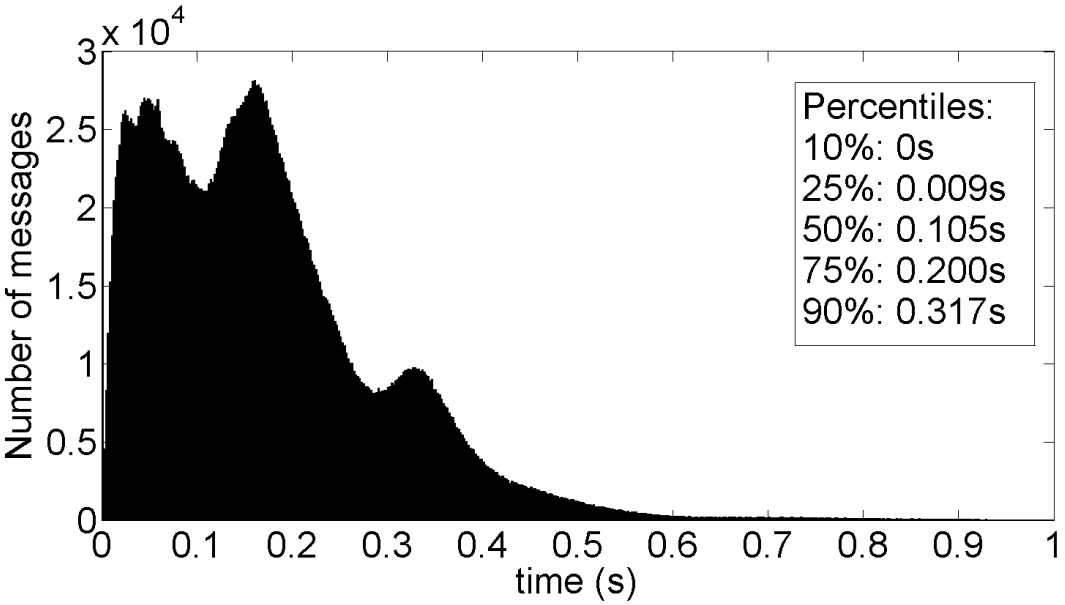


Figure 6.5: Enlarged view of the group retrieval responsiveness for fast retrieval

Figure 6.5 shows group retrieval responsiveness for fast retrieval, without showing the full extent of the zero time responses. This type of retrieval has 24% of requests with zero second latency. These requests are the requests where the node generating the object is chosen to store the object. If the required number of replicas is high, compared to the group size, there is a high likelihood that the node originating the request will be chosen as a host node to the object.

It should be noted that the node that originally stored the object does not have local access to the object. Only other nodes that did not originally store the object may have access to local copies. This is the reason why such a significant spike is not present in fast storage.

Figure 6.5 also shows multiple peaks, which is shown to be an artifact of the underlying physical layer in Section 6.3.4.

For fast group retrieval 90% of requests are serviced within 0.318s, compared to overlay retrieval where only 0.08% of requests are serviced within 0.318s.

6.3.3.2 Parallel retrieval

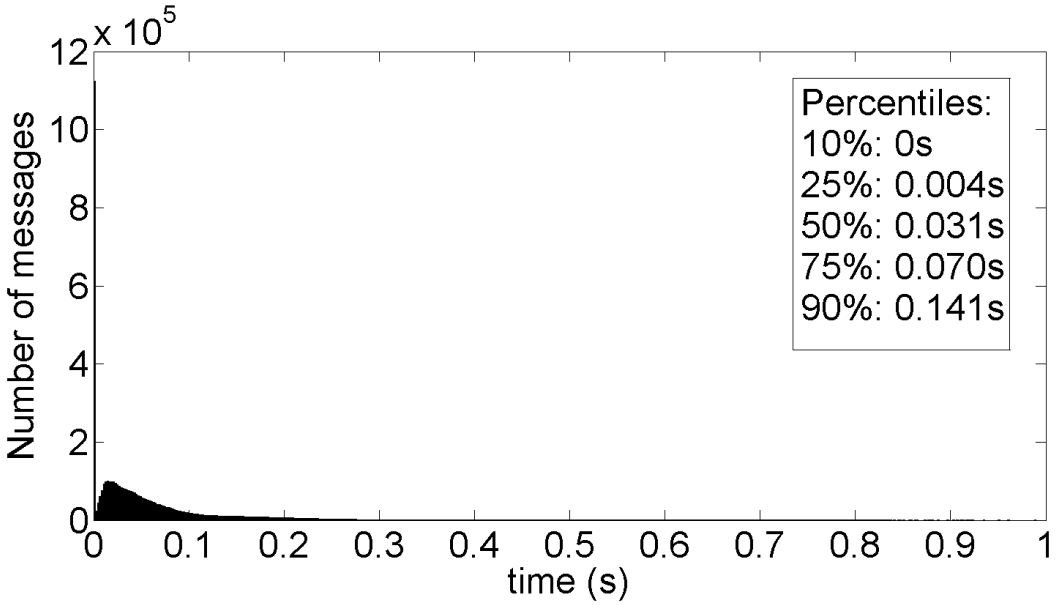


Figure 6.6: Group retrieval responsiveness for parallel retrieval

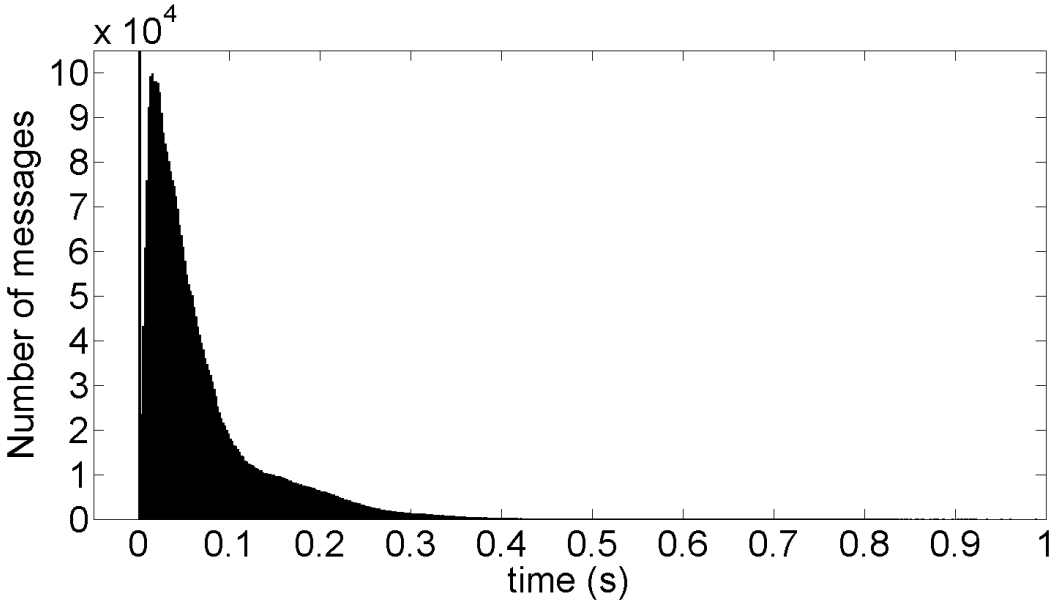


Figure 6.7: Enlarged view of the group retrieval responsiveness for parallel retrieval

Figure 6.6 shows group retrieval responsiveness for parallel retrieval. Figure 6.7 shows the enlarged view. 95% of parallel retrieval requests are serviced below 0.194s,

whereas only 74% of fast retrieval requests are serviced below 0.194s. The higher responsiveness of parallel retrieval is as a result of multiple replicas queried and the fastest response being used (as previously explained in Section 6.2.2).

The shape of Figure 6.7 differs from that of Figure 6.5 due to the difference between selecting a single random node $(1/r)$ in the group for retrieval, to selecting the fastest of r nodes in the group for retrieval, where r is the number of object replicas currently in the group.

6.3.4 LAN performance

It is expected that the characteristics of the underlying network architecture influence the performance of any higher layer architecture build on top of it. To investigate the influence of the underlay, an experiment is performed where the coordinate-based underlay network is replaced by a fixed delay 1ms underlay with 0.1% jitter.

It is expected that group storage should show two hop responsiveness, because of the fully connected group network topology, and that overlay storage should show more hops than group storage, because of the $O(\log(N))$ performance of the overlay.

6.3.4.1 Group retrieval

Testing the LAN performance of group retrieval allows for the investigation of the multimodal behaviour (local maxima) seen in Figures 6.4 and 6.5, which were suspected to be caused by the underlying network. The multimodal behaviour is a function of the distance of peers in the physical network layer.

It should also be verified that the responsiveness of Pithos will improve if the physical layer latencies are reduced.

Figure 6.8 shows group retrieval performance for fast storage and fast retrieval, running on a 1ms underlay network. This figure shows the same operation shown in Figure 6.5, but for the underlay being a fixed 1ms underlay, instead of a coordinate-based underlay.

Two main spikes are seen in the figure, one at 0s and another at 2ms. The second spike is surrounded by the 0.1% jitter, but the first spike contains no jitter, since no network message is sent out for the local storage. There are no other measured times. 24% of requests are served with zero network latency.

This is what is expected from group storage, since every request in group storage is either zero hops or two hops. For two hops, one hop is required to send the request, and another to receive the response, which gives $2 \times 1ms = 2ms$. The zero hops case is due to local objects being retrieved, as explained in Section 6.3.2.1.

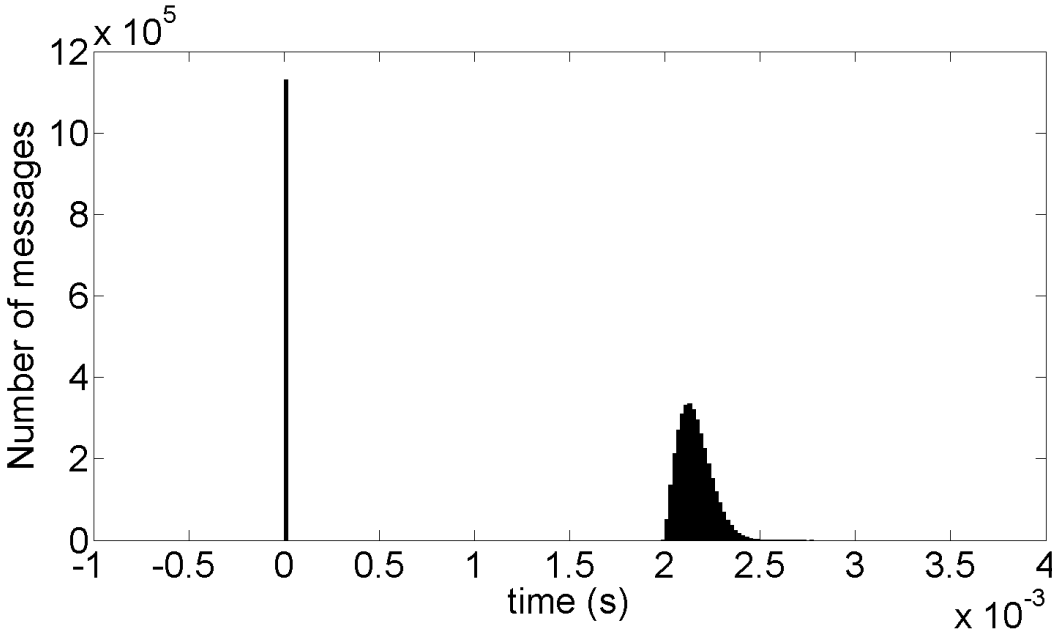


Figure 6.8: Group retrieval responsiveness for fast storage and fast retrieval, running on a 1ms underlay network.

No additional peaks are perceived, as opposed to the peaks seen earlier. The mean group storage performance has improved to 1.6ms, from the 192ms for the coordinate-based underlay network.

6.3.4.2 Overlay retrieval

Figure 6.9 shows the responsiveness of overlay storage for the LAN underlay network. Three groups of latencies can be observed. One close to zero seconds, one at 1.5s and another at 3s. The first group contains the majority (96%) of overlay requests, the second 1% and the third, 3% of overlay requests. It is not clear why there are three distinct responsiveness groups. Because these features are only responsible for 4% of the requests, they were not deemed to require an in-depth investigation.

Figure 6.10 shows the group close to zero seconds. This group, as well as the other two, exists out of multiple hops. These hops are, however, not underlay hops, but overlay hops, since they are more than 1ms apart. The overlay creates its own application layer network and the overlay storage takes $O(\log(N))$ overlay hops. A single overlay hop may contain multiple underlay hops, which is what the multiple hops represent in Figure 6.10.

The mean overlay retrieval for LAN responsiveness is 130ms, which is significantly slower than the 1.6ms of group retrieval for LAN responsiveness.

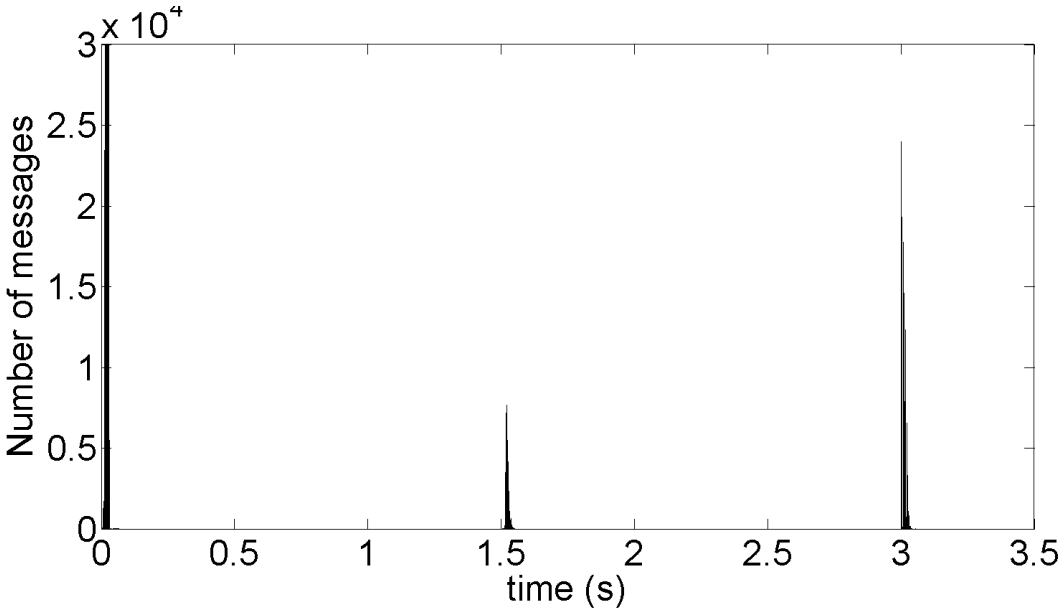


Figure 6.9: Overlay retrieval responsiveness for fast storage and fast retrieval, running on a 1ms underlay network and showing the compete x-axis.

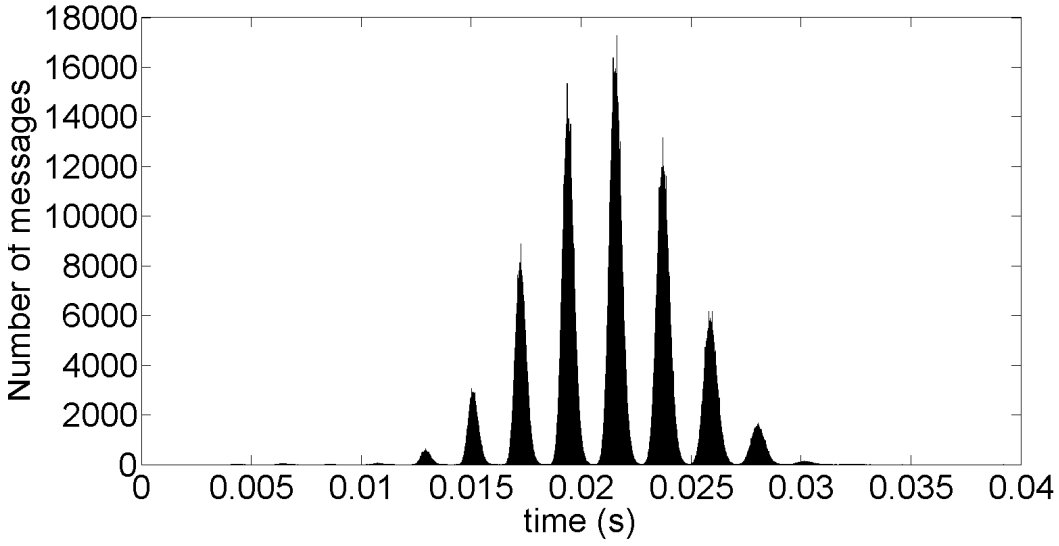


Figure 6.10: Overlay retrieval responsiveness for fast storage and fast retrieval, running on a 1ms underlay network.

6.3.5 Overall storage

6.3.5.1 Fast storage

After having presented the separate responsiveness profiles for overlay and group storage, the overall Pithos responsiveness is presented here which is a combination of the responsiveness profiles of the underlying group and overlay responsiveness.

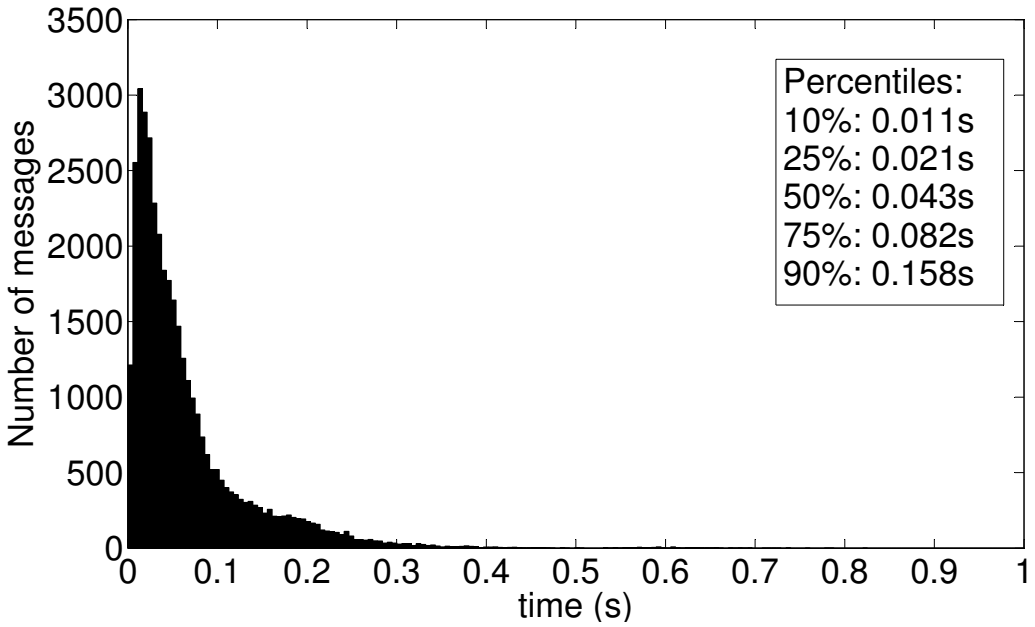


Figure 6.11: Overall storage responsiveness for fast storage

Figure 6.11 shows the overall storage responsiveness for fast storage. The shape is similar to that of Figure 6.7, a seemingly exponential distribution. The shape is derived from how fast storage tracks a success: the first successful storage response from group storage is sent to the higher layer, effectively taking all responses and choosing the fastest one. Fast storage responsiveness is, therefore, exactly the same mechanism as with parallel retrieval, which is why the distributions look that same.

95% of fast storage requests are serviced within 0.185s.

6.3.5.2 Safe storage

Figure 6.12 shows the overall storage responsiveness for safe storage. Because a response is only sent when sufficiently many group responses have been received *and* when the overlay response has been received, the shape of the distribution is identical to that of the overlay storage distribution, shown in Figure 6.1.

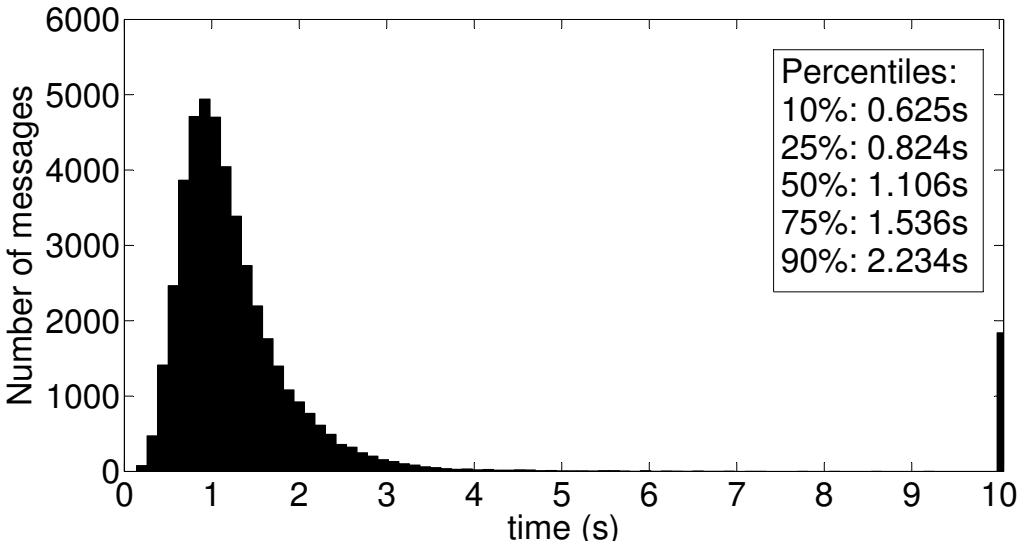


Figure 6.12: Overall storage responsiveness for safe storage

The reason why the shape has no correspondence to group storage responsiveness is because, as previously shown, overlay storage is almost always slower than group storage. Because safe storage always waits for the reply from the overlay storage module, its responsiveness is dominated by the responsiveness of overlay storage.

95% of safe storage requests are serviced within 3.373s. Compared to fast storage, only 0.0174% of safe storage requests are serviced within 0.185s.

6.3.6 Overall retrieval

6.3.6.1 Fast retrieval

Figure 6.13 shows overall retrieval responsiveness for fast retrieval. The shape is similar to that of fast group retrieval in Figure 6.5, because in this case, fast retrieval returns the first result received. Since fast group retrieval is mostly faster then overlay retrieval, the overall retrieval distribution will closely match that of fast group retrieval.

6.3.6.2 Parallel retrieval

Figure 6.14 shows the overall retrieval responsiveness for parallel retrieval. Overall parallel retrieval mirrors parallel group retrieval, since parallel retrieval is faster than overlay retrieval.

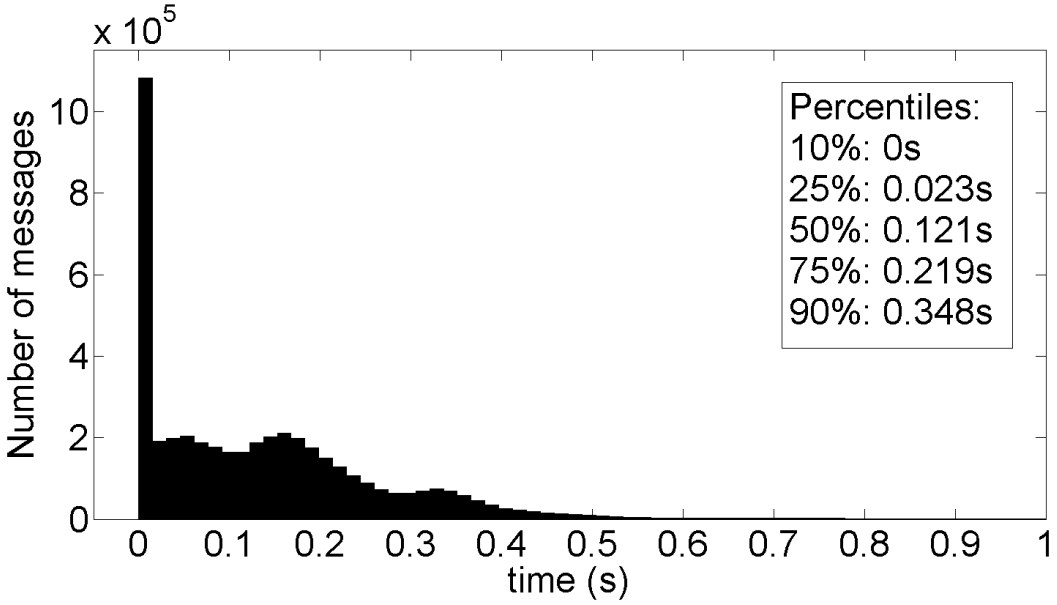


Figure 6.13: Overall retrieval responsiveness for fast retrieval

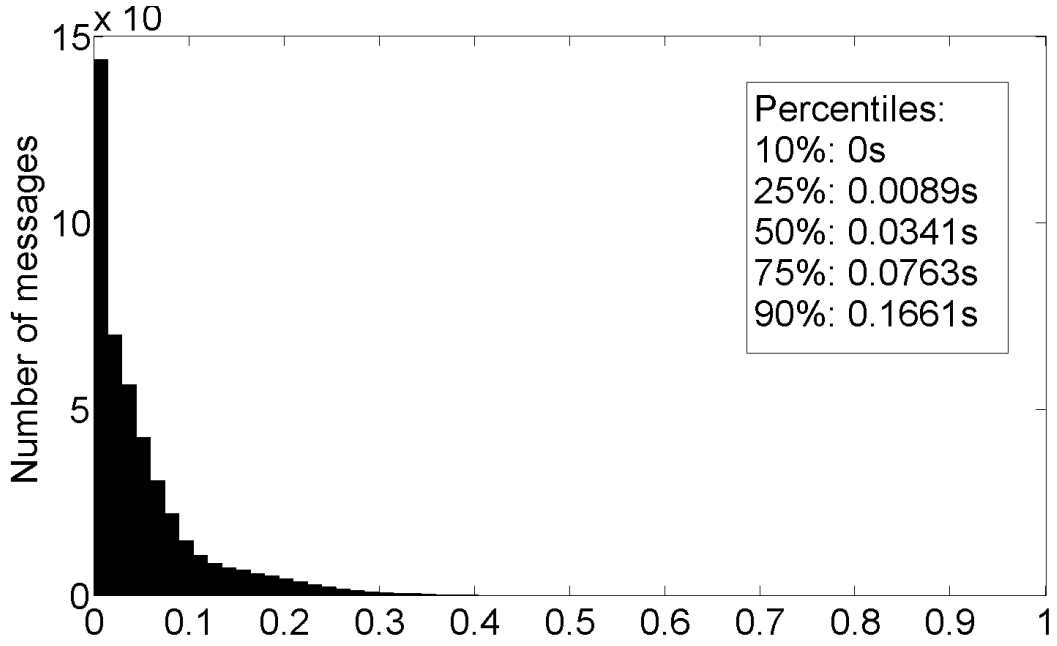


Figure 6.14: Overall retrieval responsiveness for parallel retrieval

6.3.7 Conclusion

In this section some responsiveness distributions were shown. The first reason is that a fair comparison does not only consider the mean, but compares the distributions. These include the distributions for overlay storage and retrieval, fast and safe group storage, fast and parallel group retrieval, LAN performance, fast and safe overall storage, and fast and parallel overall retrieval. Secondly, it enables the highlighting of various structures and characteristics of storage and retrieval as they relate to certain quantities or mechanisms in Pithos. Thirdly, the distributions verify that Pithos meets the responsiveness and reliability design requirements.

When comparing the overall Pithos performance to the underlying group and overlay performance, it shows that Pithos is indeed a hybrid of those two storage types, but also that it is selecting the best qualities of both. This means that Pithos should perform better than the overlay chosen for use in Pithos.

6.4 Reliability, responsiveness and bandwidth for various group probabilities

The design of Pithos makes use of user groups and distance-based storage on a group layer, as stated in Sections 4.3.1.4 and 4.3.1.8. To verify the actual reliability and responsiveness of Pithos, it is necessary to investigate Pithos's performance for various group probabilities. In the final P2P MMVE architecture it is assumed that the percentage of in-group requests will be much higher than that of out-of-group requests, since distance-based storage is used. This assumption is based on the fact that with group-based distance-based storage, the objects that are frequently of interest to the user will be stored in that user's group.

6.4.1 Experimental setup

Because the exact probability that an object request will be for a group object is not yet known, a sweep is done for various group probabilities (the probability that a request is for a in-group object) to investigate the effect of group probability on Pithos's performance.

Because we expect that as the group probability changes, we will have a linear combination of group performance and overlay performance, the low overlay configuration is selected to better visualise this relationship.

The simplest storage and retrieval methods: fast storage and fast retrieval are selected. The specific type of group storage retrieval method is not important for

this experiment, since the purpose of the experiment is to show the interplay between group and overlay storage.

As with the previous experiments, the following configuration parameters are chosen for reasons already discussed:

- Network of 2500 peers and 100 super peers.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.
- Exponential object lifetime with 1800s mean and 300s TTL.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- No object repair is performed.
- Storing six object replicas.

6.4.2 Reliability

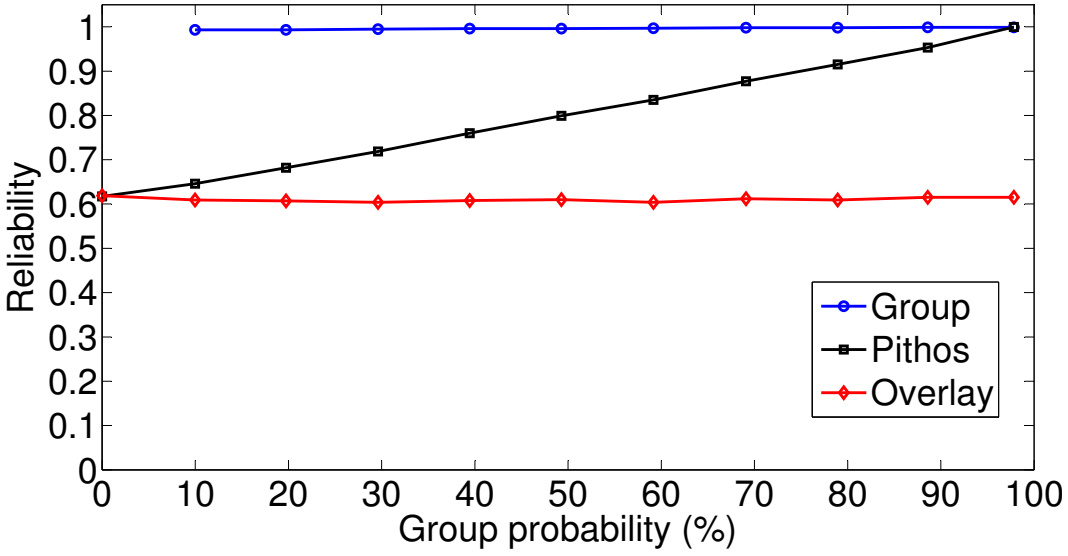


Figure 6.15: Reliability of Pithos, group storage and overlay storage for various group probabilities.

Figure 6.15 shows the effect group probability has on Pithos's overall reliability, compared with the underlying group storage and overlay storage reliabilities. The figure shows that the most reliable storage mechanism is group storage. It shows that group and overlay reliability are independent of the group percentage, which is as expected since the group storage and overlay storage modules are independent. Each request that is received from the higher layer (PithosTestApp) is relayed to both the DHT storage and group storage modules by the peer logic module.

When an object request is sent to group storage for an object that is not stored in that group, the group storage module reports that there does not exist such an object in the group and the request is not handled. The only possible reply is then from the overly storage module. Figure 6.15 shows that the overall Pithos reliability is a linear combination of group storage and overlay storage, weighted by the group probability.

These results can be verified theoretically by seeing that the reliability of Pithos is the union of the reliability of both group and overlay storage and is given by

$$\Gamma_{\text{Pithos}} = \Gamma_{\text{group}} \cup \Gamma_{\text{overlay}} \tag{6.4.1}$$

$$\begin{aligned} \text{which } E[\Gamma_{\text{Pithos}}] &= E[\Gamma_{\text{group}} \cup \Gamma_{\text{overlay}}] \\ &= E[\Gamma_{\text{group}}] + E[\Gamma_{\text{overlay}}] \\ &= P(\text{group})\Gamma_{\text{group}} + (1 - P(\text{group}))\Gamma_{\text{overlay}} \end{aligned} \tag{6.4.2}$$

where Γ_{Pithos} is the overall Pithos reliability, Γ_{group} is the group reliability, Γ_{overlay} is the overlay reliability, and P_{group} is the group probability.

This result shows that the reliability of Pithos varies linearly between overlay and group storage reliability as the group probability is varied. It should be noted that if a more reliable overlay was used, the medium or high configurations for example, the difference between group and overlay reliability is less. The low overlay configuration was just selected to better illustrate the linear relationship.

6.4.3 Responsiveness

Figure 6.16 shows the responsiveness of Pithos compared with the underlying responsiveness of group and overlay storage. Take note that a lower value is preferred in this graph and that low latency is high responsiveness. As with the reliability, overall responsiveness is also shown to be a linear combination of group and overlay responsiveness.

An apparent anomaly is the group storage latency that increases with an increase in group probability. Recall that if a request is sent to group storage for an object not in the group, group storage immediately returns failure. The failure response is recorded as a response from group storage, even if this response is a failure.

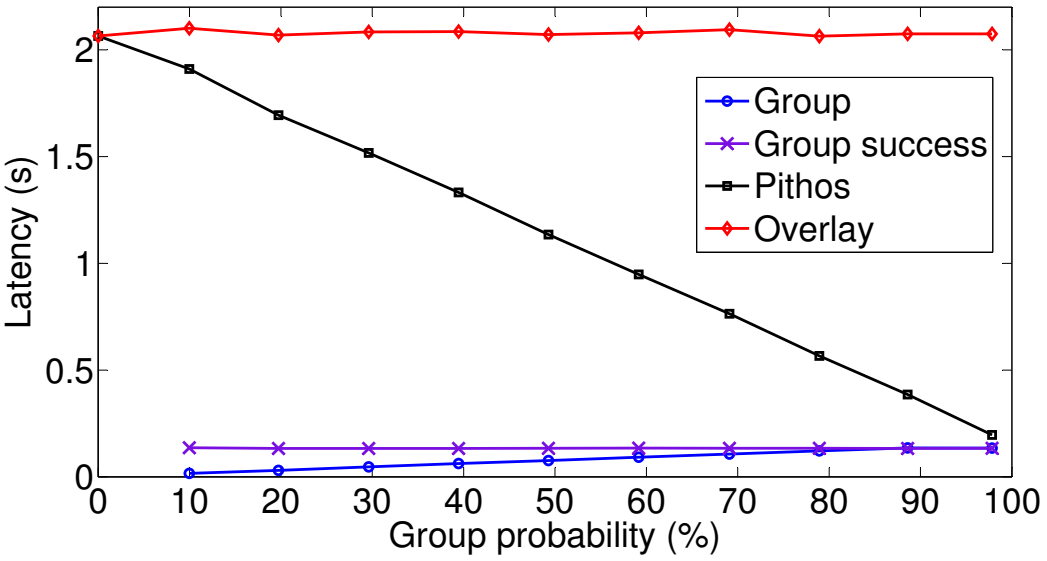


Figure 6.16: Responsiveness of Pithos, group storage and overlay storage for various group probabilities.

The lower the group probability, the more failure responses are returned by group storage, which are returned instantly. It is these responses that decrease the mean group storage latency for lower group probabilities. When failure responses are ignored, group latency remains constant at the value of 100% group probability.

6.4.4 Bandwidth

Figure 6.17 shows the bandwidth requirements of Pithos in bytes per second (Bps). Pithos, group and overlay bandwidths show the mean of the inbound and outbound bandwidth. Figure 6.17 also shows the underlying bandwidth requirements of group storage and overlay storage. To determine the actual overhead that Pithos requires, the graph also shows the amount of data that is sent to, and received from, Pithos, from and to the higher layer (PithosTestApp).

Figure 6.17 shows that overlay storage bandwidth remains constant, but that group storage bandwidth increases. Overlay storage remains constant, because no matter what the group probability is, the overlay storage is always queried for a data item and a data item is returned with the same probability (based on the reliability). Group storage, on the other hand, only returns data if the object exists within the group. Higher group probability means that more objects are requested from within the group, which means that group probability will return more objects and thereby use more data.

As shown, the overall Pithos bandwidth the the sum of the bandwidth required by group and overlay storage. The Pithos overhead is approximately 76% of Pithos's

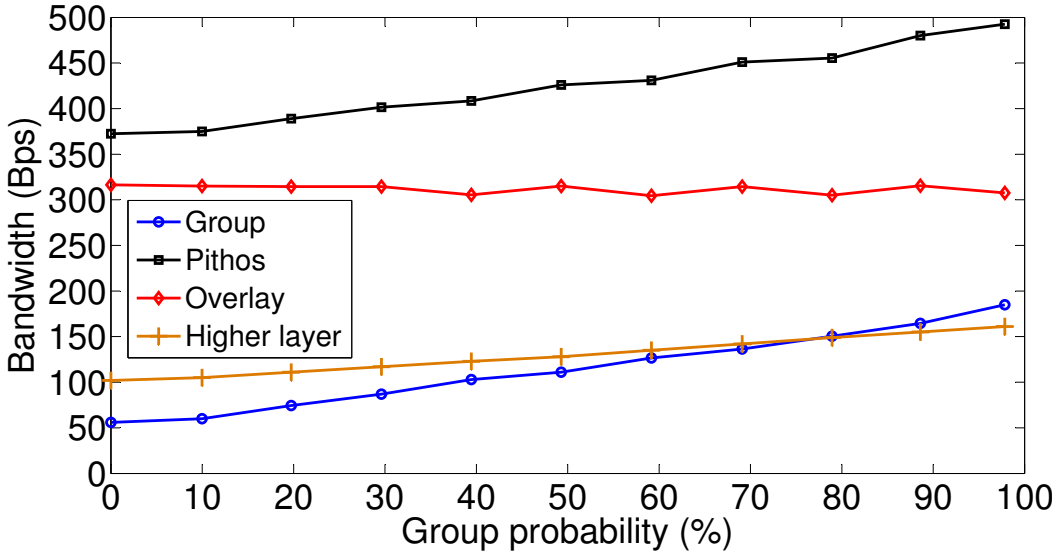


Figure 6.17: Bandwidth usage of Pithos, group storage and overlay storage for various group probabilities, as well as the bandwidth sent to, and received from Pithos.

UDP data, where, depending on the group probability, 85% to 62% of overhead is contributed by overlay storage for the low overlay configuration.

6.4.5 Conclusion

This section reviews the performance of Pithos for various group probabilities and also shows the interaction between overlay and group storage in terms of reliability, responsiveness and bandwidth.

All the experiments verified that for a low group probability, the characteristics of overlay storage are dominant and for high group probability, the characteristics of group storage are dominant. As previously stated, because of the distance-based storage design, high group probabilities are expected.

What was shown in this section is that overlay storage provides lower reliability than group storage with higher bandwidth requirements. This underpins the need for an overlay better suited to environments with network churn and improved lookup reliability. Before overlay efficiency can be improved, it is therefore important to create a grouping algorithm that has the highest possible group probability.

6.5 Reliability, responsiveness and bandwidth for various peer lifetimes

Object replicas inserted into Pithos will all eventually disappear due to network churn. The rate at which replicas are removed from the network can be reduced

by employing object repair. This enables Pithos to trade storage space for network bandwidth, by using fewer replicas with a higher repair rate. Network bandwidth can also be saved by having more replicas and a lower repair rate.

The effect of node replicas on the reliability, responsiveness and bandwidth usage of Pithos will be explored in this section. Reliability is tightly coupled to the lifetime of peers, since peers that live longer will store objects longer, meaning objects stored on peers with longer expected object lifetimes will have longer expected lifetimes themselves. The main reason for being unable to retrieve an object from Pithos is because a peer has left the network. It is, therefore, expected that if peers life longer the system reliability increases.

6.5.1 Experimental setup

It is expected object repair will increase objects lifetimes, thereby increasing reliability, but also increasing bandwidth usage. It is also expected that object lifetimes are directly related to peer lifetimes. To show this, node lifetimes in the experimental setup were varied from 100s to 1800s. Time steps were selected closer together, for period where reliability, responsiveness and bandwidth values changed faster.

Three types of retrieval methods are compared: fast retrieval, parallel retrieval and fast retrieval with object repair. Two types of repair are compared: periodic and leaving repair. Periodic repair is set to have repair periods of 100s.

As was stated earlier, the object TTL will also influence the reliability, since if objects have to be maintained for longer periods of time there exists a greater probability of all object replicas being lost. To better show the effect of repair on object reliability, object TTLs were increased to 1000s from 300s.

To show the sensitivity of reliability to node lifetime, two node lifetime distributions are explored in this experiment: the exponential and Pareto lifetime distributions. The Pareto distribution is a heavy-tailed distribution, which means that a certain percentage of nodes will live a significantly longer time than others. It is expected that the Pareto distribution will lead to higher reliability, since there is a higher probability of one of the object replicas being stored on a node with a long lifetime.

The rest of the experimental setup is as follows:

- Fast group storage is used.
- Network of 2500 peers and 100 super peers.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.

- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- Storing six object replicas.
- All requests are for local group objects.

6.5.2 Reliability

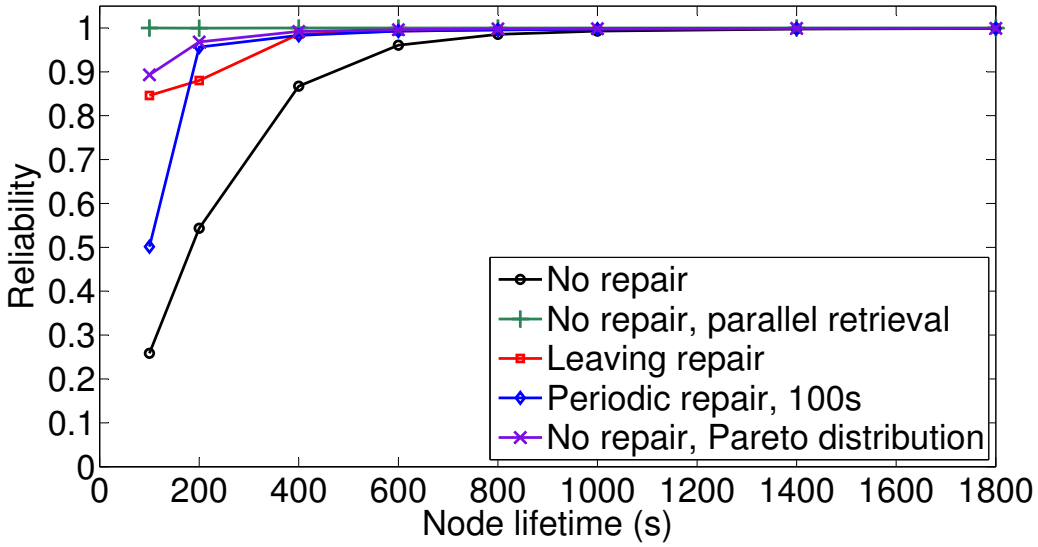


Figure 6.18: Reliability of Pithos for various mean node lifetimes, comparing no repair, no repair with parallel retrieval, no repair using Pareto node lifetimes, leaving repair and periodic repair with 100s periods.

Figure 6.18 shows request reliability for various node lifetimes for no repair, no repair using parallel retrieve, leaving repair and periodic repair.

As expected, doing no repair does not cope well with short node lifetimes compared to an object TTL of 1000s. The object TTL is important when evaluating performance, since a network that requires objects to be stored for longer has to maintain the object replicas for a longer amount of time. When the expected node lifetime is comparable to the object TTL, the reliability of the system when not using repair is similar to the system reliability when repair is used. In other words, when peer lifetimes are of the same order of magnitude as object TTL, repair is not necessary.

This observation is helpful when designing a P2P MMVE. Objects might be classified according to how long they should remain in the system. Objects that should only remain in the system for approximately the same amount of time as the expected user session time need not use repair mechanisms, if a sufficient number of replicas are used.

When node lifetimes are the same as the periodic repair interval, leaving repair copes better than periodic repair. The periodic repair mechanism might be adjusted to less than the minimum expected node lifetime in the system, but this will increase the load on the super peer, which has to perform the periodic checks. From the data it seems that leaving repair might be able to better handle a situation where node lifetimes have a high dynamic range, since the rate at which peers leave the network is directly tied to how often leaving repair performs repairs.

Except for the case where the periodic repair timer is equal to the expected node lifetime, leaving and periodic repair appear to perform similarly in terms of reliability.

Parallel retrieval, employing no repair mechanism, is most reliable and consistently reliable for all node lifetimes. The second most reliable setup is no repair with Pareto lifetime distributions. It should be noted that a Pareto node lifetime distribution leads to more reliable system in every instance. In other words, when comparing the cases of no repair with repair, Pareto lifetime distributions with repair also performs better than the no repair case for the same lifetime distributions.

6.5.3 Responsiveness

Figure 6.19 shows the latencies for various mean node lifetimes. The time that it takes to service a request is highly coupled to the reliability of the system. More requests will fail for a less reliable system. The only way for the peer making the request to know that a request has failed is for that request to timeout. The more requests that fail, the more the latency migrates to the timeout latency of 10s. This shows the importance of choosing a timeout as short as possible, that is still able to handle all requests.

6.5.4 Bandwidth

Figure 6.20 compares the bandwidth usage of the various repair techniques. It should be noted that the largest part of Pithos's bandwidth usage is contributed by overlay storage, as shown in Sections 6.2.1 and 6.4.4.

Although parallel retrieval has been shown to be the most reliable retrieval mechanisms for various node lifetimes, Figure 6.20 shows that it required the most bandwidth, with only between 1.5% to 8% usable data transferred depending on node

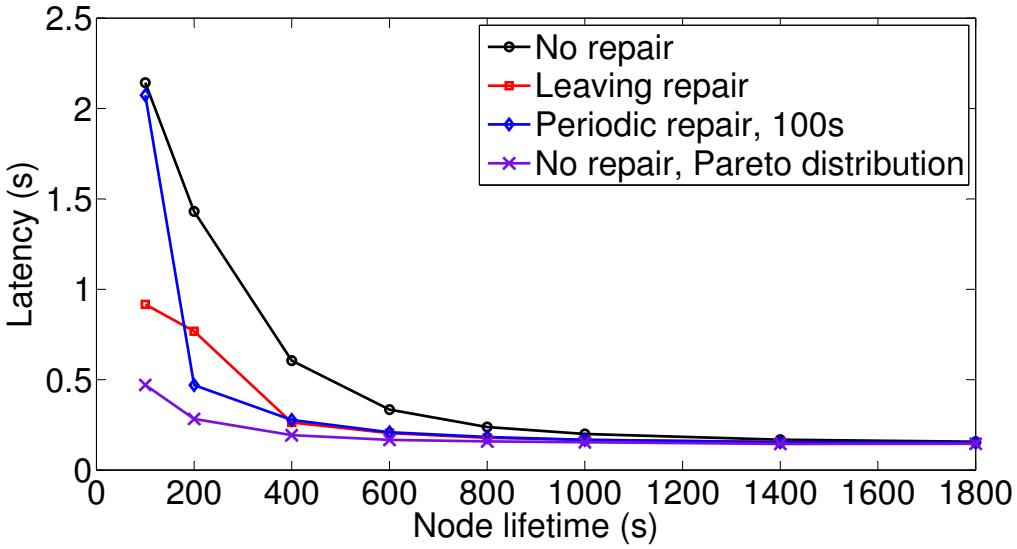


Figure 6.19: Latency of Pithos and the higher layer requests for various mean node lifetimes for the the case with no repair, no repair using parallel retrieval, no repair using Pareto node lifetimes, leaving repair and periodic repair.

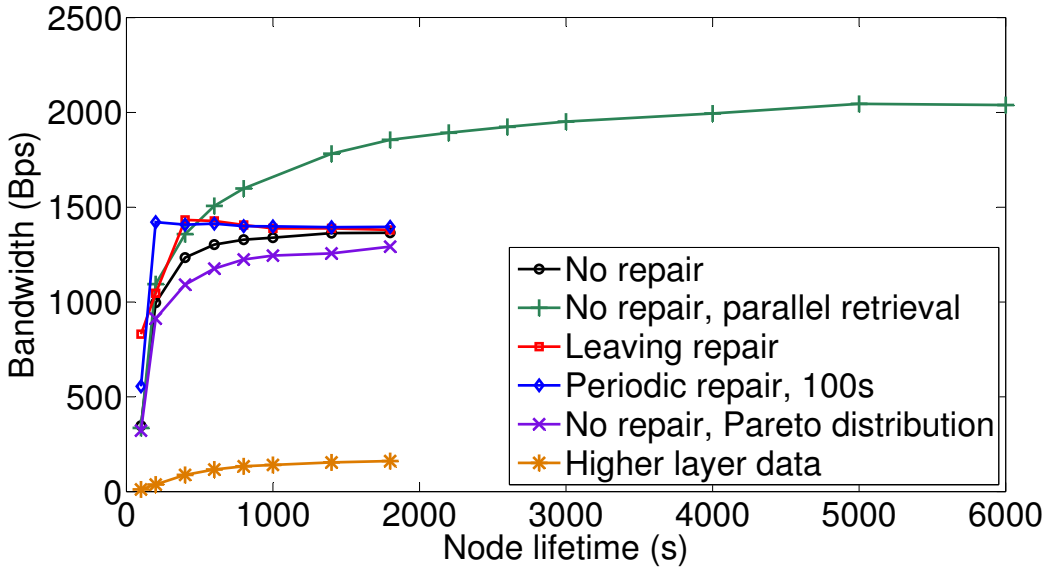


Figure 6.20: Bandwidth usage of Pithos and the higher layer for various mean node lifetimes for the the case with no repair, no repair using parallel retrieval, no repair using Pareto node lifetimes, leaving repair and periodic repair.

lifetime. Parallel retrieval also converges to a higher required bandwidth of approximately 2000Bps, compared to other methods that converge to approximately 1400Bps for high node lifetimes.

The reason why parallel retrieval's bandwidth continues to grow is due to the mean number of object replicas available. Although parallel retrieval always attempts to retrieve six replicas (for this experiment), depending on the network conditions, the average number of object replicas is always smaller than the required number of object replicas. This is especially true for the case with no repair, where an object starts with the maximum number of replicas, but that number gradually decreases over the lifetime of the object. Object lifetime, which determines the average number of available replicas, is highly dependant on node lifetime, as will be discussed in Chapter 7.

This means that the longer the average node lifetime, the higher the average number of replicas per object. The bandwidth usage shown for parallel retrieval thus converges to an amount representing six replicas in the system. The bandwidth does, however, not increase six fold, since most of the bandwidth is due to overlay storage, which does not increase.

The bandwidth usage of leaving and periodic repair are similar for long peer lifetimes, compared to the periodic repair timer. Bandwidth usage is low for short node lifetimes, because of the low reliability. Many retrieve requests fail for short node lifetimes, which means that fewer object are returned and less bandwidth is used.

Although doing no repair is less reliable, it used less bandwidth, because no object repairs have to be performed. Pareto lifetimes distributions also require less bandwidth, since the overlay does not have to exchange as much routing information.

6.5.5 Conclusion

The leaving and periodic repair mechanisms were found to possess similar reliability and bandwidth performances. It seemed that leaving repair does, however, cope better with dynamic node lifetimes. A periodic timer also has to be designed for specific network parameters, including expected node lifetime and object TTL. If, during the lifetime of the network, the parameters change, the system will not be able to dynamically adapt.

As expected, doing no repair is less reliable than doing repair, but uses less bandwidth. Parallel retrieval is highly reliable, but requires a large amount of bandwidth.

Pithos shows improved performance when using Pareto lifetime distributions, discussed in Section 5.1.2.3, compared to exponential node lifetime distributions. Since Pareto distributions have been found to better match node lifetimes in virtual

worlds, this improvement is significant. Exponential distributions were used in this section precisely because they are a more rigorous test of Pithos, compared to Pareto distributions.

6.6 Security

The results thus far have shown that Pithos is both responsive and reliable. In this section, in order to verify that Pithos meets the security design requirements specified, we investigate the object retrieval reliability in the presence of malicious users in the P2P network.

As a result of its design, fast retrieval is not resistant to malicious users, since it will retrieve corrupted objects, if the first retrieval response was received from a malicious user. Safe retrieval uses a quorum mechanism to respond with the object, which the majority of queried peers agree is correct. We therefore expect safe retrieval to be resistant to the presence of malicious users.

We also wish to investigate the reliability of the security mechanism as a function of the number of queried peers. We expect the reliability to increase with a larger quorum, but at the expense of a slower response time and an increase in bandwidth.

6.6.1 Experimental setup

Each object in Pithos has a value attribute that is set to a random floating point value when an object is created for the simulation. The value of the object is stored in PithosTestApps's global object list. It should be noted that for simulation purposes, an object's size is not related to its content. An object's size is set using a random distribution and this is the value used to compute transmission bandwidth and latency.

Whenever a malicious node receives a retrieval request it replies with a modified version of the requested object. The object is modified by modifying the object value. An object is assigned a uniformly random value by a malicious node. Because of the way objects are altered, we implicitly assume that there exists no collusion in the network. When PithosTestApp receives a requested object, it checks that object against the global object list to determine whether the correct object was received. PithosTestApp then records Pithos's reliability.

Pithos selects the object that is the same in a majority of responses and sends it to the higher layer. If no object has a majority, a failure response is sent to the higher layer.

The percentage of users that are malicious is varied from 0% to 100% in steps of 12.5%. A comparison of safe retrieval is performed by first retrieving four of the

six stored replicas and then increasing the number of retrievals to six.

As with the previous experiments, the following configuration parameters are chosen for reasons already discussed:

- Fast group storage is used.
- Network of 2500 peers and 100 super peers.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.
- Exponential object lifetime with 1800s mean and 300s TTL.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- Storing six object replicas.
- No object repair is performed.
- All requests are for local group objects.

6.6.2 Reliability

Figure 6.21 shows Pithos's reliability as a function of the percentage of malicious users in the network for both the fast retrieval and safe retrieval schemes. As expected, the fast retrieval reliability is almost the same as the malicious user percentage. System reliability for fast retrieval is the product of the malicious user factor and the system reliability when malicious users are present in the network.

The reason for this direct relationship is because the probability of an object retrieval being corrupted in the network is equal to the probability of a peer being malicious, since a malicious peer corrupts all objects returned and a non-malicious peer corrupts none of the objects returned. The overall probability is then the probability that an object retrieval would have succeeded if no malicious nodes were present, multiplied by the probability of nodes being malicious.

Overlay retrieval, using no form of security as with fast retrieval, performs worse than fast retrieval. This is especially true for higher malicious node probabilities.

Figure 6.21 also shows the improvement received from using safe retrieval. The curve shows that safe retrieval provides greater benefit for smaller factors of malicious nodes. As soon as the majority of objects are corrupted, which occurs when the malicious users probability is greater than 50%, object reliability drops significantly.

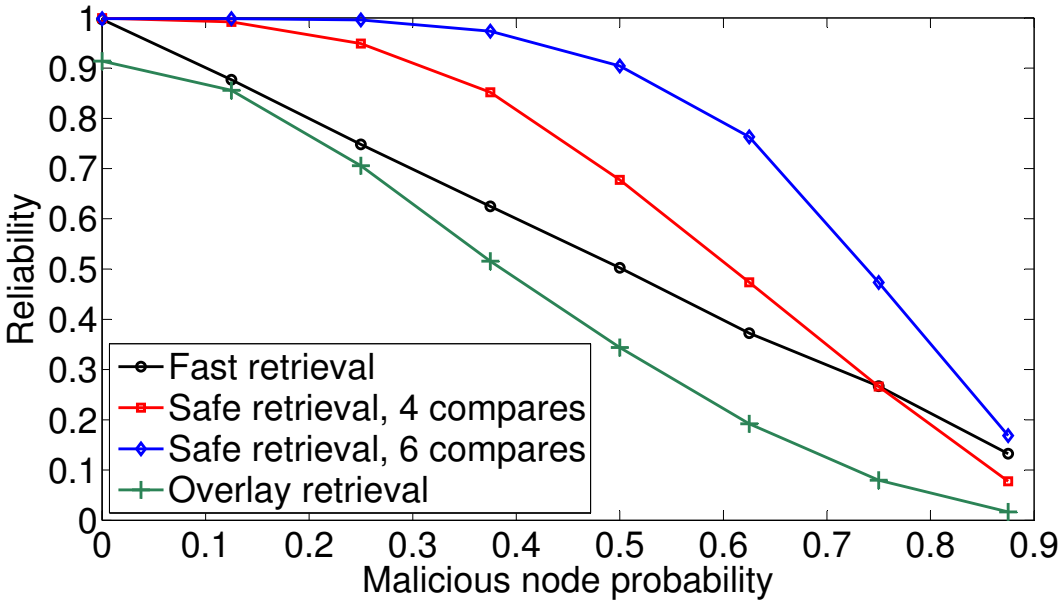


Figure 6.21: Reliability of various Pithos retrieval schemes for varying factors of malicious node probability.

The figure also shows improved reliability, when more received objects are compared to select a majority. Retrieving more objects for comparison will, however, increase bandwidth requirements. Bandwidth increases in line with that shown in Section 6.2.3, because safe retrieval is parallel retrieval, but for an additional compare operation. It should be noted that more retrievals than compares may be performed to increase responsiveness at the cost of additional bandwidth.

An interesting situation arises, where safe retrieval reliability drops below fast retrieval reliability. When a majority can no longer be identified by safe retrieval, a failure response is sent up. This is as opposed to fast retrieval that sends any object received. It is possible the the object that was selected by fast storage was not maliciously altered, but also not in the majority. In this scenario, fast retrieval will sometimes send objects to the higher layer that are correct, where safe storage just responds with failure, because it cannot achieve a majority decision.

Figure 6.21 also shows the performance of overlay storage under the presence of malicious nodes if no security mechanisms are implemented in overlay storage. The performance degrades as the percentage of malicious users increase. The performance is also worse than the overall Pithos performance. It should be noted that the overall Pithos performance is not influenced by Overlay storage in this graph, since 100% group probability is used. For lower percentages of group probability, the linear relationship between group and overlay storage, explored in Section 6.4 exists.

6.6.3 Responsiveness

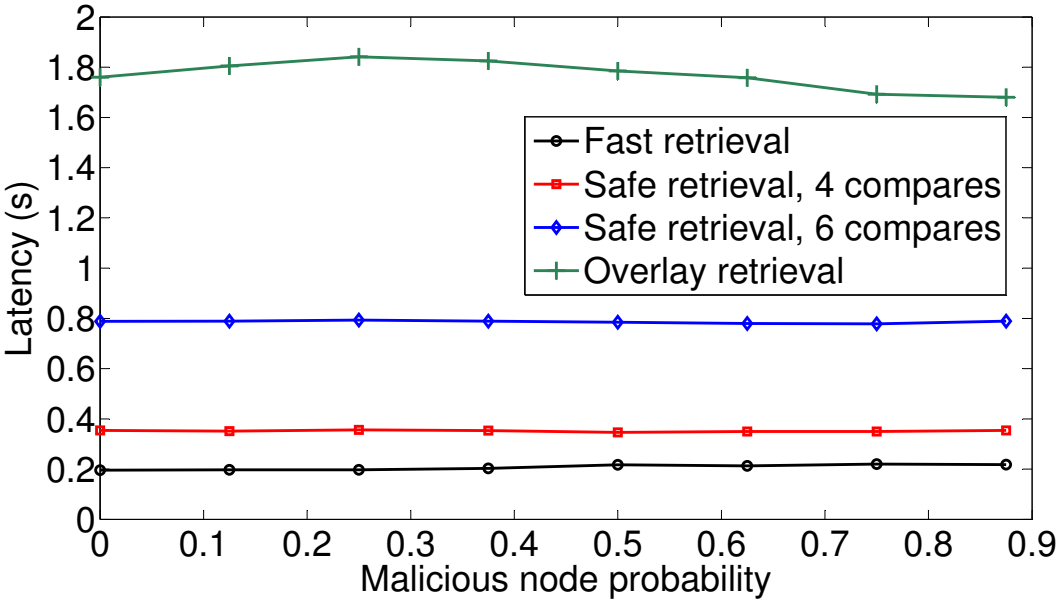


Figure 6.22: Responsiveness of various Pithos retrieval schemes for varying malicious node probabilities.

Figure 6.22 shows the responsiveness of Pithos for various malicious node probabilities. Fast retrieval has the lowest latency, since it responds with the first object received and does not wait to compare objects. Overlay retrieval is slower than all other forms of retrieval, because of the nature of overlay retrieval as previously discussed.

Safe retrieval, when comparing four objects is slower than fast retrieval, since it has to wait for four objects before it can compare. The responsiveness of the four compare retrieval is thus the responsiveness of the slowest of four requests. Safe retrieval, comparing six objects is slower than comparing four objects, since the responsiveness is now the slowest time of six requested objects. When requesting more objects, there is a higher probability that a slower object will be selected.

6.6.4 Bandwidth

Figure 6.23 shows the bandwidth usage of various retrieval schemes for various malicious node probabilities. Overlay retrieval uses the least amount of bandwidth, since Pithos also contains an overlay storage section, which means Pithos cannot use less bandwidth than overlay retrieval. Fast retrieval uses more bandwidth than

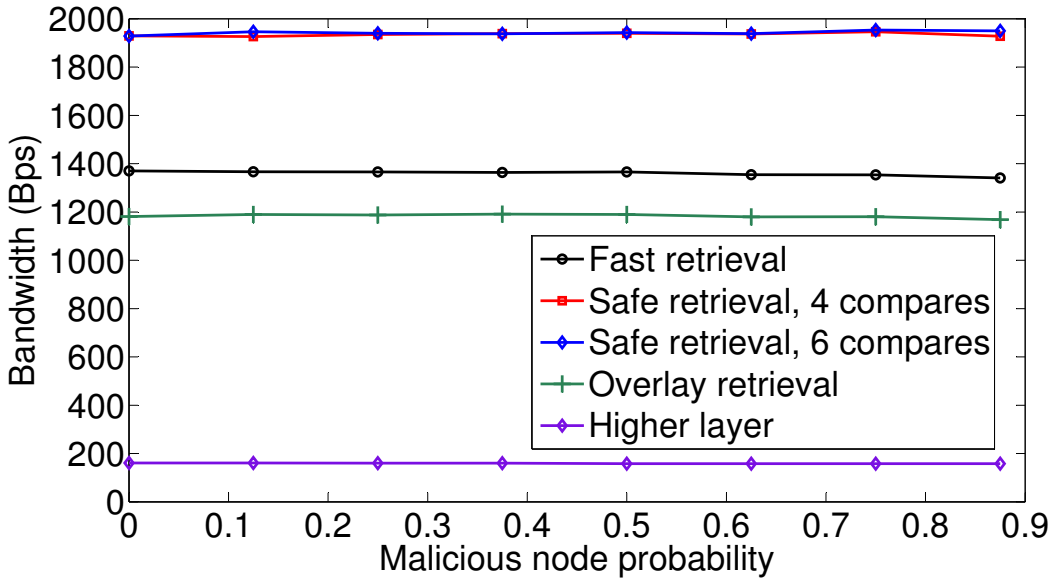


Figure 6.23: Bandwidth usage of various Pithos retrieval schemes for varying factors of malicious node probability.

overlay retrieval and the difference between overlay retrieval and fast retrieval is the amount of bandwidth required by the group storage module.

Safe retrieval performing four comparisons and safe retrieval performing six comparisons use the same amount of bandwidth, since both retrieval methods request six objects. The one method only uses the first four responses, while the other uses all six responses. Safe retrieval uses more bandwidth than fast retrieval, since six objects are requested in group storage, instead of the one object requested in fast retrieval.

6.6.5 Conclusion

The section shows that Pithos's safe retrieval mechanism can be used to increase reliability in the presence of malicious users in the network, at the cost of responsiveness and bandwidth. Since we cannot assume that there will not be malicious nodes in a virtual environment, this cost is deemed acceptable.

During the lifetime of the virtual world, it is important to monitor the malicious user percentage, to determine how parameters, such as number of retrieve requests and number of compares, should be adjusted to combat the threat while minimising bandwidth and maximising responsiveness.

In practice, a low malicious node factor is expected, since it is assumed that malicious users are in the minority. If the virtual world has a majority of malicious users, it will most likely not be sustainable.

6.7 Fairness

It has been shown that Pithos meets at least three of the five design requirements. In this section, it is verified that Pithos also meets the design requirement of fairness. Fairness requires that no peers should contribute none of their storage space and that no peers should have to contribute a significantly larger proportion of their storage space than other peers. We also compare Pithos to overlay storage and show it to be almost as fair as overlay storage.

6.7.1 Experimental setup

To measure object fairness, the length of time that nodes generate objects for were increased from 20s to 100s, to allow more objects to enter the storage system. The mean rate of object generation was kept constant at 1 object every 5s.

Object fairness is measured as the average number of objects that were stored on a node during its lifetime. Each node records its local object store size at regular intervals during its lifetime. At the end of a node's lifetime, it calculates the average number of objects that were stored on it. The results shown are, therefore, the average number of objects stored per node over that node's lifetime and the distribution is recorded over all nodes that were removed from the network during the simulation.

As with the previous experiments, the following configuration parameters are chosen for reasons already discussed:

- Fast group storage is used.
- Fast group retrieval is used.
- Network of 2500 peers and 100 super peers.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.
- Exponential object lifetime with 1800s mean and 300s TTL.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- Storing six object replicas.
- No object repair is performed.

- All requests are for local group objects.

6.7.2 Overlay fairness

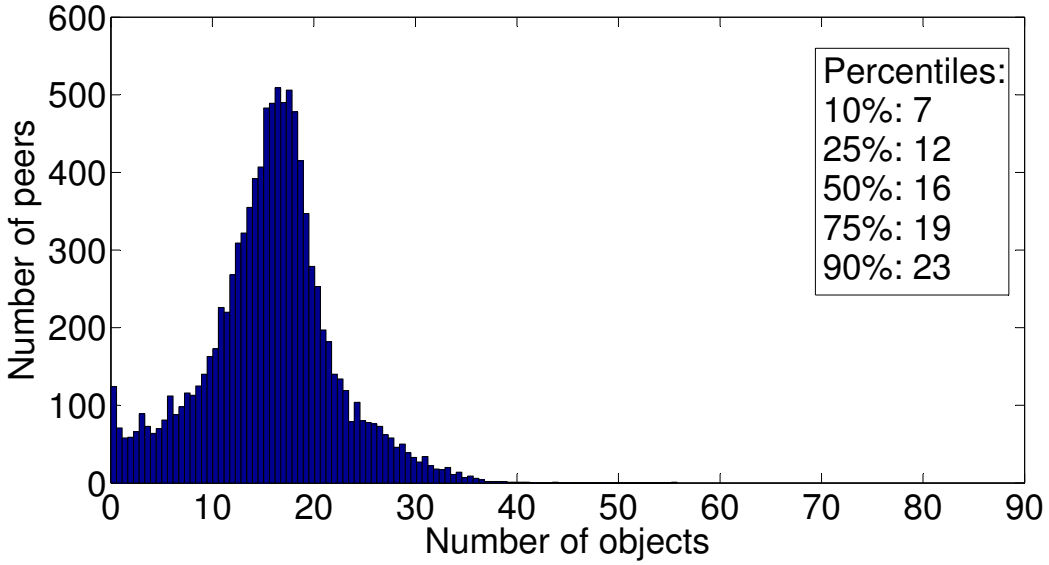


Figure 6.24: The distribution of objects over peers for overly storage, when measuring the average number of objects stored during a peer's lifetime.

Figure 6.24 shows the distribution of overlay objects over nodes in the network. The distribution has a mean and standard deviation of 15.7 and 6.4 objects per node respectively. The figure also lists the 10%, 25%, 50%, 75% and 90% percentiles for overlay storage.

1% of peers do not store any objects, which although not ideal, is a small percentage of the total number of peers.

6.7.3 Pithos fairness

Figure 6.25 shows the object distribution of Pithos, with a mean and standard deviation of 31.3 and 9 objects per node respectively. The figure shows that no peers store no objects. While all peers in Pithos store objects, the standard deviation of the objects stored is higher for Pithos (9 objects) than for overlay storage (6.4 objects), which means that Pithos is not as fair as overlay storage.

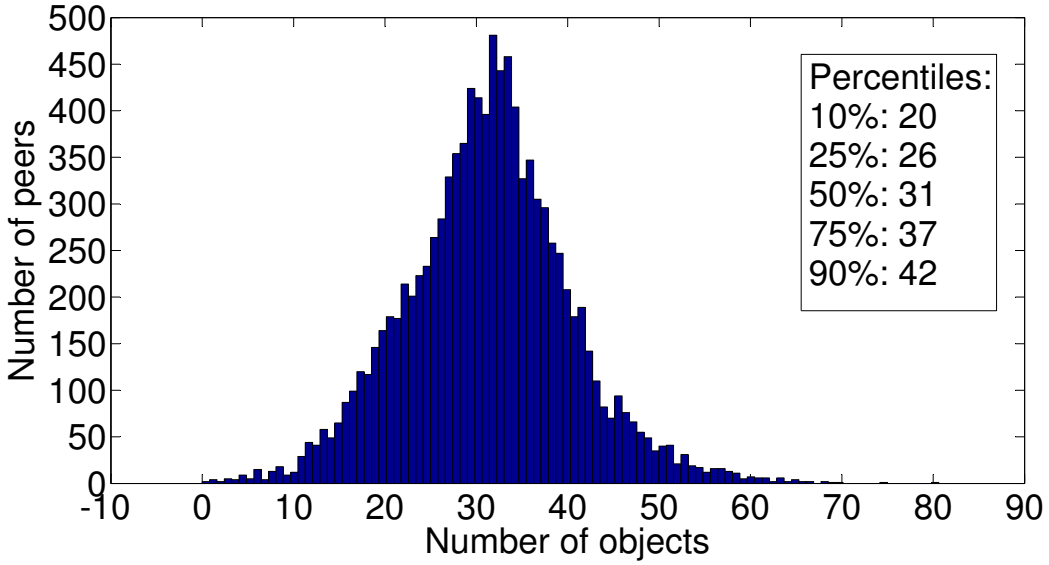


Figure 6.25: The distribution of objects over peers for Pithos, when measuring the average number of objects stored during a peer's lifetime.

6.7.4 Conclusion

What should be concluded from this section is that there does not exist a minority of peers in Pithos that store the majority of objects. All peers are required to contribute to the P2P network. It is difficult to define an empirical measure of fairness. When standard deviation is used, Pithos performs somewhat worse than overlay storage. The performance is, however, still comparable.

It should be noted that this section compared Pithos with theoretically the fairest storage system reviewed in Section 3.4. All other storage systems are expected to be as fair at best, or less fair (for example super peer storage).

6.8 Scalability

As identified in Section 3.3.1, a key requirement of P2P MMVEs is that they should be scalable. It was argued that scalability is achieved when all other requirements are met, for a large numbers of peers. The results shown thus far have been for a *sufficiently scalable* system of 2500 peers.

A scalable system should not have decreased performance for larger network sizes. For a P2P MMVE state persistency architecture, a scalable system should be as reliable and responsive for smaller networks than for larger ones, while using little or no extra bandwidth.

6.8.1 Experimental setup

To show the scalability of the Pithos design and implementation, Pithos is simulated for 10,000 peers and 400 super peers, four times more peers and super peers than previously simulated. During this simulation, a total of 15.8 million storage and retrieval requests were generated and a total of 2.4 million objects were stored.

As with the previous experiments, the following configuration parameters are chosen for reasons already discussed:

- Fast group storage is used.
- Fast group retrieval is used.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.
- Exponential object lifetime with 1800s mean and 300s TTL.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- Using Chord as overlay.
- Storing six object replicas.
- No object repair is performed.
- All requests are for local group objects.
- Medium overlay configuration is used.

The 10,400 peer simulation setup was not shown during the previous sections, because of the resources required to complete it. The simulation requires 19 hours of run time and 14 GB of RAM on an quad core Intel Core i7, 3 GHz processor. A simulation on this scale would not have been possible using anything other than Oversim's simple underlay. The efficiency of the C++ language also provides great gains in terms of both processer and memory efficiency. Although lengthy, a 19 hour simulation time is still a feasible time to performs simulations in.

Number of peers	Module	Reliability (%)	Responsiveness mean (var.) (s)	Bandwidth (Bps)	
				in	out
2600	Overall	99.70	0.192 (0.181)	1370	1380
2600	Group	97.75	0.134 (0.0629)	187	183
2600	Overlay	91.40	1.760 (0.824)	1183	1197
10,400	Overall	99.71	0.191 (0.194)	1647	1657
10,400	Group	98.19	0.134 (0.0674)	180	177
10,400	Overlay	90.06	1.960 (1.005)	1467	1480

Table 6.4: Comparison of the reliability, responsiveness and bandwidth usage of Pithos, group storage and overlay storage for 2600 and 10400 peers respectively.

6.8.2 Results

For both cases, the average data per node from the higher layer is still measured as 4 Bps sent to Pithos and 157 Bps received from Pithos.

Table 6.4 shows the scalability of Pithos for large numbers of peers. Comparing group storage, it is evident that the reliability and responsiveness of the 2600 peer case is the same as that of the 10,400 peer case. Also of note is that group storage requires no more bandwidth for larger numbers of peers. The results show group storage to be scalable.

Overlay storage, however, fares worse both in responsiveness as well as required bandwidth.

Overall, Pithos is as reliable and responsive for 2600 peers as it is for 10,400 peers, with the exception that it uses somewhat more bandwidth, which has been shown to be as a consequence of the overlay storage not scaling as well as group storage.

This section shows that Pithos is scalable for large numbers of peers in terms of responsiveness, reliability as well as bandwidth requirements.

6.9 Summary

This section initially showed the storage and retrieval performance of Pithos, without taking into account group probability, object repair or malicious peers. Fast storage was found to be sufficiently reliable for the large responsiveness gain it added. It was found that parallel storage are both more responsive and reliable than fast storage at the cost of additional bandwidth.

Overlay storage was found to be less reliable than initially thought, when taking into account its required bandwidth. Overlay storage is, however, the only way in which a peer may acquire data from outside of its group.

The responsiveness distributions for Pithos were also presented, along with a

discussion of the results. The distributions verified the methods used to implement the store and retrieve mechanisms and also showed how the underlying group and overlay storage modules relate to the overall storage performance.

It was discussed that it is important to take group probability into account when evaluating Pithos performance. By varying the group probability, it was shown that the overall performance is a weighted average of the underlying group and overlay performances.

It was shown that all peers in Pithos are required to contribute storage space to the network.

The effect of repair was shown by evaluating Pithos for various repair methods with varying node lifetimes. It was found that repair significantly increases reliability when the expected node lifetimes are small, compared to the object TTL. Repair is not required, if node lifetimes are large, compared to the object TTL.

From the evaluation of object repair, it was found that there are many factors that influence how retrieval reliability. On a basic level, retrieval reliability is directly proportional to object lifetime. From the results shown, it was found that object lifetime is, therefore, related to node lifetimes, repair rates and object TTL.

A question arises on what is acceptable reliability levels for MMVEs. There are probably types of objects in an MMVE that have to be retrieved with 100% reliability and that for those objects, any reliability of less than 100% is unacceptable. Firstly, it should be noted that the values shown in the above results were for specific churn rates and network parameters. It is theoretically possible to make the system reliability arbitrarily large, by exchanging bandwidth for reliability as shown. Higher update frequencies will lead to a more reliable system.

That said, it might never be possible to always achieve 100% reliability because of network churn. This might be due to a high current churn rate in the network. Mechanisms have to be put in place in the higher level application that can handle these scenarios. A retry might be the first solution. Another would be a reporting path for failed requests. Such a system might allow for failed requests to be reported to a group and the group replication mechanisms can then dynamically increase the repair frequency to ensure all objects are always available. A study into ensuring an arbitrary level of reliability for Pithos is left for future work.

It should be possible to design a storage system with predictable levels of reliability, to ensure correct functionality of the larger system that uses the storage system. It is, therefore, of benefit to be able to predict objects lifetimes in a distributed storage system. Of greater benefit is to be able to design a storage system to ensure required levels of expected object lifetimes. This is the focus of the next chapter. Predicting object lifetimes in finite network under churn.

After the evaluation, Pithos was found to be reliable, responsive, fair and secure. Pithos is more secure than overlay storage, although a similar safe retrieval mechanism can be employed to increase the security of overlay storage. This will, however, further increase the latency present in overlay storage. Pithos is more responsive than overlay storage. Pithos is also more reliable than overlay storage, when taking into account bandwidth used. In other words, Pithos does more with a given amount of overhead than overlay storage. Pithos was found to be less fair than overlay storage, but Pithos is by no means unfair. No peers store no data and the storage distributions did not show any long tail.

It has, therefore, been shown that Pithos satisfies all requirements initially identified.

Chapter 7

Modelling object lifetimes in finite networks under churn

The purpose of this chapter is twofold: to validate the expected object lifetimes measured using the Oversim simulation of Pithos and to provide a tool for designing distributed storage systems with expected object lifetimes. This is done by developing a statistical model for object lifetimes, which is used to verify the simulation results.

During the Pithos evaluation, it was found that user groups can be limited in size, in that the average group size is comparable to the number of required replicas. Object lifetime prediction models were found in the literature that assume an infinite network size, but none that take into account the network size.

7.1 Background and related work

To increase the time an object is available within a distributed storage system, two techniques are used: redundancy and repair. In storage systems using replication as redundancy technique, R replicas of every object are stored. The number of required replicas R is a design decision, the effect of which will be shown in Section 7.3. When all nodes containing an object's replicas have left the network, the object is no longer available. Periodic repair, which occurs at a rate of $\mu = 1/T_{\text{repair}}$, replaces all missing replicas.

This chapter mainly extends and improves upon the work by Wu, Tian and Ng [123]. The related work models three characteristics of distributed hash table (DHT) lookups under churn: expected lookup latency of various routing schemes, expected lookup overhead of various routing schemes and expected object lifetime. We focus on the object lifetime characteristic. Wu, Tian and Ng develop two models for object lifetime when exponentially distributed node lifetimes are assumed. One

for the case without object repair and one for the case with object repair. Node residual lifetimes are used when calculating expected object lifetimes without repair. In this case object lifetimes are equal to the maximum residual lifetime of the nodes the objects were replicated on.

A continuous time Markov chain, shown in Figure 7.1, is used to model the expected object lifetimes for the case with repair. The model has $R+1$ states and is

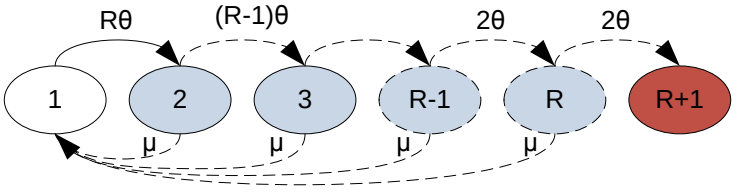


Figure 7.1: Markov chain modeling object replica number for an infinite network size with repair.

in state k when $R+1-k$ replicas are alive in the network. The system is, therefore, in state 1 when all replicas are present and in state $R+1$ when no replicas are present. The authors assume that all objects are inserted into the network with R replicas, i.e. enter the network in state 1. This initial state is colored white in Figure 7.1. If the node departure rate (the rate at which nodes leave the network) under steady state is θ and there are r replicas present in the network, the replica departure rate under steady state is $r\theta$. When all replicas are lost, the chain enters state $R+1$ (coloured dark) and is said to be "absorbed". This means it cannot return to any of the other states in the Markov chain.

In this chapter, we improve on the work by Wu, Tian and Ng [123] in two ways: firstly, the model is extended to take into account a finite network size, since there might not be sufficient nodes to replicate the data on when objects are stored or when the repair mechanism activates. Secondly, our model unifies the cases "with repair" and "without repair", to produce a single model where the effects of both might be evaluated. When our model is used with larger average network sizes, expected object lifetimes converge to those shown in the work by Wu, Tian and Ng, *using their two separate models*.

Predicting object lifetimes in a distributed storage system is grounded in reliability engineering theory. When network size is ignored, it is similar to predicting the mean time to failure (MTTF) of a set of parallel components with repair. Chun et al. [27] also uses a Markov chain to model object replicas with network churn and repair. They use a birth-death process similar to the model by Wu, Tian and Ng, but using incremental, instead of complete repair.

No literature could be found in general reliability engineering or distributed systems reliability research that deals with object lifetimes with finite network sizes.

In the rest of the chapter we first show how the model of Wu, Tian and Nig was extended. We specifically discuss how expected object lifetimes can be calculated from our extended model. We use the results of these calculations to compare the simulated results of Pithos and validate the correctness (at least for object lifetimes) of the simulation.

7.2 The model

In order to model the effects of a finite network size on the lifetime of an object, the continuous time Markov chain model introduced in Section 7.1 is expanded by adding a second parameter to every state, namely the network size. This effectively adds another dimension to the Markov chain. We also introduce ϕ to be the rate of arrival of nodes under steady state conditions.

Every state is a tuple of the form `(replicas,nodes)`. The state (r, n) represents that r replicas of an object are currently stored in a network currently containing n nodes. R is the required number of object replicas and N is the maximum number of nodes in the network. It is assumed that R replicas are always stored in the network, if sufficient space is available. If the number of nodes currently in the network n are fewer than the required number of replicas ($n < R$), only n replicas are stored. The initial states of the Markov chain are therefore all the states (R, n) as well as all the states (n, n), for $n < R$. There are therefore N initial states, one for every possible network size. The initial state is the initial network size that an object is placed in.

If there are no more replicas in the network, an object cannot be repaired and the Markov chains remains in the set of states $(0, n)$. These are said to be the absorbing states of the Markov chain and there exists $N - R + 1$ absorbing states in the model presented here. All other states out of which transition is possible are said to be transient states. It can be shown that if sets of transient and absorbing states exist, the system will always end up in the absorbing states [52]. The time to absorption can be calculated, which in the case of object storage means the time when no more objects exist in the network.

7.2.1 State transitions

There are four types of state transitions possible in the Markov model presented here:

1. A node that contains a replica departs the network.

2. A node that does not contain a replica departs the network.

3. A node joins the network.

4. An object is repaired.

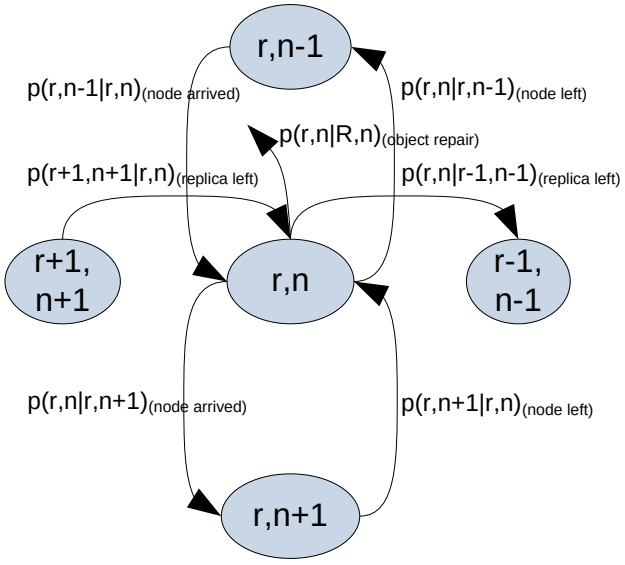


Figure 7.2: Markov chain modeling object replica number as well as network size, where $n-1 \geqslant R$ and $r > 1$

State transitions can be explained by the example in Figure 7.2. The figure shows all state transitions relative to the centre state (r, n), which represents a state with r replicas and n nodes. For the purposes of explanation, only transitions to and from the centre state are shown. Starting at state $(r+1, n+1)$, where $r+1$ replicas and $n+1$ nodes are present: If a node that contains a replica departs the network, the system moves to state (r, n): one fewer replica and one fewer node.

If a node that does not contain a replica now departs from the network, the system moves from state (r, n) to $(r, n-1)$: no fewer replicas and one fewer node. A node can join the network, which will have the model move from $(r, n-1)$ to (r, n) and another node joining the network will move the system into state $(r, n+1)$.

The periodic repair mechanism, introduced in Section 7.1 is also present. With sufficient nodes available for a full repair, the system moves from state (r, n) to (R, n). When the network size is smaller than the required number of replicas $(n < R)$, the system moves to the state (n, n) instead, since there cannot be more replicas in the system than nodes.

7.2.1.1 State transition rates

For dual states, state transition rates are characterised by moving from state ik to state jl, where i is the number of replicas in the current state, k the number of nodes in the current state, j the number of replicas in the next state, and l the number of nodes in the next state. Every state transition rate can then be expressed as a dual state transition equation.

Let θ be the departure rate of nodes under steady state. For a current state of i replicas, the departure rate of nodes containing replicas from the network is $i\theta$. This transition can be expressed in terms of the dual state transition equation:

$$p(ik, jl)_{(\text{replica left})} = i\theta, \quad \text{for} \quad j = i - 1, \quad l = k - 1, \tag{7.2.1}$$

where the next state contains one fewer replica and one fewer node than the current state.

The departure rate of nodes not containing replicas is the difference between the number of nodes currently in the network k and the number of replicas currently in the network i. The departure rate is then $(k - i)\theta$ and the transition is given by

$$p(ik, jl)_{(\text{node left})} = (k - i)\theta, \quad \text{for} \quad j = i, \quad l = k - 1, \tag{7.2.2}$$

where the next state contains one fewer node, but the same number of replicas as the current state.

The Markov model assumes the presence of a finite sized network, with some maximum number of nodes N. The departure rate of nodes from the network is dependant on the number of nodes in the network. Similarly, if a maximum network size is assumed, the arrival rate of nodes in the network can be modelled as a function of nodes *not* in the network. This creates a symmetry between the network departure and arrival rates, which creates a "force" in the Markov model that pushes the network to some average network size $\tilde{n}$, which is a ratio of θ to ϕ. A stationary average network size is required to model a steady state network.

The arrival rate of nodes can thus be modelled as $(N - k)\phi$, the product of a single node arrival rate and the number of nodes not currently in the network, as given by

$$p(ik, jl)_{(\text{node arrived})} = (N - k)\phi, \quad \text{for} \quad j = i, \quad l = k + 1, \tag{7.2.3}$$

where the next state contains one more node, but the same number of replicas as the current state.

N should be chosen sufficiently large, compared to the average network size $\tilde{n}$, to ensure that the probability of the network model ever reaching N is negligible. What constitutes a sufficiently large maximum network size will be evident from the

results presented in Section 7.3. This requires that the ratios of θ to ϕ be chosen to produce a network with an average network size significantly smaller than N.

It is assumed that the repair mechanism repairs objects once every T_{repair} time or at a rate of $\mu = 1/T_{\text{repair}}$, which gives

$$p(ik, jl)_{(\text{object repair})} = \mu, \quad \text{for} \quad j = \min(R, l), \quad l = k, \quad i \neq j. \tag{7.2.4}$$

The number of replicas in the next state is either equal to the required number of replicas or the current number of nodes, whichever one is smallest, and the number of nodes remain the same as in the current state. When no replicas are missing, no repair occurs.

No transitions other than the ones described above can occur in the presented Markov model, therefore $p(ik, jl) = 0$ for all other state transitions.

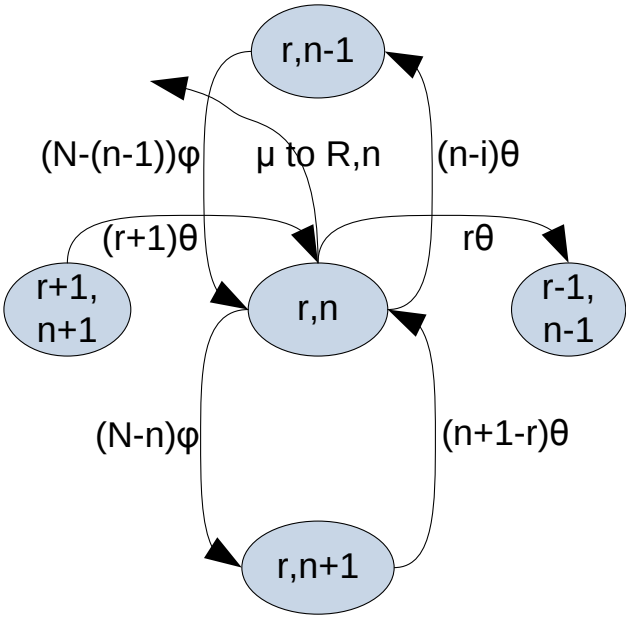


Figure 7.3: Markov chain modeling object replica number as well as network size, where $n - 1 \geqslant R$ and $r > 1$.

Figure 7.3 shows the example Markov chain shown earlier with the transition rates as described for the example. Apart from the example model developed, the initial states and absorption states have also been identified. These place bounds on the final Markov chain. An edge case has also been identified when there are too few nodes to perform a full repair. This limits the extent to which a repair can cause a state transition in the model.

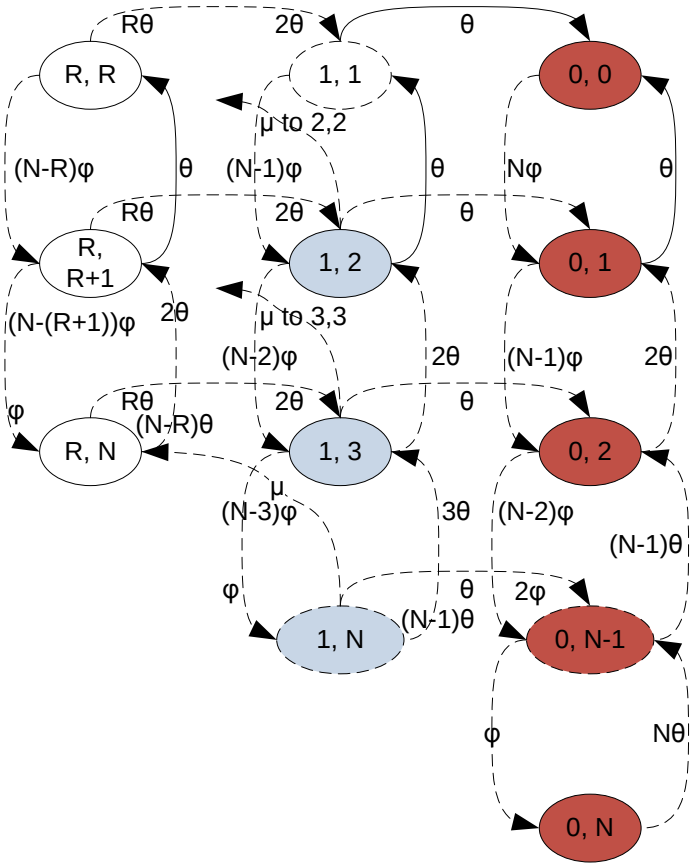


Figure 7.4: Markov chain modeling object replica number as well as network size and taking into account initial states, absorption states and the edge case of the network size limiting the replica number.

When these three aspects are taken into account, a Markov model of the form shown in Figure 7.4 is described. State transitions from one state to a consecutive state are shown by solid lines and transitions from one state to another state that contains intermediate state are shown by dashed lines. For dashed lines, to transition probabilities are shown: the transition probability from the source state and the transition probability to the destination state. The exception here is repair that has a fixed transition probability of μ. For repair, the destination state is also given.

In Figure 7.4, white states are initial states, red states are absorption states and blue states are neither initial nor absorption states. As explained earlier, initial states are states where the number of required replicas $r = R$ are inserted into the network if $n \geq R$, or where number of replicas equal to the number of nodes $r = n$ are inserted into the network if $n < R$. Absorption states are where no more object replicas exist in the network. There is, therefore, no way for objects to be repaired

from this state.

It is evident that this model is a finite state model when a maximum number of nodes are assumed. In the horizontal, it is bounded by the required number of replicas R and in the vertical it is bounded by the maximum number of nodes N. It can also be seen that this model does not have a rectangular shape, since there can never be more replicas than nodes ($r \leq n$).

7.2.1.2 Node departure rate

The node departure rate θ can be calculated from the lifetime distributions of the nodes in the network. As in the related work discussed in Section 7.1, all node lifetimes are assumed to be statistically independent and exponentially distributed. The exponential distribution is characterised by the single "rate" parameter λ.

The departure rate of nodes correspond to the failure rate (also called the hazard rate) of the node lifetime distribution [89]. An advantage of the exponential distribution is its "memoryless" property, which has a constant failure rate $h = \lambda$. For nodes with exponential lifetime distributions, the node departure rate is therefore $\theta = \lambda$.

As discussed in Section 5.1.2.3, the Pareto distribution is also used in reliability engineering to model node lifetimes and might be a better fit to online session times than the exponential distribution. The issue with using the Pareto distribution to model node lifetimes is its variable failure rate. This means that for a Pareto distribution, the average node departure rate under steady state is a function of the lifetimes of the nodes in the network $\theta = h(x)$.

In the work by Wu, Tian and Ng [123], an approximation is made for Pareto failure rate, given by $\theta = (\alpha - 2)/\beta$, where α and β are two parameters of the Pareto distribution. [1] During the calculation of expected object lifetimes, the expected lifetimes were found to be sensitive to small changes in the value of θ, which renders the approximation unusable for practical predictions using Pareto lifetimes.

7.2.1.3 Node arrival rate

For the network to be in steady state, the average arrival rate of nodes must equal the average departure rate of nodes, which gives

$$(N - \tilde{n})\phi = \tilde{n}\theta. \tag{7.2.5}$$

Making ϕ the subject of Equation (7.2.5) gives

$$\phi = \frac{\tilde{n}\theta}{N - \tilde{n}}, \tag{7.2.6}$$

[1] No derivation is provided for the approximation and the approximation could also not be found in the cited book.

which provides a value for ϕ that will produce a network with the desired average network size $\tilde{n}$, for a given θ and N.

7.2.2 Number of states

The Markov model presented, possesses a large number of states. To calculate the total number of states, two cases are identified. The first case is where the number of nodes in the network is greater or equal to the required number of replicas $k \geq R$. For every possible network size in this group, there are $R+1$ replica states. There are also $N-R+1$ possible network sizes. From this it is evident that there are $(R+1)(N-R+1)$ of these states in the Markov chain.

The second case is where the number of nodes in the network is fewer than the required number of replicas. For every possible network size in this case, the number of states are equal to the number of nodes in the network, going from R to 1.

The total number of states is the sum of the two cases and is given by

$$S = (R+1)(N-R+1) + \sum_{x=1}^{R} x \tag{7.2.7}$$

$$= R(N-R+1) + 0.5(R+1)R$$

$$= 0.5(R+1)(2N-R+2), \quad \text{where} \quad N \geq R. \tag{7.2.8}$$

For the size of network that Pithos was simulated for with 2500 nodes and 6 replicas per object, the resulting Markov chain contains 17486 states. The large number of states makes a closed form solution to the object lifetime problem intractable. Fortunately, numerical methods can be used to calculate object lifetimes for given numbers of replicas and maximum node sizes.

When calculating absorption times in the Markov chain, the number of transient states is also required. Transient states, as opposed to absorption states, are states out of which a transition can occur. In the model presented, all states containing zero replicas are absorption states. If the absorption states are removed, following the same process as with Equation (7.2.8), but using R instead of $R+1$ replica states, the number of transient states is given by

$$S_{\text{transient}} = 0.5R(2N-R+1), \quad \text{where} \quad N \geq R. \tag{7.2.9}$$

Equation (7.2.9) can also be calculated by substituting $R-1$ into R in Equation (7.2.8).

7.2.3 Calculating object lifetimes

To calculate the expected object lifetimes, a transitional rates matrix is required for the Markov model presented. The number of rows and columns of the matrix are

each equal to the total number of states in the Markov model. The transitional rates matrix contains the rate of movement from a source state ik (represented by the row) to a destination state jl (represented by the column). The transitional rates matrix contains the rates calculated in Section 7.2.1.1 for all values of $p(ik, jl)$.

Since each element in the transitional rates matrix is a rate, the sum of all the elements in a single row of the rates matrix gives the rate of moving from that state to any other state. The inverse of this rate is the expected time t_{ik} the Markov chain spends in state ik, given by

$$t_{ik} = \left(\sum_{jl} p(ik, jl) \right)^{-1}, \tag{7.2.10}$$

which is the sum of all the columns of the transitional rates matrix for a specific row.

In an embedded Markov chain, the sum of all rows in the transitional rates matrix must equal one. This is achieved by normalising each row in the transitional rates matrix $\mathbf{P}$ to produce the normalised transitional rates matrix $\hat{\mathbf{P}}$, where each row in the matrix must satisfy

$$\sum_{jl} \hat{p}(ik, jl) = 1. \tag{7.2.11}$$

Using t_{ik}, $\hat{\mathbf{P}}$ is given by

$$\hat{p}(ik, jl) = p(ik, jl) t_{ik}. \tag{7.2.12}$$

Object lifetime can be calculated by calculating the expected time to absorption of the embedded continuous time Markov chain. To do this, the normalised transitional rates matrix is partitioned into the form

$$\hat{\mathbf{P}} = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array} \right]. \tag{7.2.13}$$

Every state transition can be considered a discrete event, which allows for the continuous time Markov chain to be embedded. This allows for theory from discrete time Markov chains to be used, with the exception that events are not equally spaced in time.

$\hat{\mathbf{P}}$ is partitioned in such a way that all absorbing states are in the last rows and columns of $\hat{\mathbf{P}}$. Suppose there are a absorbing states and τ transient states in the model. The matrix $\mathbf{Q}$ is then a $\tau \times \tau$ sub-matrix of $\hat{\mathbf{P}}$ that contains all transient states. $\mathbf{I}$ is an $a \times a$ identity matrix, $\mathbf{0}$ is a $a \times \tau$ zero matrix and $\mathbf{R}$ (not to be confused with R) is a nonzero $\tau \times a$ matrix.

From $\mathbf{Q}$, the fundamental matrix $\mathbf{N}$ may be calculated as [52]

$$\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}. \tag{7.2.14}$$

Every element $n_{ik,jl}$ in the fundamental matrix $\mathbf{N}$ gives the expected number of times that the model is in the state jl, if it started in the transient state ik.

The expected time to absorption is then the product of the time spent in each state and the expected number of times a state will be entered, as given by

$$\mathbf{E[L]} = \mathbf{Nt}, \tag{7.2.15}$$

where $\mathbf{t}$ is the vector consisting of the times t_{ik} for all ik.

7.2.4 Example

To illustrate how the developed equations can be used to calculate expected object lifetimes and to clarify the use of the equations to the reader, an example system is now shown, making use of the equations developed. To make the number of states manageable in the example, a network with a maximum size of $N = 4$ nodes is used and $R = 2$ replicas are required per object. Exponential node lifetimes are assumed with a mean node departure rate chosen as $\theta = \lambda = 0.5$. The mean network size is chosen as $\tilde{n} = 2$. The repair rate is chosen as $\mu = 0.01$.

It should be notes that because of the small number of nodes, the maximum network size N is not significantly larger than the mean network size $\tilde{n}$, which reduces the accuracy of the results generated. The final results do still match intuitively between the number of replicas stored and object lifetime.

Equation (7.2.6) is used to calculate the node arrival rate using N, θ and $\tilde{n}$ as $\phi = \frac{2\times 0.5}{4-2} = 0.5$.

Equation (7.2.4) allows for the transitional rates matrix for object repair to be

calculated as

$$\mathbf{P_{repair}} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 0,4 \\ 2,3 \\ 1,3 \\ 0,3 \\ 2,2 \\ 1,2 \\ 0,2 \\ 1,1 \\ 0,1 \\ 0,0 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 2,4 & 1,4 & 0,4 & 2,3 & 1,3 & 0,3 & 2,2 & 1,2 & 0,2 & 1,1 & 0,1 & 0,0 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, \tag{7.2.16}$$

where the matrix is bordered by the dual state that each row and column represents. A row of the matrix gives the source state a column gives the destination state. In the matrix in Equation (7.2.16), it can be seen that repair occurs at a rate $\mu = 0.01$ for a state with fewer than 2 replicas, moving to a state with the maximum allowed replicas, as described earlier.

Equation (7.2.3) allows for the transitional rates matrix for node arrival to be calculated as

$$\mathbf{P_{node\ arrived}} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 0,4 \\ 2,3 \\ 1,3 \\ 0,3 \\ 2,2 \\ 1,2 \\ 0,2 \\ 1,1 \\ 0,1 \\ 0,0 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 2,4 & 1,4 & 0,4 & 2,3 & 1,3 & 0,3 & 2,2 & 1,2 & 0,2 & 1,1 & 0,1 & 0,0 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix} \end{array}. \tag{7.2.17}$$

A state transition occurs, moving to a state with one more node from any state that does not already possess the maximum number of nodes. For example, in the

first column a state transition occurs from (2,3) to (2,4) at a rate of $(N-k)\phi = (4-3)\times 0.5 = 0.5$.

Equation (7.2.1) allows for the transitional rates matrix for replica departure to be calculated as

$$\mathbf{P}_{\textbf{replica left}} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 0,4 \\ 2,3 \\ 1,3 \\ 0,3 \\ 2,2 \\ 1,2 \\ 0,2 \\ 1,1 \\ 0,1 \\ 0,0 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 2,4 & 1,4 & 0,4 & 2,3 & 1,3 & 0,3 & 2,2 & 1,2 & 0,2 & 1,1 & 0,1 & 0,0 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}. \tag{7.2.18}$$

A state transition occurs when moving from a state with more than one replicas to another state with one fewer replicas and one fewer nodes. An example transition occurs when moving from state (2,3) to state (1,2) at a rate of $i\theta = 2\times 0.5 = 1$.

Equation (7.2.2) allows for the transitional rates matrix for node departure to be calculated as

$$\mathbf{P}_{\textbf{node left}} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 0,4 \\ 2,3 \\ 1,3 \\ 0,3 \\ 2,2 \\ 1,2 \\ 0,2 \\ 1,1 \\ 0,1 \\ 0,0 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 2,4 & 1,4 & 0,4 & 2,3 & 1,3 & 0,3 & 2,2 & 1,2 & 0,2 & 1,1 & 0,1 & 0,0 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}. \tag{7.2.19}$$

A state transition occurs when from a state with more than zero nodes to a state

with one more node and no more replicas. An example transition occurs when moving from state (0,4) to (0,3) at a rate of $(k-i)\theta = (4-0)\theta = 2$.

The final transitional rates matrix may be obtained by the sum of the previous transitional rates matrices $\mathbf{P} = \mathbf{P}_{\textbf{repair}} + \mathbf{P}_{\textbf{node arrived}} + \mathbf{P}_{\textbf{replica left}} + \mathbf{P}_{\textbf{node left}}$, which gives

$$\mathbf{P} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 0,4 \\ 2,3 \\ 1,3 \\ 0,3 \\ 2,2 \\ 1,2 \\ 0,2 \\ 1,1 \\ 0,1 \\ 0,0 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 2,4 & 1,4 & 0,4 & 2,3 & 1,3 & 0,3 & 2,2 & 1,2 & 0,2 & 1,1 & 0,1 & 0,0 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.01 & 0 & 0 & 0 & 1.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.01 & 0 & 0 & 0 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0.01 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix} \end{array}. \tag{7.2.20}$$

The rates vector can then be calculated from the transitional rates matrix making use of Equation (7.2.10), which gives

$$\mathbf{t} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 2,3 \\ 1,3 \\ 2,2 \\ 1,2 \\ 1,1 \end{array} \begin{array}{c} \\ \begin{bmatrix} 0.5000 \\ 0.4975 \\ 0.5000 \\ 0.4975 \\ 0.5000 \\ 0.4975 \\ 0.5000 \end{bmatrix} \end{array}. \tag{7.2.21}$$

The transitional rates matrix is then normalised using Equation (7.2.12) and the rates vector $\mathbf{t}$ produced by Equation (7.2.21). The normalised transitional rates matrix $\hat{\mathbf{P}}$ is than partitioned as described in Equation (7.2.13) and the transient

normalised transitional rates matrix is created, given by

$$\mathbf{Q} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 2,3 \\ 1,3 \\ 2,2 \\ 1,2 \\ 1,1 \end{array} \begin{array}{c} \begin{array}{ccccccc} 2,4 & 1,4 & 2,3 & 1,3 & 2,2 & 1,2 & 1,1 \end{array} \\ \begin{bmatrix} 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.005 & 0 & 0 & 0.7463 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 & 0.25 & 0.5 & 0 \\ 0 & 0.2488 & 0.005 & 0 & 0 & 0.4976 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.4975 & 0.005 & 0 & 0.2488 \\ 0 & 0 & 0 & 0 & 0 & 0.75 & 0 \end{bmatrix} \end{array}. \quad (7.2.22)$$

The transient normalised transitional rates matrix $\mathbf{Q}$ can be calculated by simply removing all rows and columns where there are zero replicas from the normalised transitional rates matrix $\hat{\mathbf{P}}$. Take note that most rows in $\mathbf{Q}$ sum to one, but that some do not. These are where an absorption state was removed. When taking the absorption state into account, all rows sum to one.

The fundamental matrix can then by calculated by Equation 7.2.14, which gives

$$\mathbf{N} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 2,3 \\ 1,3 \\ 2,2 \\ 1,2 \\ 1,1 \end{array} \begin{array}{c} \begin{array}{ccccccc} 2,4 & 1,4 & 2,3 & 1,3 & 2,2 & 1,2 & 1,1 \end{array} \\ \begin{bmatrix} 1.1730 & 0.4080 & 0.6839 & 1.6401 & 0.1785 & 1.5058 & 0.4638 \\ 0.0112 & 1.3703 & 0.0175 & 1.4886 & 0.0090 & 0.9253 & 0.2347 \\ 0.3388 & 0.3225 & 1.3489 & 1.2965 & 0.3461 & 1.7817 & 0.6162 \\ 0.0072 & 0.4935 & 0.0189 & 1.9837 & 0.0108 & 1.2299 & 0.3114 \\ 0.1715 & 0.2751 & 0.6803 & 1.1058 & 1.1782 & 1.6377 & 0.9965 \\ 0.0054 & 0.3035 & 0.0157 & 1.2201 & 0.0138 & 1.9916 & 0.5023 \\ 0.0041 & 0.2276 & 0.0118 & 0.9150 & 0.0104 & 1.4937 & 1.3768 \end{bmatrix} \end{array}. \quad (7.2.23)$$

The expected node lifetimes, for every initial state, can then be calculated using Equation (7.2.15) and the results from Equations (7.2.21) and (7.2.23), which gives

$$\mathbf{E[L]} = \begin{array}{c} \\ 2,4 \\ 1,4 \\ 2,3 \\ 1,3 \\ 2,2 \\ 1,2 \\ 1,1 \end{array} \begin{array}{c} \\ \begin{bmatrix} 3.0177 \\ 2.0188 \\ 3.0169 \\ 2.0184 \\ 3.0150 \\ 2.0175 \\ 2.0131 \end{bmatrix} \end{array}. \quad (7.2.24)$$

Table 7.1 shows the results from Equation (7.2.24), rounded and reorganised by number of nodes and number of replicas. The results generated shows that an object for which 2 replicas were stored lives longer than an object for which a single replica was stored, which is as expected.

	2 replicas	1 replica
4 nodes	3	2
3 nodes	3	2
2 nodes	3	2
1 node		2

Table 7.1: Expected object lifetimes for the number of nodes and replicas calculated in the example.

7.3 Model results

The simulation model will now be explored for larger numbers of objects and replicas to get an understanding of how the various network parameters influence object lifetime. For all results presented in this section, the following parameter values are used: a maximum network size of $N = 120$, a required number of replicas of $R = 10$ and exponential node lifetime distributions with $\lambda = \theta = 1/1800$, which gives expected *node* lifetimes of $1/\lambda = 1800$ seconds.

It should be noted that the network topology of the practical network is not important for the model developed in this chapter. All that should be ensured is that objects are replicated and repaired as assumed in the model. The purpose of this chapter is, however, to investigate object lifetimes in Pithos groups. This is why a focus was placed on finite sized networks, since Pithos groups are often size limited. Which means that the size of a Pithos groups is comparable to the number of required object replicas. This is why a model assuming an infinite sized network is not applicable to Pithos.

During an investigation of the model, it was found that for the maximum network size N to be sufficiently large, it does not have to be many orders of magnitude higher than the average network size presented. The choice of $N = 120$ is the smallest number of nodes that no longer have an influence on the maximum average group size of $\tilde{n} = 50$ being investigated in this section. It is preferable to choose the smallest possible maximum network size to minimise the state explosion of the Markov model and thereby allowing modern desktop computers to calculate the expected node lifetimes successfully.

Figure 7.5 shows surface plots of expected object lifetimes against initial and average network sizes for a repair rate of $\mu = 1/180$ (or T_{repair} 10% of the expected node lifetime) and no repair rate. It is evident that object lifetimes decrease greatly for average network sizes smaller than 20 nodes for the case with repair, but that average network size has no effect when repair is not used. The model presented in this chapter shows a significant decrease in expected lifetime every time the average network size is comparable to the required number of replicas, for the case with repair. The figure also shows that expected object lifetimes decrease when the

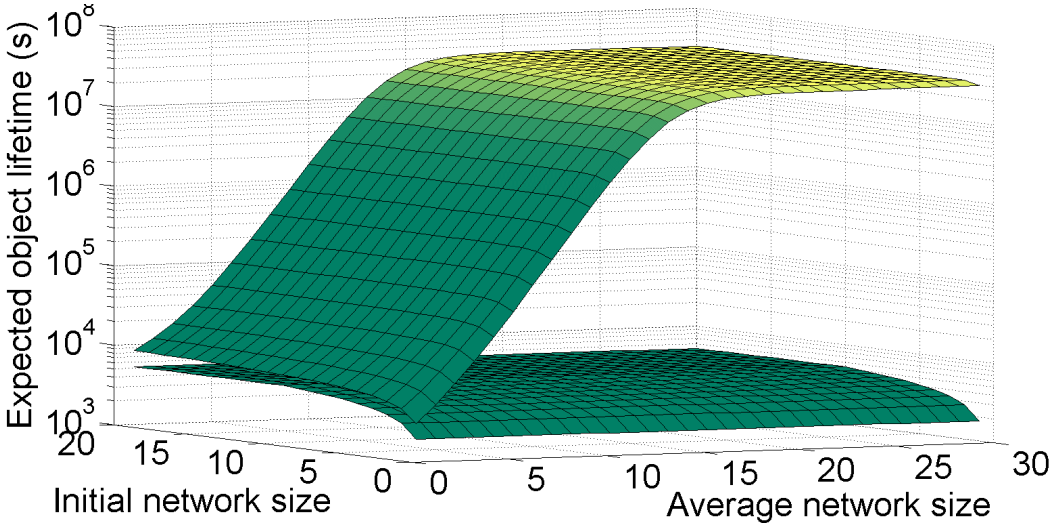


Figure 7.5: Surface plots of expected object lifetimes as functions of initial and average network sizes for $\mu = 1/180$ (above) and $\mu = 0$ (below).

initial network size is smaller than the required number of replicas. From the figure it can also be seen that initial network size has a more pronounced effect on the expected object lifetime, when no repair is used than when repair is used.

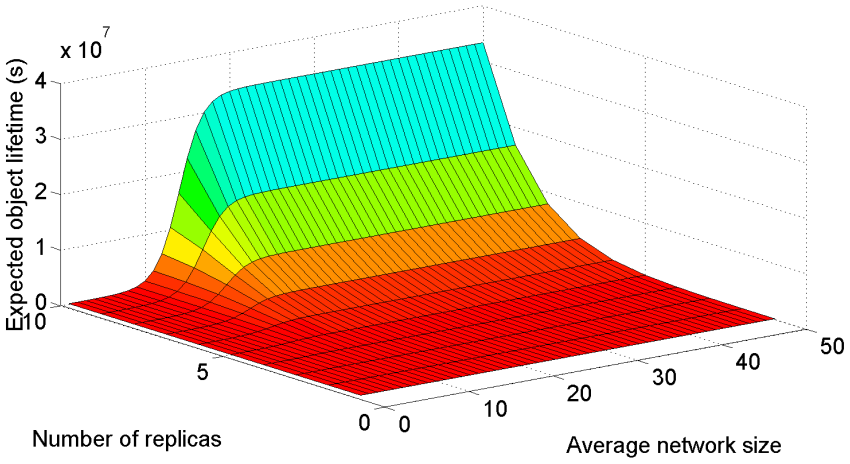


Figure 7.6: Surface plot of expected object lifetimes as functions of average network size and required number of replicas R for $mu = 1/180$.

Figure 7.6 depicts the expected object lifetime as functions of average network size and required number of replicas R for a repair rate of $\mu = 1/180$. To generate

this figure, R is swept from 1 to 10 to illustrate the effect that an increase in the number of replicas has on the expected node lifetime. The figure shows that an increase in R leads to an exponential increase in the expected object lifetime, but that this gain is limited when the average network size is small, compared to the required number of replicas.

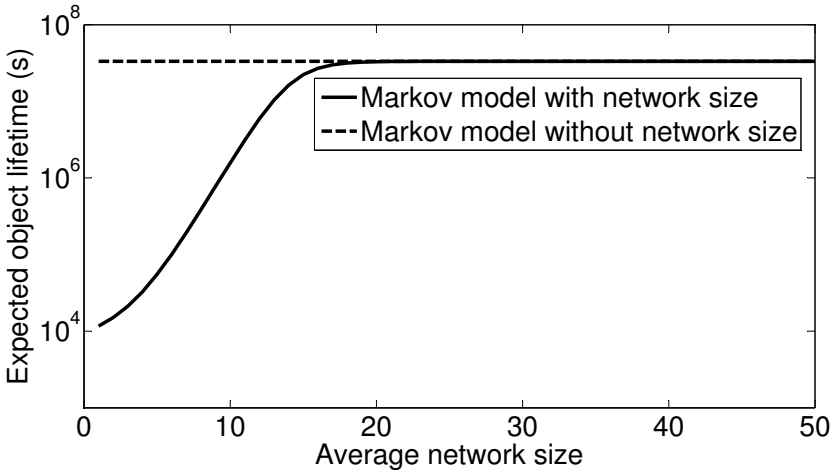


Figure 7.7: Surface plot of expected object lifetimes as functions of initial and average group sizes.

Figure 7.7 compares the model presented in this chapter with the model of Wu, Tian and Ng that assumes object repair [123]. For this figure, a repair rate of $\mu = 1/180$ and a required number of replicas of $R = 10$ were used. As shown, because the model by Wu, Tian and Ng does not take average network size into account (or initial network size for that matter) it cannot predict the object lifetimes correctly for average network sizes comparable to the required number of replicas. Figure 7.7 verifies that the extended model presented in this chapter converges to the model that assumes an infinite network size, for sufficiently large network sizes.

7.4 Comparison with Pithos simulation

To determine practical usability of the theoretical results presented in Section 7.3, a comparison was performed against the Pithos simulation [50].

7.4.1 Simulation setup

Exponential node lifetimes were used. A single *super peer* was also created and never destroyed. A single group was used, because multiple groups have different average

group sizes during different simulations, which reduces the number of measurements per average group size to below statistical significance.

As with the model results generated, 10 object replicas were used. Exponential node lifetimes with 100s means were used. The shorter node lifetimes allowed for shorter object lifetimes, which allowed for more objects to be simulated in the memory available.

During a simulation run, the Pithos test application generates random objects and require these objects to be stored in the network. Every node only generated objects for 20s. This is done to limit the total number of objects present in the system to 600,000. For numbers larger than this, the simulation would run out of memory, because of the long object lifetimes when simulating cases with repair. To allow simulation of the case of long lived objects, the Pithos simulation was deployed on a "High-Memory Quadruple Extra Large Instance" in the Amazon EC2 cloud, which contains 68.4 GB of RAM.

Object TTLs were also set to be effectively infinite (longer than the simulation run time) to ensure that any object that is destroyed in the system is destroyed as a consequence of network churn.

For the simulation run, the overlay storage module was also turned off, since only object lifetimes in group storage were investigated. This allowed for the simulation of more objects with the memory available.

As with the previous experiments, the following configuration parameters are chosen for reasons already discussed:

- Fast group storage is used.
- Fast group retrieval is used.
- Simulation length of 10,000s.
- Using the Oversim SimpleUnderlayNetwork for the physical network.
- Object sizes of 1024 bytes.
- Generating a store and retrieve request once every 5s.
- All requests are for local group objects.

7.4.2 No repair

"Box and whiskers plots" are used to compare the simulation data with the theoretical model. This allows for comparisons of mean values, but also shows how the data are distributed. In the box plot, the black striped whiskers show the minima

and maxima of the data sets. The lower and upper bounds of the boxes show the 25th and 75th percentiles respectively. The horizontal lines in the boxes show the medians of the data, and the "notches" around the medians show the 5% significance levels. The cross present in every box shows the simulation data mean and the solid line running through the boxes shows expected object lifetimes as predicted by the theoretical model.

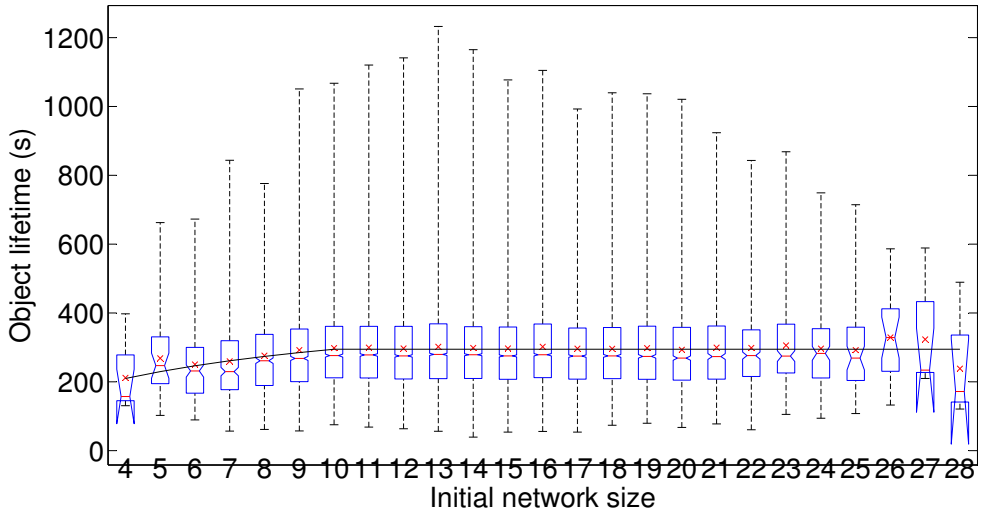


Figure 7.8: Comparison of object lifetime simulation results, as a box plot, with theoretical model results for no repair, node lifetimes of 100s and an average network size of 14 nodes.

Figure 7.8 compares the theoretical results to simulation results for the case where no repair is performed. Nodes have expected lifetimes of 100s, the number of required replicas is 10 and the average network size is 14 nodes.

The expected object lifetimes as predicted by the theoretical model pass through the crosses that show the simulation data means. The mean prediction error is 4.5s with a standard deviation of 16.5s. Comparing the mean prediction error to the predicted lifetime means, a mean prediction error percentage of 1.32% is achieved.

For initial network sizes on the edges of measured data, the simulation data might seem to deviate from the model data. However, in these ranges, only a few measurements could be made, instead of the thousand measurements made away from the maximum and minimum initial network size. The inaccuracy in these ranges can also be seen from the wide 5% significance levels.

Figure 7.9 shows the number of measurements taken for the various initial network sizes. The highest number of measurements (6855) are taken at the mean, as expected, with the number of measurements falling to 7 for initial network sizes of both 4 and 27.

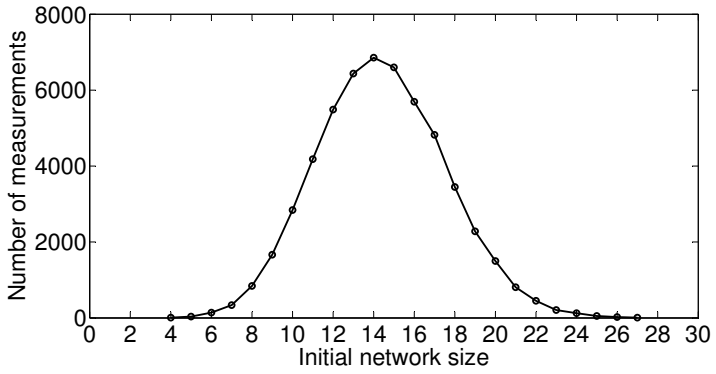


Figure 7.9: Number of simulation measurements taken for the object lifetime comparison for no repair, node lifetimes of $100s$ and an average network size of 14 nodes.

7.4.3 Repair time of 20s

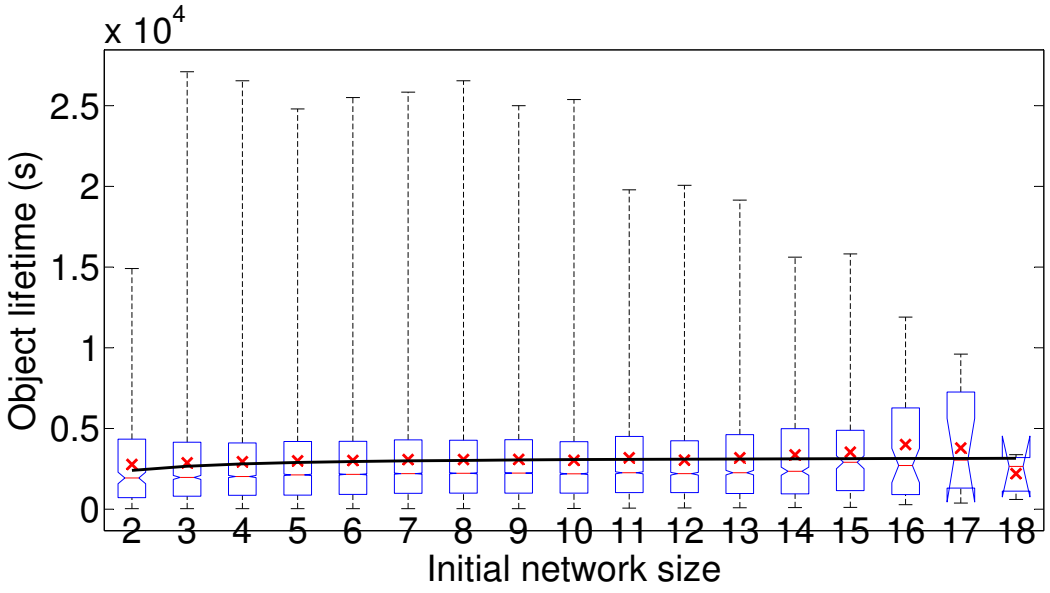


Figure 7.10: Comparison of object lifetime simulation results, as a box plot, with theoretical model results for a repair time of $20s$, node lifetimes of $100s$ and an average network size of 7 nodes.

Figure 7.10 shows simulation data and theoretical model data for the same parameters as the previous figure, but for the case with $20s$ repair time. A significant increase in object lifetimes can be observed. The mean prediction error is $139.5s$ with a standard deviation of $371.3s$. Comparing the mean prediction error to the predicted lifetime means, a mean prediction error percentage of 3.18% is achieved.

One issue that was encountered when comparing simulation data to model data was the imperfect repair scheme used in simulation. In the simulation, every T_{repair}

time, a repair of all objects in the system is initiated. In simulation, there is, however, a chance that the node that was chosen to repair an object leaves the network before the repair can complete. This reduces the effectiveness of the repair mechanism, reducing the effective repair rate. The percentage repair successes was measured during simulation and found to be 70% exactly. All repair rates in the simulation were adjusted with this number, which successfully aligned the simulation data means with the model data means.

7.4.4 Repair time of 500s

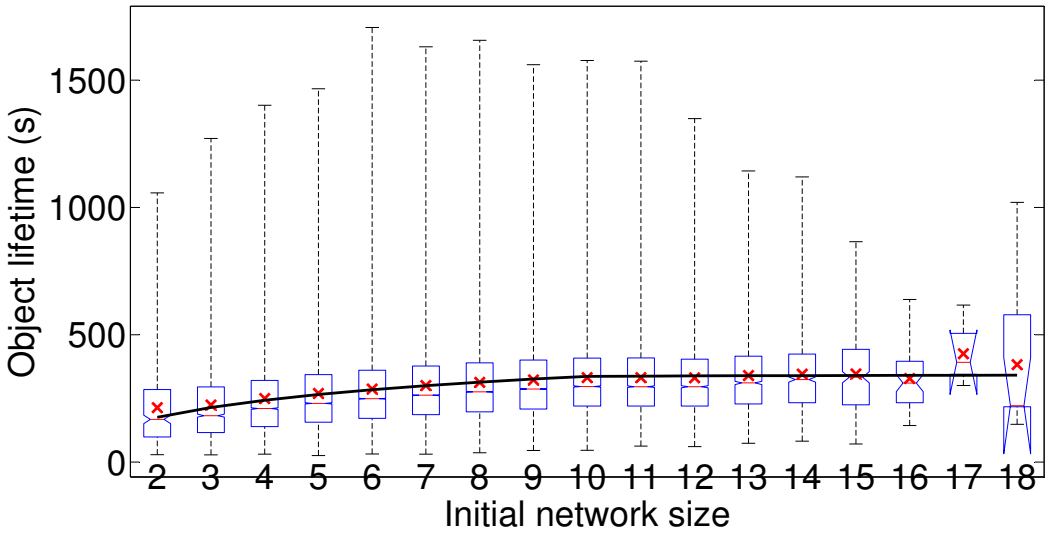


Figure 7.11: Comparison of object lifetime simulation results, as a box plot, with theoretical model results for a repair time of 500s, node lifetimes of 100s and an average network size of 7 nodes.

Figure 7.11 shows the simulation data and theoretical model data for the case with 500s repair time. The mean prediction error is 9.8s with a standard deviation of 23.9s. Comparing the mean prediction error to the predicted lifetime means, a mean prediction error percentage of 3.05% is achieved.

7.5 Conclusion

7.5.1 Summary

The Markov chain model proposed by Wu, Tian and Ng was extended to take network size into account when predicting object lifetimes. Results from the theoretical model show that when the average network size is comparable to the required

number of replicas, object lifetimes are significantly decreased as is expected. The theoretical model validated the results of the model that ignores network size, when the average network size is large, compared to the required number of replicas. The model was also shown to compare well to simulation, with predicted and simulated mean object lifetimes differing on average by 3%.

We show how to reliably predict expected object lifetimes under the assumption of finite network sizes. When it is known that the average network size is comparable to the required number of replicas, the network size should be taken into account when modelling object lifetimes in a situation with repair. The theoretical model presented here allows for the design of a distributed storage system when the various average network sizes and node lifetime distributions are known.

7.5.2 The model in practice

The theoretical model allows for the Pithos architecture to be designed with specific expected node lifetimes. It provides a guideline for the values of various parameters in the Pithos architecture, such as repair rate and required number of replicas. The expansion of the theoretical model to take into account finite networks allows for the model to accurately predict object lifetimes for small Pithos groups.

Objects in Pithos possess finite TTLs. This is the time that an object is required to exist for in the group. The object TTL will depend on the type of object, as well as the overhead required to extend the TTL of objects. If a value has been chosen for the TTL, it fixes the required object lifetime in the storage system. The object lifetime prediction model allows for the network parameters to be taken into account and values for the required number of replicas and repair rate to be chosen to ensure (to a high probability) that an object will live as long as its required TTL.

It should be noted that the Markov chain model developed in this chapter does not take malicious users into account. It is possible to have a required number of replicas in the network that produces expected object lifetimes that is sufficiently longer than the object TTL, to such an extent that object repair is not required. This scenario will lead to the number of object replicas steadily decreasing.

The lower the number of replicas in the system, the more influence a malicious user or group of malicious users can exert in the system. For example, if there are only two replicas remaining in the system and a retrieval request returns different versions of the same object, it is not possible to determine which object has been maliciously altered.

Because of this security weakness, it is recommended that object repair always be used and that the required number of replicas be made sufficiently large to enable quorum mechanisms to function correctly.

Chapter 8

Conclusions and Recommendations

As stated in the introduction: the objective of this work was to design and develop a novel state management and persistency architecture, specifically designed for P2P MMVEs that satisfies all P2P MMVE storage requirements and takes into account the requirements of the consistency architecture in which it will operate.

8.1 State consistency

In order to evaluate and compare the many consistency models described in literature, a generic state consistency model was developed.

The generic consistency model provides a common framework for describing the interaction of modules such as event ordering, event and update dissemination, object storage and interest management.

Thus, the generic consistency model developed provides a framework for the design and development of MMVEs in general. Because many aspects of the generic consistency model are trivial in C/S models, the model is perhaps more applicable to a P2P MMVE environment. This is not to say that the generic model is not able to describe the classic C/S case, just that it contains additional complexity not required to fully describe the classic C/S model.

8.2 State management and persistency

State management determines how objects in primary memory are stored and updated and requires a highly responsive system, State persistency determines how objects will be stored on persistent storage as backup for long term integrity. State persistency, therefore, does not require as responsive a storage system.

From the perspective of a state consistency architecture, five main requirements were identified for state management and state persistency: fairness, reliability, responsiveness, scalability and security.

Although a lack of a thorough comparison of different storage systems in literature made it difficult to empirically compare the existing storage systems, it was found that none of the storage systems satisfy all identified requirements [51]. We therefore developed our novel state management and persistency architecture, to fulfil all the previously identified requirements [50].

8.3 Pithos design

The Pithos use case is that of a generic storage system, supporting storage, retrieval, modification and removal. Pithos is a hierarchical distributed storage system that uses grouping to reduce latencies of requests, as is required by massively multi-user virtual environments (MMVEs).

Group storage, distance-based storage and replication are used to make Pithos responsive. Overlay storage, replication and repair are used to make Pithos reliable. Certification, replication and quorum mechanisms are used to make Pithos secure. Group storage and overlay storage are used to make Pithos fair.

Users in Pithos are divided into groups which form group storage networks. Group storage networks are $O(1)$ structured overlays that ensure one hop latencies for requests. Super peers exist to manage groups and peer join groups using a directory server. Group ledgers are by each peer to keep track of the peers and objects stored within a group. A third-party $O(\log(N))$ overlay storage implementation is used to allow nodes to retrieve data stored outside of their groups.

Pithos is implemented as an Oversim simulation that runs on the OMNeT++ network simulation framework. The simulation allowed for the evaluation of Pithos using up to 10,400 nodes, using realistic latency profiles.

8.4 Pithos evaluation

The responsiveness and reliability of Pithos was evaluated, taking bandwidth usage into account. Pithos performance was compared to overlay storage performance.

What was found in Pithos is that a trade-off exists between reliability and bandwidth. Pithos can be made highly reliable by increasing the amount of bandwidth used and reliability decreases with a decrease in bandwidth.Pithos is more reliable than overlay storage for the same amount of bandwidth used. Pithos uses a total of 1950 Bps bandwidth to achieve 99.98% reliability, while the most reliable overlay storage configuration tested achieved 93.65% reliability, using 2182 Bps bandwidth.

Pithos is also more responsive than overlay storage, with a responsiveness of 0.192s, compared with the overlay responsiveness of 1.4s.

A latency of 192ms for Pithos might seem high, but the effect of the underlay network should be taken into account. This was shown by simulating Pithos in a LAN environment, where it achieved a responsiveness of 1.6ms in a network where each hop had a 1ms delay.

When comparing group probabilities, it was found that the worst Pithos could perform was equal to the performance of overlay storage.

The performance of Pithos under invalid data attacks were also investigated for various probabilities of malicious nodes. It was found that Pithos performs consistently better than overlay storage i.t.o. reliability. As expected, using a quorum mechanism further improved the Pithos reliability. At 40% malicious users, overlay storage is 46% reliable. In comparison, when using the least bandwidth, Pithos is 60% reliable whilst it is 96% reliable when using the most bandwidth.

The distribution of objects on peers were also investigated to determine the fairness of Pithos. It was found that Pithos is comparable i.t.o. fairness to overlay storage.

The scalability of Pithos was also tested, by quadrupling the network size, increasing the number of nodes in the simulation by 300% from 2600 nodes to 10,400 nodes. The overall reliability and responsiveness remained the same at 99.7% and 0.19s respectively. The bandwidth usage, however, increased by 20% from 1375 Bps to 1652 Bps due to increased bandwidth usage by overlay storage.

It was, therefore, found that Pithos is a reliable, responsive, fair and scalable state management and persistency architecture that will be able to fulfill its role as the authoritative object store for the P2P MMVEs of the future.

Simulating for 10,400 nodes is also 10 times larger than the P2P Second Life Walkad simulation of 1024 nodes [114]. Before Pithos, Walkad is considered to be the most recent and significant contribution to the area of state management and state persistency for P2P MMVEs. The advantage of Walkad is that it is a practical implementation of a state management and persistency architecture that is already integrated into an existing MMVE. The disadvantages of Walkad are that it does not take security consideration into account; it runs on an existing file sharing network where users may have different usage profiles; it was emulated on a relatively small scale (1024 peers and 600 objects at most); and its theoretical performance is below that of Pithos. The Walkad routing time for local queries is between $O(1)$ and $O(N_c)$ depending on the cell distribution, where Pithos's routing time is fixed at $O(1)$.

8.5 Modelling object lifetime

Since the reliability of Pithos depends on whether an object is available in the storage network, the expected object lifetime has a strong correlation with the system reliability. It was therefore deemed important to be able to predict expected object lifetimes in realistic network under realistic network effects, such as churn.

A deficiency was discovered in the literature on object lifetime prediction. It was assumed that an infinite network existed into which an object is placed. This is in contradiction to the Pithos case, where size limited groups exist. Group sizes in Pithos might be comparable to the number of required replicas in the system. When this occurs, it is sometimes not possible to insert the required number of replicas in the system. Existing Markov chain models were extended to take finite network sizes into account.

Object lifetime was found to be dependant on the node departure rate, average network size under steady state, the required number of replicas and the object repair rate. The object lifetime predictions from the model were compared to object lifetimes measured in Pithos. The predicted expected object lifetimes were found to be within 3% of the measured values, which validates the correctness of the Pithos implementation.

8.6 Recommendations

8.6.1 State management and persistency

It is believed that a greater focus should be placed on whether an authoritative storage design is designed for state management or state persistency. As we discussed, it seems logical that some requirements, such as responsiveness, are not applicable to both, which might allow for different design choices.

It might be of benefit to develop a generic state categorisation, where a study is performed to identify the types of states in P2P MMVEs, along with their characteristics and how different types of states are altered. Similarities might be investigated and used to define a finite set of state types and interactions with these types. This would allow for further optimisation to distributed storage systems for P2P MMVEs.

8.6.2 P2P MMVEs

Significant research effort has now been invested to solve the remaining challenges of P2P MMVEs, with some issues, most notably security, ever present in P2P networks. Future work in P2P MMVEs should not be on further improving the various modules of state consistency, but rather in building a complete architecture.

Possible applications for such a complete P2P MMVE include:

- Development of a P2P virtual conference environment.
- Distributed social networks, since social networks usually possess natural grouping elements in the social network that may be directly transferred to the network layer.
- Free-to-play P2P MMOGs, playing to the strengths of P2P MMVEs such are responsive environments.

Responsive MMVEs are an especially interesting field, since these types of MMVEs are not currently possible in classic C/S MMVEs. All MMVEs have high latencies while playing (in the order of 200ms). This only supports specific genres of games, such as strategy games and RPGs. It is currently difficult to support first-person shooters or racing games using the MMVE paradigm, although the recently launched Planetside 2 shows progress in this direction.

Something that might have to be reevaluated is whether the P2P design paradigm still makes sense in an economy where bandwidth is becoming much cheaper and where hosting a server in the cloud is a possibility. A study on the economics of P2P versus centralised architectures would be an interesting future research topic to determine whether the reasons for moving away from a centralised architecture remain valid in the current cloud age.

Another way to approach the migration into the cloud would be to transfer what was learned from P2P MMVEs, which are distributed application, into a new field of MMVEs that exist in the cloud. It seems that much of what was learned from distributed applications remain valuable for a cloud age.

8.7 Future work

8.7.1 State management and persistency

- An empirical comparison of the various state management and persistency architectures would be beneficial. As seen in our evaluation of overlay storage, it was less reliable than originally thought. Empirical comparisons between other storage types might lead to similar discoveries.
- The interaction between state management and state persistency has not been explored in detail. Further research is required in determining when to back-up state to persistent storage and what types of state require persistent backups.

- A categorisation of different types of state will be helpful. Especially in terms of the differences in requirements. Distinctions such as ephemeral as opposed to permanent state might be investigated. The storage parameters, such as object TTL, for ephemeral state, compared to permanent state will be different. Storage system can then distinguish different types of state and treat them differently to improve overall performance.

8.7.2 Pithos

- Request distributions should be investigated. The rate at which different virtual worlds generate storage, retrieval and modification requests to the authoritative store. How regularly objects are updated, how regularly objects are created and also retrieved should be investigated. The ratios of different requests should also be investigated. It is assumed that the request profile will vary depending on the game type (World of Warcraft vs. Second Life). A better understanding of request profiles and their reliability and responsiveness requirements might allow for improvements to Pithos.

- Pithos is an authoritative object store that will function within a consistency architecture. The question arises whether there are certain aspects of the consistency framework that has to take the Pithos architecture into consideration. An example of this might be the type of interest management schemes used for event and update interest management. It can be investigates whether some interest management schemes are more appropriate to the architecture of Pithos.

- Pithos is an authoritative object store that forms part of a consistency framework, but the network topology of Pithos might be used by other aspects of the consistency framework as well. For example, two models to maintain consistency between authoritative and non-authoritative objects might be investigated. It might be possible to use an event-based consistency model for group storage, while using an update-based consistency model for overlay storage. There might be some advantages to using two levels of consistency models, for example an increase in responsiveness through the use of the event-based model in group storage.

8.7.2.1 Real-world implementation

- The next step in the Pithos implementation will be to add libraries to allow for physical network operation and then to deploy Pithos on a global scale network, such as PlanetLab. Although PlanetLab will not be able to show Pithos for

larger numbers of nodes than one or two thousand, it will show real-world performance. It will also enable the measurement of resource requirements on a per-node basis, such as processor and memory requirements, that are quantities that cannot easily be measured in simulation.

- Before a real-world implementation, a "hardware in the loop" implementation of Pithos might be done. Oversim supports real-world computers to be connected to the simulated network. This will allow for the Pithos to be tested on a computer to determine memory and CPU requirements, while still having a larger network than what PlanetLab can provide.

8.7.2.2 Remaining performance comparisons

- A study should be made to determine what are acceptable levels of retrieval reliability in MMVEs. This might tie in with the study on different object storage classes proposed earlier. Pithos might then be extended to provide assurances of reliability levels. This can be achieved by using request-retry mechanisms and by dynamically adjusting repair rates as a function of network churn.

- The performance of Pithos should be evaluated with peer and super peer movement traces captured from a virtual world, instead of the random walk movement model used.

- More work is required into testing the performance of object modification in Pithos. This requires the above-mentioned request distributions to determine how regularly objects are updated and how significant object updated are, compared to the object sizes. This may lead to improvements in the object modification design that may have specialised handling mechanisms for different types of object updated.

- Pithos should be compared to other storage systems, for example super peer storage and distance-based storage.

- The Pithos evaluation has used a fixed ratio of super peers to peers. Although group sizes from 5 to 50 nodes per group were experienced during testing, the effect of group density on storage will have to be investigated. Group traffic as a function of group density should be investigated to determine for which group densities group storage is no longer scalable.

8.7.2.3 Grouping

- Various distributed grouping algorithms should be compared to establish which work best in the P2P MMVE and Pithos context. A distributed grouping algorithm should then be selected or developed for Pithos, using avatar movement traces.
- In the evaluation of Pithos, it was seen that the best performance is achieved when more requests remain local to the group. A grouping algorithm should therefore be evaluated in terms of how well it can localise requests when actual user and object traces are used for a variety of virtual environments. The more group requests, the better the overall performance will be.
- A group-based distance-based storage scheme should be designed for Pithos, where virtual objects are mapped to the nearest group of peers.
- The group-based approach designed for Pithos should be compared to region-based approaches.

8.7.2.4 Super peers

- Super peers can also be made part of a Voronoi network, which will allow each super peer to be aware of their neighbours in the virtual world. A distributed join mechanism can then be implemented, where a single random node can be used as entry point into the network. Super peer can pass a join request on to another super peer that is closer to the joining node.
- Backup super peers will have to be implemented to ensure that super peers leaving the network does not affect group consistency. This task should be assisted by the fact that all peers already contain all group state information.
- A super peer selection scheme should be implemented for Pithos that will determine which peers are selected as super peers. Reputation schemes might be investigated to select super peers by. Super peers might also be demoted if they are found to be malicious. The number of required super peers should also be investigated, and might depend on the grouping algorithm used and the allowable group density.

8.7.2.5 Quorum

- Many enhancements can be made to the quorum mechanism. When a parallel request is performed, a single object can be requested and multiple hashes of the object can be requested from other nodes. The hash of the received object

can be compared with the hashes received from other nodes. If the received object has matches a majority of received hashes, the retrieval was successful. If not, another node that did contain the correct object might be queried. This approach will reduce responsiveness but decrease bandwidth usage.

- Quorum techniques for overlay storage can be implemented and a comparison of Pithos using quorum can then be done with overlay storage using quorum.

Appendices

Appendix A

Ensuring group consistency in Pithos

This appendix described, in chronological order, the steps taken to ensure group consistency. It provides initial findings, discusses the reasons for inconsistency and introduces the proposed solutions.

A.1 Initial approach

The initial approach to achieve group consistency was to have every peer inform all other peers when it was moving from one group to another. If a peer is removed from the network due to churn, another peer has to discover that the peer has left, which is done with timeouts. If a request times out, the peer making the request informs the group of the peer that left.

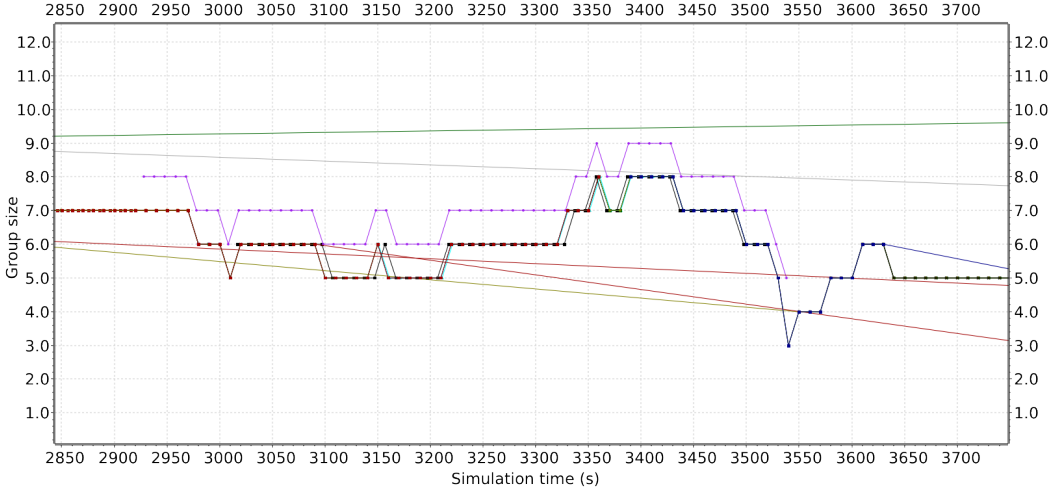


Figure A.1: Group consistency before any improvements

Figure A.1 shows an enlarged view of the perceived group size of every node in the group for the initial group consistency scheme. The lines running across the graph is due to nodes leaving the network and joining again at a later time.

From approximately 2925 s, a new peer joins the group and reports the group size as eight, when all other peers report the group size as seven. Multiple variations of these inconsistencies occurred during testing. A primary source of inconsistencies was the fact that it takes time to inform all group peers of a peer that left and that it takes time to inform all group peers of a peer that joined.

Issues also occurred because of how newly stored objects are handled. When a new peer joins the group, it starts to generate objects. The group super peer sends a joining peer a list of group peers. After this occurs, the group peers are informed of the new peer. A peer joining a group can immediately start to store objects at the request of other peers. This can happen before some peers are aware of the joining peer. The unaware peers will then get a message saying that an object has been stored on the new peer, before they are aware of the new peer. The solution was to assume that an object was generated by a valid peer and to add any unknown peer information contained in an object add message to the peers list in the group ledger.

Group inconsistency can arise because there might be some latent object add messages still enroute to peers, after the peer has already left the network and informed other peers of its leaving. This means that a peer has already left, but after it informed the group peers of its leaving and all group peers have removed the peer from memory, the group peers receive a message for an object that is supposedly now stored on the peer that just left. The mechanism mentioned above is initiated and the, now unknown, peer is again added to the peer list.

Many transient group states that led to inconsistencies were solved by the super peer storing the information of the last peer that left and the last peer that joined. Every time a new peer joins, the super peer informs the joining peer of the last peer that left. This allows the joining peer to ignore any latent object add messages received from the leaving peer. Any messages received from the last peer that left are ignored. This prevents latent messages from peers that left to affect the group view of peers.

A.2 Peer starvation problem

The mentioned improvements created the issue shown in Figure A.2, where a peer will believe that another peer has left. The group will be informed of the peer that supposedly left, but is actually still present in the group. Because all peers will

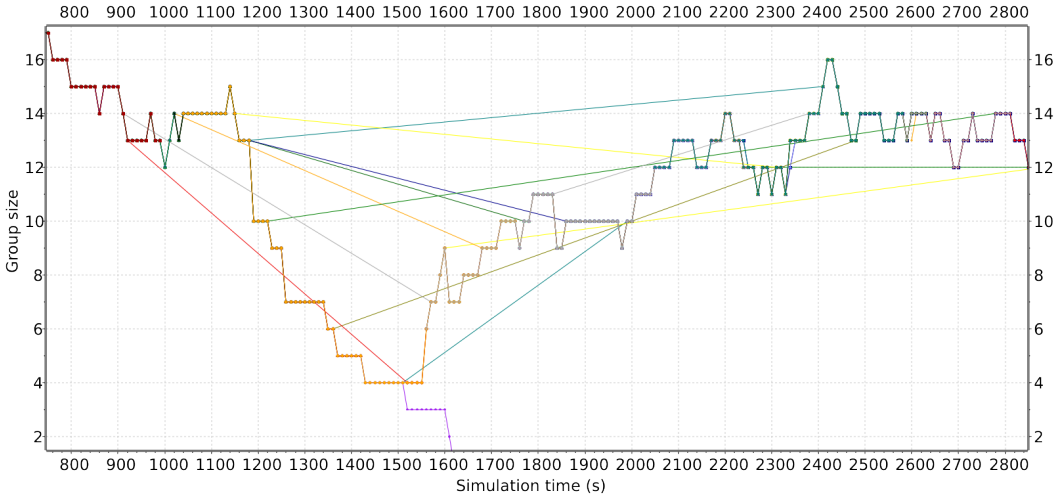


Figure A.2: Group consistency with starvation

now ignore messages from the peer that supposedly left, including the super peer, all requests sent by the peer that supposedly left will be ignored. The requests will then timeout on the ignored peer and the ignored peer will remove all group peers. This isolates the peer and prevents any requests from being serviced.

The starvation was caused by peers not responding to requests. A scenario could occur where a peer did not contain a requested object or where the network bandwidth was limited, which prevented a response from arriving at a peer before the request timeout expired. In either of these two situation, the peer making the request was under the impression that the target peer had left the network. It then informed all other peers of this.

The solution to this issue was to adjust the timeout as a function of the available network bandwidth and to make sure every node always responded to a request, whether as a success or failure. Functionality was also added to the super peer to check whether a peer that was reported as having left, actually did leave the network.

Other issues were caused when group migration was enabled. A node might leave a group to move to another, but as the node leaves some peer that has not been informed of the peer leaving would make a request form the leaving peer. The leaving peer would respond to that request, which would have the requesting peer add the peer back into the group. This could cause the situation where peers existed in multiple groups.

The solution to this was to include the group membership information in every request message. A peer's membership can be uniquely identified by the IP address and port of the group's super peer. If a peer receives a message with a different group address than its own, it ensures that that peer is not part of its group.

This also solved the case where Peer A requested an object from Peer B, just as Peer B was leaving the group. Peer B will already be in its new group when receiving the request from Peer A. Peer B will see that the request originated from outside its group, so will not add the new peer to its group. If it contains the requested object, it will however still respond with the object. When Peer A receives the object, it will detect that it was sent from another group and remove Peer B from its group ledger. In this way, the request was successfully handled and group consistency is maintained.

A.3 Final solution

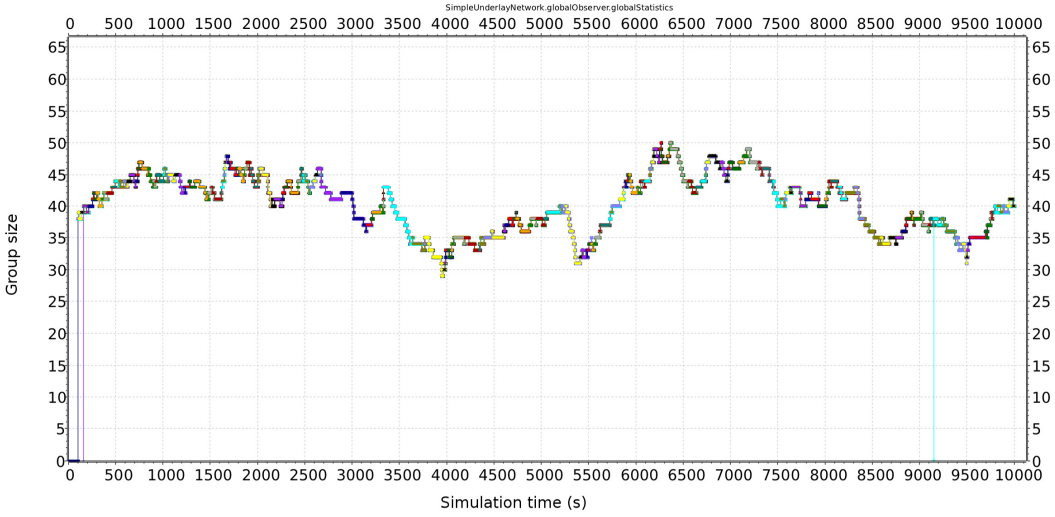


Figure A.3: Group consistency after all improvements

Figure A.3 shows the group consistency after all improvements were added for a larger group than the previous two. Even with the larger group, complete consistency is achieved.

When object lifetime was tested, all requests were stopped after a certain period of time to speed up the simulation. This created an issue where group peers did not become aware of peers leaving the group, because no peers sent requests that could timeout. This prompted the addition of keep-alive messages.

Regular keep alive messages are sent to peers. If a peer does not respond to the message, that peer is removed from the group. The time between keep alive messages can be adjusted as a function of network churn. In order to reduce the load on the super peer, each peer randomly chooses a target peer and sends a keep-alive message, as explained in Section 4.3.1.5.

Appendix B

Overlay storage configuration parameters

Parameter	Low	Medium	High
Stabilise retries	1	2	3
Stabilise delay	10s	5s	3s
Fix finger table interval	120s	10s	5s
Check predecessor delay	5s	5 s	3s
Extended finger table	no	yes	yes
Extended finger table candidate	N/A	3	4
Join retries	2	2	2
Join delay	10s	10s	10s
Routing type	iterative	iterative	iterative
Successor list size	8	8	8
Use aggressive join	yes	yes	yes
Proximity routing	no	no	no

Table B.1: Chord configuration parameters for the various test cases.

Parameter	Value
Bits Per Digit	4
Number of Leaves	16
Number of Neighbors	0
Join Timeout	20s
Ready Wait	5s
Second Stage Wait	2s
Repair Timeout	60s
Enable New Leafs	no
Optimize Lookup	no
Partial Join Path	no
Use Regular Next Hop	yes
Always Send Update	no
Use Discovery	no
Ping Before Second Stage	yes
Discovery Timeout Amount	1s
Routing Table Maintenance Interval	0s
Send State at Leafset Repair	yes
Override Old Pastry	no
Override New Pastry	no
Route Msg Acks	yes
Routing Type	semi-recursive
Minimal Join State	no
Proximity Neighbor Selection	yes

Table B.2: Pastry configuration parameters.

Parameter	Value
Lookup Redundant Nodes	8
Lookup Parallel Paths	1
Lookup Parallel Rpcs	3
Lookup Merge	true
Routing Type	iterative
Secure Maintenance	false
MinSibling Table Refresh Interval	1000s
Min Bucket Refresh Interval	1000s
Sibling Ping Interval	0s
Max Stale Count	0
k	8
s	8
b	1
Exhaustive Refresh	yes
Ping New Siblings	no
Enable Replacement Cache	yes
Replacement Cache Ping	yes
Replacement Candidates	8
Sibling Refresh Nodes	0
Bucket Refresh Nodes	0
New Maintenance	no

Table B.3: Pastry configuration parameters.

Bibliography

[1] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2006, p. 4.

[2] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, 1991.

[3] R. A. Bartle, *Designing Virtual Worlds*. New Riders, 2004.

[4] N. E. Baughman, M. Liberatore, and B. N. Levine, "Cheat-proof playout for centralized and peer-to-peer gaming," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 1–13, 2007.

[5] N. Baughman and B. Levine, "Cheat-proof playout for centralized and distributed online games," in *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, 2001, pp. 104–113.

[6] I. Baumgart, B. Heep, and S. Krause, "OverSim: A scalable and flexible overlay framework for simulation and real network applications," in *IEEE Ninth International Conference on Peer-to-Peer Computing (P2P)*, September 2009, pp. 87–88.

[7] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *IEEE Global Internet Symposium (GI) at INFOCOMM*, 2007, pp. 79–84.

[8] A. Belapurkar, *et al.*, *Distributed Systems Security: Issues, Processes and Solutions*. Wiley, 2009.

[9] S. Benford and L. Fahlén, "A spatial model of interaction in large virtual environments," in *Proceedings of European Conference on Computer-Supported Cooperative Work (ECSCW)*, 1993, pp. 109–124.

[10] M. Bergsson and J. Alberni. (2006) Eve online launches largest supercomputer in the gaming industry running on IBM server technology. CCP Games and IBM. [Online]. Available: www.eveonline.com/pressreleases/default.asp?pressReleaseID=25

[11] P. Bettner and M. Terrano, "1500 Archers on a 28.8: Network programming in Age of Empires and beyond," in *Proceedings of the Game Developers Conference (GDC)*. Ensemble Studios, 2001.

[12] A. Bharambe, *et al.*, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," *SIGCOMM Computer Communication Review*, vol. 38, pp. 389–400, August 2008.

[13] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI)*, vol. 3, 2006.

[14] A. R. Bharambe, S. Rao, and S. Seshan, "Mercury: a scalable publish-subscribe system for internet games," in *Proceedings of the 1st workshop on Network and system support for games (NetGames)*, 2002, pp. 3–9.

[15] P. Biddle, *et al.*, "The darknet and the future of content distribution," in *Proceedings of the ACM Workshop on Digital Rights Management (DRM)*, 2002.

[16] T. Blackman and J. Waldo, "Scalable data storage in project darkstar," Sun Microsystems Laboratories, Mountain View, CA, USA, Tech. Rep. SMLI TR-2009-187, September 2009.

[17] J.-S. Boulanger, J. Kienzle, and C. Verbrugge, "Comparing interest management algorithms for massively multiplayer games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2006, p. 6.

[18] E. Buyukkaya and M. Abdallah, "Data management in Voronoi-based P2P gaming," in *Consumer Communications and Networking Conference (CCNC)*, 2008, pp. 1050–1053.

[19] M. Castro, M. Costa, and A. Rowstron, "Debunking some myths about structured and unstructured overlays," in *Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, ser. NSDI'05, vol. 2, 2005, pp. 85–98.

[20] M. Castro, *et al.*, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 299–314, 2002.

[21] CCP Shadow. (2010, June) 60,453 pilots: The new Eve PCU record. CCP Games. [Online]. Available: www.eveonline.com/news.asp?a=single&nid=3934&tid=1

[22] S.-C. Chang, "Voronoi diagram based state management for peer-to-peer virtual environments," Master's thesis, National Central University, 2008.

[23] Y. Chawathe, *et al.*, "Making gnutella-like p2p systems scalable," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 407–418.

[24] F. Chen and V. Kalogeraki, "Adaptive real-time update dissemination in distributed virtual simulation environments," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, 2005, pp. 233–236.

[25] J. Chen, *et al.*, "Locality aware dynamic load management for massively multiplayer games," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, 2005, pp. 289–300.

[26] R. Chertov and S. Fahmy, "Optimistic load balancing in a distributed virtual environment," in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2006, pp. 1–6.

[27] B.-G. Chun, *et al.*, "Efficient replica maintenance for distributed storage systems," in *Proceedings of the 3rd conference on Networked Systems Design and Implementation (NSDI)*, vol. 3, 2006, p. 4.

[28] I. Clarke, *et al.*, "Freenet: a distributed anonymous information storage and retrieval system," in *International Workshop on Designing Privacy Enhancing Technologies*, 2001, pp. 46–66.

[29] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[30] E. Cronin, *et al.*, "An efficient synchronization mechanism for mirrored game architectures," in *Proceedings of the 1st workshop on Network and system support for games (NetGames)*, 2002, pp. 67–73.

[31] Daeity. (2010, August) World of Warcraft operating costs. Digital Castration. [Online]. Available: daeity.blogspot.com/2010/08/world-of-warcraft-operating-costs.html

[32] A. Dainotti, A. Pescape, and G. Ventre, "A packet-level traffic model of starcraft," in *Second International Workshop on Hot Topics in Peer-to-Peer Systems (HotP2P)*, 2005, pp. 33–42.

[33] J. Dibbell. (2007, June) The life of the Chinese gold farmer. New York Times. [Online]. Available: www.nytimes.com/2007/06/17/magazine/17lootfarmers-t.html?ei=5090&en=1676d344608cb590&ex=1339732800

[34] C. Diot, *et al.*, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, no. 1, pp. 78–88, 2000.

[35] S. Douglas, *et al.*, "Enabling massively multi-player online gaming applications on a P2P architecture," in *Proceedings of the IEEE International Conference on Information and Automation (ICIA)*, 2005, pp. 7–12.

[36] P. Druschel and A. Rawstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[37] L. Fan, "Solving key design issues for massively multiplayer online games on peer-to-peer architectures," Ph.D. dissertation, School of Mathematical and Computer Sciences – Heriot-Watt University, 2009.

[38] L. Fan, H. Taylor, and P. Trinder, "Mediator: a design framework for P2P MMOGs," in *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2007, pp. 43–48.

[39] L. Fan, P. Trinder, and H. Taylor, "Design issues for peer-to-peer massively multiplayer online games," *International Journal of Advanced Media and Communication*, vol. 4, no. 2, pp. 108–125, 2010.

[40] C. A. for Internet Data Analysis. (2012, August) The CAIDA UCSD macroscopic topology dataset. [Online]. Available: http://www.caida.org/tools/measurement/skitter/

[41] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[42] D. Frey, *et al.*, "Solipsis: A decentralized architecture for virtual environments," in *1st International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2008, pp. 29–33.

[43] P. Frey, *et al.*, "A formal specification and verification framework for time warp-based parallel simulation," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 58–78, January 2002.

[44] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, October 1990.

[45] C. Gauthier Dickey, D. Zappala, and V. Lo, "A fully distributed architecture for massively multiplayer online games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2004, pp. 171–171.

[46] C. GauthierDickey, V. Lo, and D. Zappala, "Using n-trees for scalable event ordering in peer-to-peer games," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, ser. NOSSDAV '05. New York, NY, USA: ACM, 2005, pp. 87–92.

[47] C. GauthierDickey, *et al.*, "Low latency and cheat-proof event ordering for peer-to-peer games," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, 2004, pp. 134–139.

[48] L. Gautier and C. Diot, "Design and evaluation of mimaze, a multi-player game on the internet," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, ser. ICMCS '98. Washington, DC, USA: IEEE Computer Society, 1998, p. 233.

[49] M. Ghaffari, B. Hariri, and S. Shirmohammadi, "A delaunay triangulation architecture supporting churn and user mobility in MMVEs," in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, 2009, pp. 61–66.

[50] J. S. Gilmore and H. A. Engelbrecht, "Pithos: A state persistency architecture for peer-to-peer massively multiuser virtual environments," in *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE)*, October 2011, pp. 1–6.

[51] J. S. Gilmore and H. A. Engelbrecht, "A survey of state persistency in peer-to-peer massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 818–834, May 2012.

[52] C. Grinstead and J. Snell, *Introduction to Probability*, ser. 2nd ed. American Mathematical Society, 1997, ch. 11.

[53] E. Guõmundsson (edt.). (2011, April) Eve Online fourth quarter economic newsletter for 2010. CCP Games. [Online]. Available: cdn1.eveonline.com/community/QEN/QEN_Q4-2010.pdf

[54] T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2006, p. 48.

[55] A. Harwood and E. Tanin, "Hashing spatial content over peer-to-peer networks," in *Australian Telecommunications, Networks and Applications Conference (ATNAC)*, 2003, pp. 1–5.

[56] R. Hasan, *et al.*, "A survey of peer-to-peer storage techniques for distributed file systems," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC)*, ser. ITCC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 205–213.

[57] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Consumer Communications and Networking Conference (CCNC)*, 2008, pp. 1134–1138.

[58] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "VON: a scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[59] S.-Y. Hu and K.-T. Chen, "Self-organizing spatial publish subscribe," in *Proceedings of the 8th ACM International Conference on Autonomic computing (ICAC)*. ACM, 2011, pp. 171–172.

[60] S.-Y. Hu and K.-T. Chen, "VSO: Self-organizing spatial publish subscribe," in *Fifth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, October 2011, pp. 21–30.

[61] T. Iimura, H. Hazeyama, and Y. Kadobayashi, "Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2004, pp. 116–120.

[62] J. Jardine and D. Zappala, "A hybrid architecture for massively multiplayer online games," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*, 2008, pp. 60–65.

[63] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.

[64] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th international conference on World Wide Web (WWW)*. ACM, 2003, pp. 640–651.

[65] J. Kesselman, "Server architectures for massivly multiplayer online games," in *JavaOne conference – Session TS-1351*. Sun Microsystems, 2004.

[66] B. Knutsson, *et al.*, "Peer-to-peer support for massively multiplayer games," in *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, 2004, p. 107.

[67] S. Krause, "A case for mutual notification: a survey of P2P protocols for massively multiplayer online games," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*, 2008, pp. 28–33.

[68] S. Kulkarni, "Badumna network suite: A decentralized network engine for massively multiplayer online applications," in *IEEE Ninth International Conference on Peer-to-Peer Computing (P2P)*, 2009, pp. 178–183.

[69] M. Kwok and G. Yeung, "Characterization of user behavior in a multi-player online game," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE)*, ser. ACE '05. New York, NY, USA: ACM, 2005, pp. 69–74.

[70] D. O. Lavery, *et al.*, "Dynamic deployment of custom code," US Patent 2008/0 201 707 A1, August 21, 2008.

[71] F. Lu, S. Parkin, and G. Morgan, "Load balancing for massively multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames)*, 2006, p. 1.

[72] E. K. Lua, *et al.*, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys Tutorials*, vol. 7, pp. 72–93, 2nd quarter 2005.

[73] J. C. S. Lui and M. F. Chan, "An efficient partitioning algorithm for distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 193–211, 2002.

[74] V. Massey. (2007, June) Eve online appoints in-world economist. CCP Games. [Online]. Available: www.eveonline.com/pressreleases/default.asp?pressReleaseID=34

[75] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS)*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65.

[76] M. Merabti and A. El Rhalibi, "Peer-to-peer architecture and protocol for a massively multiplayer online game," in *GlobeCom Workshops*, 2004, pp. 519–528.

[77] J. L. Miller and J. Crowcroft, "The near-term feasibility of P2P MMOG's," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games (NetGames)*, 2010, pp. 1–6.

[78] P. Morillo, *et al.*, "An adaptive load balancing technique for distributed virtual environment systems," in *Proceedings of the 15th IASTED International PDCS-03*, 2003, pp. 256–261.

[79] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 9, pp. 52–68, 2000.

[80] J. Müller and S. Gorlatch, "Rokkatan: scaling an RTS game design to the massively multiplayer realm," *Computers in Entertainment*, vol. 4, no. 3, p. 11, 2006.

[81] *Secure Hash Signature Standard*, National Institute of Standards and Technology Std., August 2002. [Online]. Available: csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[82] C. Neumann, *et al.*, "Challenges in peer-to-peer gaming," *SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 79–82, 2007.

[83] Newzoo, "MMO games, massively popular," Tech. Rep., November 2011. [Online]. Available: www.newzoo.com/press/Newzoo_Trend_Report_MMO_Games_NOV2011.pdf

[84] A. Oram, Ed., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[85] K. Pan, *et al.*, "A hybrid interest management mechanism for peer-to-peer networked virtual environments," in *IEEE International Symposium on Parallel Distributed Processing (ISPA)*, 2010, pp. 1–12.

[86] A. Pham. (2012, January) Star Wars: The Old Republic – the costliest game of all time? Los Angeles Times. [Online]. Available: latimesblogs.latimes.com/entertainmentnewsbuzz/2012/01/star-wars-old-republic-cost.html

[87] PRNewswire. (2012, May) Activision Blizzard announces better-than-expected first quarter 2012 financial results. [Online]. Available: www.prnewswire.com/news-releases

[88] S. Ratnasamy, *et al.*, "A scalable content-addressable network," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2001, pp. 161–172.

[89] M. Rausand and A. Høyland, *System Reliability Theory: Models, Statistical Methods, and Applications*, ser. Wiley series in probability and statistics: Applied probability and statistics. Wiley-Interscience, 2004.

[90] A. Rawstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001, pp. 329–350.

[91] D. Reynolds. (2010, June) The cost to make a quality MMORPG. whatmmorpg. [Online]. Available: www.whatmmorpg.com/cost-to-make-a-quality-mmorpg.php

[92] R. Rodrigues and P. Druschel, "Peer-to-peer systems," *Communications of the ACM*, vol. 53, pp. 72–82, 2010.

[93] S. Rooney, D. Bauer, and R. Deydier, "A federated peer-to-peer network game architecture," *IEEE Communications Magazine*, vol. 42, no. 5, pp. 114–122, 2004.

[94] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 188–201, 2001.

[95] G. Schiele, *et al.*, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2007, pp. 773–782.

[96] S. Schiesel. (2011, December) Star Wars MMO costs an estimated $80 million to develop. New York Times. [Online]. Available: www.industrygamers.com/news/star-wars-mmo-costs-an-estimated-80-million-to-develop/

[97] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *First International Conference on Peer-to-Peer Computing (P2P)*, August 2001, pp. 101–102.

[98] C. Seeger, *et al.*, "Area-based gossip multicast," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*, 2008, pp. 40–45.

[99] N. Sheldon, *et al.*, "The effect of latency on user performance in Warcraft III," in *Proceedings of the 2nd workshop on Network and system support for games (NetGames)*, 2003, pp. 3–14.

[100] X. Shen, *et al.*, Eds., *Handbook of Peer-to-Peer Networking*, 1st ed. Springer, 2010.

[101] A. Steed and B. Zhu, "An implementation of a first-person game on a hybrid network," in *1st International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2008, pp. 24–28.

[102] M. Steiner and E. W. Biersack, "Shortcuts in a virtual world," in *Proceedings of the ACM CoNEXT conference*, 2006, pp. 17:1–17:9.

[103] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of KAD," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC)*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 117–122.

[104] I. Stoica, *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[105] R. Süselbeck, *et al.*, "Challenges for selecting optimal coordinators in peer-to-peer-based massively multi-user virtual environments," in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, August 2011, pp. 1–6.

[106] R. Süselbeck, *et al.*, "Adaptive update propagation for low-latency massively multi-user virtual environments," in *Proceedings of 18th Internatonal Conference on Computer Communications and Networks (ICCCN)*, August 2009, pp. 1–6.

[107] G. Swamynathan, B. Y. Zhao, and K. C. Almeroth, "Exploring the feasibility of proactive reputations," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, pp. 155–166, 2008.

[108] T. Sweeney, "Unreal networking architecture," Epic MegaGames, Tech. Rep., 1999. [Online]. Available: unreal.epicgames.com/Network.htm

[109] P.-Y. Tarng, K.-T. Chen, and P. Huang, "An analysis of wow players' game hours," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*. ACM, 2008, pp. 47–52.

[110] The9, "The9 limited reports fourth quarter and fiscal year 2008 unaudited financial results," The9," Earnings Release, 2009. [Online]. Available: www.the9.com/en/pdf/The9_4Q08_ER_FINAL.pdf

[111] A. Vahdat, *et al.*, "Scalability and accuracy in a large-scale network emulator," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 271–284, December 2002.

[112] I. Van Geel. (2012, March) Total MMOG active subscriptions. MMOData.net. [Online]. Available: users.telenet.be/mmodata/Charts/TotalSubs.png

[113] A. Varga. (2012, October) OMNeT++. Technical University of Budapest. [Online]. Available: http://www.omnetpp.org/

[114] M. Varvello, C. Diot, and E. Biersack, "A walkable kademlia network for virtual worlds," in *IEEE INFOCOM Workshops*, April 2009, pp. 1–2.

[115] M. Varvello, "A peer-to-peer architecture for networked virtual environments," Ph.D. dissertation, Telecom Paristech, December 2009.

[116] M. Varvello, *et al.*, "Is there life in second life?" in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 1:1–1:12.

[117] VAST Development Team. (2010, December) VAST. [Online]. Available: vast.sourceforge.net

[118] D. S. Wallach, *Software Security – Theories and Systems*. Springer, 2003, vol. 2609/2003, ch. A Survey of Peer-to-Peer Security Issues, pp. 253–258.

[119] L. Wang, S. Turner, and F. Wang, "Interest management in agent-based distributed simulations," in *Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*, 2003, pp. 20–27.

[120] S. D. Webb and S. Soh, "A survey on network game cheats and P2P solutions," *Australian Journal of Intelligent Information Processing Systems*, vol. 9, no. 4, pp. 34–43, 2007.

[121] S. D. Webb, S. Soh, and W. Lau, "RACS: A referee anti-cheat scheme for P2P gaming," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, 2007, pp. 34–42.

[122] J. Winick and S. Jamin, "Init-3.0: Internet topology generator," University of Mishigan, Tech. Rep. CSE-TR-456-02, 2002.

[123] D. Wu, Y. Tian, and K. W. Ng, "An analytical study on optimizing the lookup performance of distributed hash table systems under churn," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 4, pp. 543–569, 2007.

[124] A. P. Yu and S. T. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, 2005, pp. 99–104.

[125] J. Yu and P. Chong, "A survey of clustering schemes for mobile ad hoc networks," *IEEE Communications Surveys Tutorials*, vol. 7, pp. 32–48, 1st quarter 2005.

[126] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California at Berkeley, Tech. Rep., 2001.

[127] H. Zimmermann, "OSI reference model–The ISO model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, April 1980.