Nomad: A P2P Distributed Storage Network

IGNATIUS T. DE VILLIERS, Stellenbosch University, South Africa
HERMAN A. ENGELBRECHT*, Stellenbosch University, South Africa

Massively multi-user virtual environments (MMVEs) can be defined as virtual environments where thousands of users are able to simultaneously interact with one another or with a virtual world. A popular implementation of an MMVE is massively multi-user online games (MMOGs/MMOs), such as World of Warcraft (WoW). In order for an MMO to be successful, it needs to be reliable, responsive, scalable, secure, and fair. In recent years, there has been much research and development surrounding distributed systems, in particular peer-to-peer (P2P) systems, although the topic of P2P MMVEs remains comparatively unexplored. Research by Gilmore and Engelbrecht in 2013 identified an aspect of P2P MMVEs that had not received sufficient attention, namely *state consistency*. This observation led to the creation of a novel state management and persistence (SMP) architecture, called Pithos, specifically designed to satisfy the key requirements of P2P MMVE storage. This study uses Nomad, a P2P distributed storage network (DSN) based on the Pithos architecture, to verify the simulated results obtained by Pithos, in a real-world environment. In order to verify the Pithos architecture, the key functional requirements of Pithos were identified and used to design a reliable, responsive, scalable, secure, and fair DSN, which was then implemented as a standalone Java application. After evaluating Nomad against Pithos, it was found that both systems satisfy the storage requirements of P2P MMVEs. Nomad was found to be reliable, responsive and secure, and although scalability and fairness were not explicitly tested, these requirements were inherently satisfied due to Nomad's scalable components and load-balancing techniques. The evaluation of Nomad further proved the accuracy of the Pithos results, and indicated that Pithos may be a suitable storage architecture for P2P MMVEs.

CCS Concepts: • **Information systems** → **Massively multiplayer online games**; • **Computer systems organization** → *Peer-to-peer architectures*; • **Peer-to-peer architectures** → Pithos.

Additional Key Words and Phrases: distributed systems, distributed storage, massively multiuser virtual environments, state consistency, state management, state persistence

ACM Reference Format:
Ignatius T. de Villiers and Herman A. Engelbrecht. 2021. Nomad: A P2P Distributed Storage Network. *J. ACM* 37, 4, Article 111 (November 2021), 24 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 Background

Massively multi-user virtual environments (MMVEs) can be defined as virtual environments (VEs) where thousands of users are able to simultaneously interact with one another or with a virtual world [6]. A popular implementation of an MMVE is massively multi-user online games (MMOGs/MMOs), such as *World of Warcraft* (WoW) [18]. In recent years, there has been much

*Project supervisor

Authors' addresses: Ignatius T. de Villiers, Stellenbosch University, Stellenbosch, South Africa, iggydv12@gmail.com; Herman A. Engelbrecht, Stellenbosch University, Stellenbosch, South Africa, hebrecht@sun.ac.za.

0004-5411/2021/11-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

research and development surrounding distributed systems, in particular peer-to-peer (P2P) systems; however, the topic of P2P MMVEs remains comparatively unexplored.

P2P architectures, due to their distributed nature, are able to solve many of the challenges faced when designing an MMO. However, the benefits of using a distributed architecture come at the cost of, amongst other things, added complexity to state consistency and scaling challenges [26]. This work reviews the key design challenges of P2P MMVE architectures. From the identified design challenges, this project identifies state consistency as an interesting challenge in P2P MMVEs. This work presents the design and verification of a P2P storage network, called Nomad, specifically designed for P2P MMVEs. The proposed system aims to satisfy the state consistency requirement of P2P MMVEs, by making use of modern development techniques combined with an established P2P storage architecture called Pithos.

1.2 Virtual Environments

In the context of this project, a virtual environment (VE) can be defined as a digital world consisting of various objects, characterised by an environment state [6]. In MMOs, VE objects can be categorised into four object types, namely *immutable objects*, *mutable objects*, *player characters/avatars* and *non-player characters* (NPCs).

Immutable objects are objects in the VE that hold a static state and cannot be altered in any way by players or NPCs. Mutable objects are objects that may be altered via interaction from a player or a NPC. Mutable objects are accompanied by game logic, which ensures any interaction is handled deterministically and follows the logic of the VE. Avatars are special objects that are player controlled and allow players to interact with and alter the VE. Game logic is specifically catered for avatars, to ensure that their interactions are secure, deterministic and abide by the game logic. NPCs are characters, typically controlled by the game logic or artificial intelligence (AI), that serve as a means to make the VE a more interactive for players [26].

1.3 Massively Multi-user Virtual Environments

In the online gaming world, MMVE are extremely popular with the implementation of MMOs. Some of the most popular MMOs to date are *World of Warcraft* [18], and *New World* [16], both of which have millions of active players [9], [12]. Design requirements that need to be taken into account when designing an MMO include:

(1) Environment logic, which allows player interaction.
(2) State consistency, which ensure all users have a consists view of the VE
(3) Object replication, which is used to distribute the game logic and resources across multiple machines.
(4) Interest Management (IM), which determines which interactions and events are important to specific users.
(5) An underlying network architecture, which allows users to connect to the VE.

1.4 Requirements of Massively Multi-user Online Games

MMOs are required to satisfy the same key requirements of classic single player games, identified by Gilmore and Engelbrecht [6] as the following: a graphics and physics engine, storage mechanisms, NPCs, game logic, sound and music and networking. One of the greatest challenges of MMVEs is that a large number of users should be able to interact with one another in the VE in a consistent and deterministic manner. This requirement is called the *consistency architecture* [6], which ensures all users share an identical view of the virtual world they inhabit.

The consistency architecture is responsible for interest management by relaying actions of other clients to those who are impacted or have a view of the manipulated, added or removed objects. This is known as *state management*. In addition, the consistency architecture is also responsible for object and information persistence. An example of this is a user's avatar and world state. Between sessions, a user's profile should remain consistent, as well as the state of the VE before session termination. This is known as *state persistence* [6].

Another way of describing the responsibilities of the consistency architecture, is in terms of a non-authoritative and authoritative object store. The non-authoritative store is responsible for ensuring consistency between users and the world, i.e. for *state management*. Users normally hold a non-authoritative view of the world and other users. The authoritative store is responsible for ensuring consistency between sessions, or, if two users disagree on world state, the authoritative mechanism is used as the true state. The authoritative object store is therefore responsible for *state persistence* [6].

1.5 Peer-to-Peer Systems

The P2P model is fundamentally different to the well known Client/Server (C/S) model. P2P networks can be defined as systems for which all content, services and other resources, are provided by the peers that form the P2P network. In contrast to a C/S networks, a peer can both serve content to other peers and request content from other peers in the network. Participants in a P2P network can access resources directly from other peers with little to no centralisation [10].

1.6 Peer-to-peer Massively Multi-user Virtual Environments

In P2P MMVE systems, individual peers contribute the required resources to the network to host itself, such as memory, processing power, storage and bandwidth [10]. Since peers act as both a server and a client, the function of the consistency architecture also becomes distributed amongst peers. This means that each peer will share the responsibility of the authoritative state of the environment; and, similarly to classic C/S MMVEs, peers are solely responsible for their own non-authoritative state [6].

1.7 State Consistency in P2P MMVEs

The *state consistency* architecture of an MMVE dictates how clients interact with one another and the VE. The state consistency architecture has two primary responsibilities, *state management* and *state persistence* [6]. In order to provide a robust solution for state consistency in P2P MMVEs, the requirements of such an architecture needs to be defined. Gilmore and Engelbrecht [6] define the key requirements for P2P MMVE storage as: (1) Responsiveness, (2) reliability, (3) scalability, (4) security and (5) fairness/load-balancing.

1.7.1 Scalability. Scalability is the most important requirement for P2P MMVEs, as it underpins all other requirements [5]. In order for a system to be considered scalable, performance should not diminish as the size of the network increases. The authors argue that overall system scalability is determined by the scalability of individual components.

1.7.2 Responsiveness. Since MMOGs are real-time systems, storage and retrieval requests need to be handled in real time, i.e. within a specific latency range. Variance in response times are required to be small. Typically in RTS games, the latency is required to be less than a second [5]. Normoyle et al. [15] argue that in multi-player platform games, latencies up to 300 ms barely affect player experience. Only when latencies above 500 ms are present, player experience is significantly altered. An Earlier study suggests that fast paced games require latencies of below 100 ms, whereas third-person strategy games can tolerate latencies of up to 500 ms [3].

1.7.3 Reliability. Storage reliability is defined as the robustness and availability of resources stored within the system. Robustness refers to resilience against network churn, whereas availability means that an object should be available to authorised peers in the network [5].

1.7.4 Security. Security in P2P MMVEs is crucial to the integrity of the network. Resources stored in the system should therefore be resilient against malicious peers that may attempt to alter their state and contradict the VE logic. The system should therefore be able to detect and identify malicious peers and maliciously altered objects [5].

1.7.5 Fairness. Fairness refers to the system's ability to balance load across multiple peers within the system. P2P systems require distributed computing, as each peer is required to contribute their available resources to the network. Ensuring fairness in the system requires load to be distributed equally among all peers within the network, which promotes high availability and resilience against network churn [5].

2 THE PITHOS ARCHITECTURE

Pithos is a reliable, responsive, secure, load-balanced and scalable distributed storage system, designed specifically for P2P MMVEs. The Pithos architecture addresses deficiencies such as lack of load-balancing, responsiveness, and scalability in available state management and persistence (SMP) architectures. It uses two different storage layers, namely **group storage** and **overlay storage**, to achieve low latency, object state management and persistence. Group storage splits the VE into various segments, and groups peers into logical geographical groups. On a network layer, group storage can be seen as a fully connected overlay implementation with $O(1)$ resource lookup. Group storage is responsible for object state management. The overlay storage is an implementation of an existing structured overlay, such as a distributed hash table (DHT), with $O(logN)$ resource lookup [6]. The overlay's main responsibility is ensuring object persistence [4]. Pithos was designed in 2013 and only evaluated through simulation.

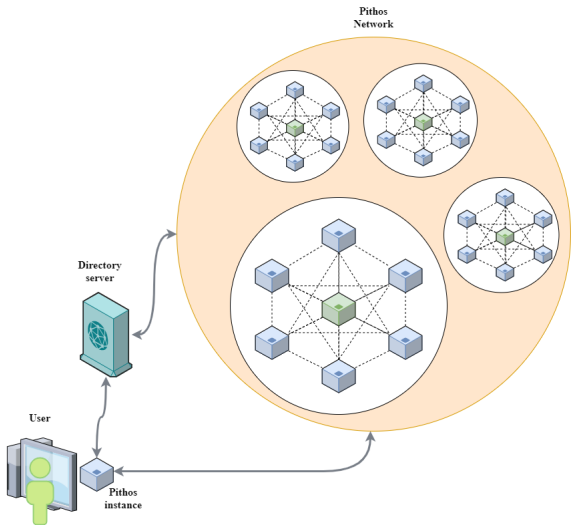

Fig. 1. Simplified illustration of the Pithos architecture.

2.1 Peer Types

Group storage consists of three peer types, namely *peers, super-peers* and a single *directory server*. A directory server provides a well-known entry point to the network, effectively allowing peers to bootstrap to the network. Peers, also called storage peers, contribute to both group and overlay storage and handle all storage, replication and repair requests. Additionally, peers are responsible for maintaining a group ledger that provides a consistent view of all objects and peers within its group. Super-peers are authoritative peers in the network, that serve a more administrative role within each group. Super-peers are mainly used to ensure object and group consistency for a group. They handle network join and leave requests and initiate object repair. Super-peers also facilitate peer migration and maintain a group ledger of their own. Figure 1 provides a simplified illustration of the Pithos architecture.

2.2 Pithos Use Cases

The Pithos architecture provides an application programming interface (API) which consists of the following four operations: **store, retrieve, modify** and **remove** [6].

- Store – A store operation is invoked by the VE logic when new objects are created. Object creation events in a VE include: (1) Initial map (VE) generation, (2) spawning of loot boxes, (3) spawning of NPCs, (4) player interactions (e.g. throwing a grenade, summoning a skeleton).
- Retrieve – A retrieval operation is invoked on all VE interactions, namely self, player-to-object or player-to-player interactions. VE interactions require object retrievals, as interactions can effect the state of objects within their AoI.
- Modify – Interactions often lead to object modification. Hence a modify operation is invoked in the case where an interaction alters the state of an object.
- Delete – A delete operation is invoked in the case where an object needs to be removed from the VE. A delete operation might be called if, for instance, an interaction causes the object to be completely destroyed.

3 NOMAD: A REAL-WORLD PITHOS IMPLEMENTATION

Nomad is a real-world Pithos implementation, which aims to implement the modules and mechanisms of the Pithos architecture in order to verify its simulated results. Nomad consists of three main peer types, namely **peers**, **super-peers** and a **directory server**. This section provides an overview of each peer type and its responsibilities.

4 NOMAD KEY MODULES

4.1 Peer Types

4.1.1 Peer. Similar to Pithos' peer module, Nomad's peers are representative of storage nodes in the network. Peer nodes are mainly responsible for group and overlay storage, but also contain client logic for joining and leaving the network, group migration, maintaining group consistency, maintaining group ledgers and executing object repair. Nomad's peer implementation deviates from the original Pithos architecture design. The architecture states that the peer module is responsible for executing maintenance tasks from the super-peer and participating in group and overlay storage. Nomad's design separates these two concerns by implementing two separate modules, namely the *peer* and *peer-storage* modules, which enables the separation of a peer's maintenance and storage responsibilities. This separation of concerns simplifies the required logic classes significantly, since storage modules are now only concerned with storage operations, and maintenance modules with maintenance operations. In Nomad, peers communicate using remote procedure calls (RPCs). The RPC protocol uses an existing transport protocol such as Transmission Control Protocol/Internet

Protocol (TCP/IP) or User Datagram Protocol (UDP) to transport messages. Nomad uses *gRPC* [23], a modern RPC framework, as its peer-to-peer communication framework.

4.1.2 Super-peer. Nomad's super-peers are authoritative nodes in the network. Similar to Pithos' implementation, super-peers are representative of a group and effectively act as gatekeepers, by determining which peers may join their group. Super-peer nodes do not participate in group storage, but in contrast to Pithos', in Nomad super-peers participate indirectly in overlay storage. This means that they act as overlay storage nodes, but do not have an interface that supports direct overlay calls. Super-peers in the Nomad network are mainly responsible for facilitating group join, leave and migration operations, maintaining object replicas, and maintaining a consistent view of peers and objects in the network.

4.1.3 Redundant Super-peers. In Nomad, every peer in the group is a redundant super-peer. This means that every peer has an *inactive* super-peer server running alongside its *active* peer server. These inactive super-peer servers are exclusively used to ensure that if the current (active) super-peer leaves the group, a new super-peer is immediately selected and able to take control of the group.

4.1.4 Directory Server. The Pithos architecture makes use of a directory server to store super-peer IP addresses, keep track of super-peer changes, facilitate new peers in joining the network, and assist peers with group migration, by providing them with relevant information. Nomad's directory server implementation partially follows Pithos' directory server design requirements, with minor additions to improve in-group and out-of-group communication. Nomad's directory server is mainly responsible for facilitating peers joining the network, group migration and storing super-peer IDs. Additionally, the directory server is used for maintaining group membership, leader election and storing public peer and super-peer data. The above mentioned mechanisms are all discussed in subsequent sections. Nomad uses Apache ZooKeeper [25] for its directory server implementation, to satisfy these requirements.

4.2 Storage Modules

4.2.1 Peer Storage. Nomad's peer-storage module can be seen as the top level storage module, which controls local, group and overlay storage, as illustrated by Figure 3. The peer storage module provides a basic interface for *retrieve, store* and *modify* operations, which are used by Nomad's storage API. Since only group storage peers are required to serve storage operations, when a peer is promoted to a super-peer, the peer storage component is disabled. Nomad supports various storage and retrieval modes, to satisfy different storage requirements. For the purpose of this and subsequent sections, it is sufficient to note that Nomad supports parallel requests.

4.2.2 Local Storage. In contrast to Nomad's design, Pithos' implementation ensures highly responsive group storage operations, by exclusively using in-memory storage. P2P MMVEs have high storage requirements, and can potentially require storage of hundreds of thousands of objects. When storing large amounts of data in-memory, peers with lower available resources may become severely impacted. The requirements of local storage will vary per implementation, since requirements depend heavily on VE object density and group AoI. Nomad therefore supports two high-level storage modes, namely *in-memory* and *disk-based* storage. In-memory storage is used when resource usage is not a concern, whereas disk-based storage is used when peers are expected to store an excessive amount of data. For in-memory local storage, a lightweight relational database (RDB), H2 [24] is used. For disk-based local storage, an efficient key-value store, RocksDB [21] is used.

4.2.3 Group Storage. According to the Pithos architecture, group storage is used to manage object state. In the Pithos evaluation, group storage was found to be highly responsive and reliable. The peer storage module is responsible for Nomad's group storage mechanism. Group storage is an $O(1)$ implementation of a structured, fully connected overlay. As illustrated by Figure 2, all group peers' local storage instances are combined to form group storage. Group storage requests therefore only require a single hop to reach any group peer, which ensures high responsiveness and constant lookup times, given the requested object is contained within the group.

4.2.4 Overlay (DHT) Storage. According to the Pithos architecture, overlay storage is responsible for maintaining a backup object store, which improves reliability of object retrieval. In the Pithos evaluation, it was found that overlay storage is highly reliable and scalable. Nomad follows Pithos' overlay storage component design. In Nomad, both peers and super-peers participate in the overlay storage network, in order to ensure an even higher degree of reliability. This is contrary to Pithos' architecture, where super-peers do not participate in any storage. Even though super-peers contribute their storage resources to the overlay network, they do not support any higher layer overlay requests. Nomad uses TomP2P [2], a Kademlia [11] based DHT, for its overlay storage component.

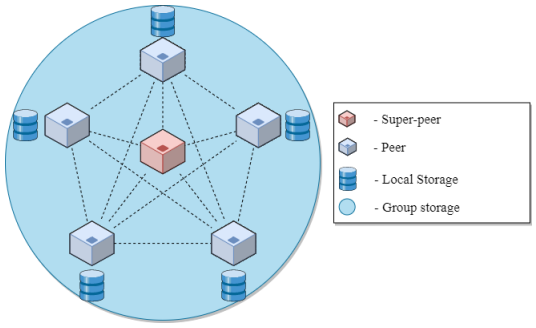

Fig. 2. Nomad group storage architecture

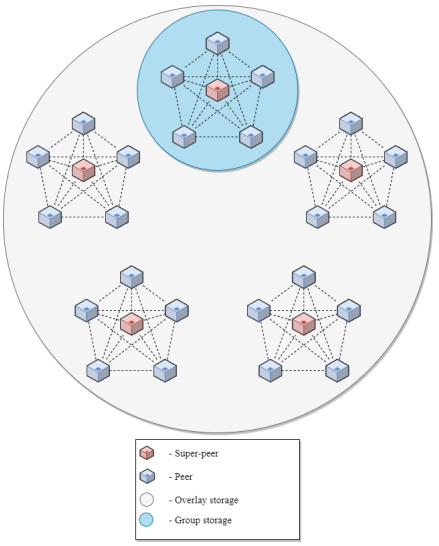

Fig. 3. Nomad storage architecture

4.2.5 Group Ledgers. Nomad peers and super-peers make use of an abstract data structure, called group ledgers, in order to keep track of all peers and objects within a group. Peer and super-peer group ledgers store exactly the same information, i.e. no discernment between the two ledgers can be made According to the Pithos architecture, group ledgers contribute to highly responsive group storage operations, by physically storing the object information within the ledger. All objects in Pithos are effectively stored in-memory. Pithos' group ledgers therefore have two responsibilities: in-memory storage and determining which peers store which objects. Nomad peers make use of a separate local storage component to store objects, whereas the group ledger is exclusively used to keep track of peers and objects within the group. This separation of concerns simplifies Nomad's group ledger implementation. Therefore, instead of storing the entire object, the *object ledger* only stores object IDs and a *metadata object*, which represents the peer ID and a last-modified date. Similarly, *peer ledgers* only store peer IDs and a *metadata object*, which represents the object ID and

a last-modified date. The last modified date contained in each entry's metadata is used to remove expired data from the group ledger. Nomad relies on more mature and efficient technologies for storage, and is only required to maintain IDs in the ledger.

5 NOMAD KEY MECHANISMS

5.1 Grouping Mechanism

Pithos-based systems make use of group-based storage. Nomad therefore provides two logical grouping mechanisms: either *random grouping* or *Voronoi grouping*.

5.1.1 Random Grouping. When Nomad's random grouping mechanism is used, peers are grouped sequentially. This means that a peer joining the network is directed towards the first sequential group that has an available peer position. Availability is determined by the *max peer* configuration property of the group super-peer. Super-peers therefore determine whether a peer can join their group or not. If no group has an available peer position, a new group is formed and the peer is promoted to a super-peer. Similarly, when all peers leave a group, the directory server removes the group from its cache to ensure no peers attempt to join a non-existent group.

5.1.2 Voronoi Grouping. Nomad uses a *Voronoi grouping* [8] mechanism, to logically split the world into super-peer oriented sections. A Nomad Voronoi map consists of finitely many points, called *site points* which are representative of super-peer locations. Each site point has a corresponding region, called a *Voronoi cell*, which is representative of a group's AoI. When Voronoi grouping is enabled, Nomad peers and super-peers construct their own Voronoi map, using site points from the directory server. A Voronoi map therefore segments the VE into a series of polygons, centred on super-peer locations. Figure 4 illustrates the Voronoi grouping process.

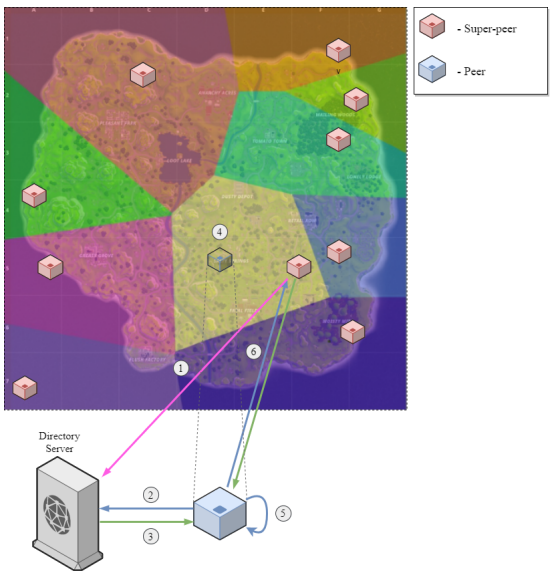

Fig. 4. Nomad Voronoi grouping mechanism, visualised using an online Voronoi map generator [1], and an arbitrary *Fortnite* map [22]

(1) When Voronoi grouping is enabled, Nomad super-peers store their hostname and VE location to the directory server. The directory server essentially maintains a cache of all Voronoi site points.
(2) Each peer requesting to join the network connects to the directory server and requests all super-peer locations, i.e. Voronoi site points.
(3) The directory server responds to the request with a list of super-peer IDs and Voronoi site points, which is used to construct the peer's Voronoi map.
(4) After the Voronoi map has been generated, a peer uses its own location to determine which segment of the Voronoi map it occupies.
(5) The segment site point, which contains the super-peer hostname, is then extracted and used for bootstrapping to the group's super-peer.
(6) The peer then follows the usual join process.

5.2 Super-peer Selection Mechanism

In order to assign a super-peer to each group, a leader or super-peer selection mechanism is required, to ensure that super-peers are selected deterministically. Nomad uses its directory server to select super-peers, since it maintains a consistent cache of the entire network. The directory server uses a sequence-based super-peer selection algorithm to establish peer hierarchy. This means that at any given time, the "first" peer in the group is always the super-peer.

5.3 Group Formation

In order to form groups, Nomad relies on its group configuration, grouping mechanism, and leader election strategy. Peers joining the network are directed towards the group that the internal grouping mechanism has determined it should join. Availability is determined by the *max peer* configuration property of the group super-peer. Super-peers therefore determine whether a peer can join their group or not. If no group has an available peer position, a new group is formed and the peer is promoted to a super-peer. Similarly, when all peers leave a group, the directory server removes the group from its cache to ensure no peers attempt to join a non-existent group. It is therefore clear that peers joining and leaving the network impact the number of groups within the Nomad network.

5.4 Group Join Mechanism

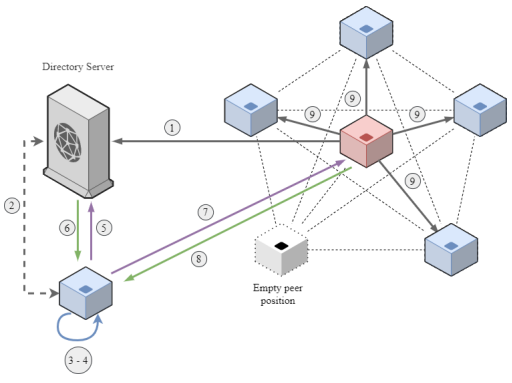

Fig. 5. Nomad group join mechanisms

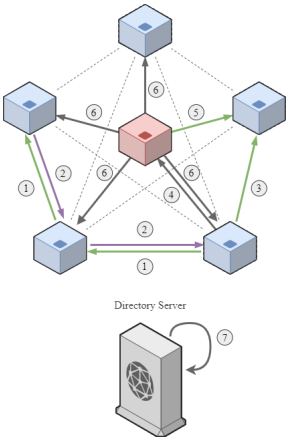

Fig. 6. Nomad leave mechanism

Super-peers represent groups within the Nomad network, and act as gatekeepers, deciding which peers may join the network. When a super-peer is elected in the network, it shares its bootstrap hostname with the directory server. On node start-up, peers automatically connect to the directory server, specified in the application configuration. After a connection is established with the directory server, the peer's client logic takes over in order to determine which group should be joined. If Voronoi grouping is enabled, the group to join is determined by the Voronoi grouping mechanism. If no grouping algorithm is used, the client logic will select the first available group to join. After determining the group to join, a request is sent to the directory server containing the super-peer ID. The directory server responds to the request with the super-peer hostname. The peer then sends a join request to the super-peer, which can either reject or accept the peer into its group. If no available group positions exist, the peer will be promoted to a super-peer and a new group will be created.
Figure 5 illustrates the join procedure, which allows peers to join the Pithos network.

(1) A super-peer shares its bootstrap hostname with the directory server. This can be used to join that super-peer's group. The super-peer server hostname is stored to the directory server.
(2) A new peer establishes a connection with the directory server, and sends a join request.
(3) The peer's internal directory server client logic ensures the peer is added to the directory server's group cache and determines the group to join (results may vary depending on grouping mechanism used). The peer rules out groups that do not have any available peer positions.
(4) The peer initialises its overlay storage component and bootstraps to the overlay network.
    (a) The peer requests a DHT hostname from one of its potential group members, which it uses to bootstrap to the overlay network.
    (b) If no other peers exist in the network, the peer becomes the initial bootstrap node.
(5) The peer then requests the corresponding super-peer hostname from the directory server.
(6) The directory server responds to the request with the hostname of the super-peer of the group the peer should join.
(7) The peer sends a join request to the super-peer, containing peer information, such as server hostnames and its VE location.
(8) The super-peer decides whether it wants to accept or reject the peer, based on the current group size and the peer's position in the VE.
    (a) If the super-peer accepts the peer, it provides the peer with a list of peers and objects in the group. The peer uses this information to populate its own group ledger.
    (b) If the super-peer rejects the peer, the peer will send a join request to a neighbouring super-peer.
    (c) If all super-peers reject the peer, no available positions exist and the peer is promoted to a super-peer.
(9) (Assuming the peer was accepted) The super-peer notifies all peers within the network that a new peer was added.

5.5 Group Leave Mechanism

If a peer ends its session, either gracefully or unexpectedly, a leave mechanism is required to maintain group consistency. If a peer leaves the network gracefully, it notifies the super-peer that it is leaving the network. The super-peer then broadcasts to the rest of the group that the peer has left. In some cases a peer might leave the network unexpectedly, which may result in group inconsistency. To ensure group consistency in Pithos, peers send keep-alive messages, in the form of network *pings*, to each other. If a keep-alive message is not acknowledged with a *pong* response,

a special leave mechanism attempts to verify whether a peer has indeed left the network. [6]. When a peer leaves the network, data stored to the directory is automatically removed from the directory server cache if the peer leaves the group. This includes the peer's group storage hostname, overlay storage hostname, peer hostname and redundant super-peer hostname. Figure 6 illustrates the group leave procedure, which ensures group consistency if a peer leaves the group unexpectedly.

(1) A peer routinely sends a keep-alive ping request to another random peer within its group.
(2) If a peer receives a keep-alive ping, it acknowledges the message with a pong.
(3) If a peer does not acknowledge the keep-alive ping within a specific time window, the peer is considered to have left the network.
(4) The original sender of the keep-alive ping notifies the super-peer that a peer is unreachable.
(5) The super-peer sends a ping of its own to the peer in question, to verify that the peer has actually left the network.
(6) If a peer does not acknowledge the super-peer ping within a specific time window, the super-peer informs all peers that the peer has left the group.
(7) The leaving peer is removed from all group ledgers and ephemeral data, like the peer's group storage hostname, overlay storage hostname, peer hostname and redundant super-peer hostname, and is automatically removed from the directory server and group caches.

5.6 Super-peer Leave Mechanism

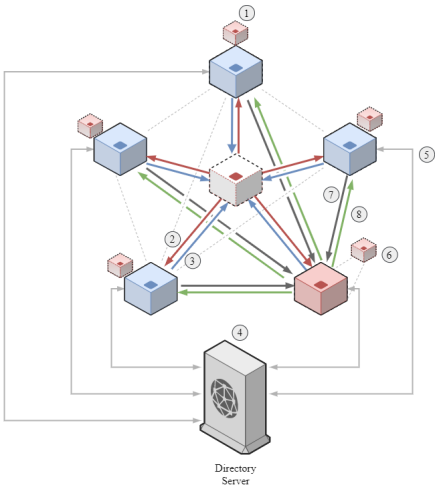

Fig. 7. Illustration of Nomad's super-peer leave mechanism

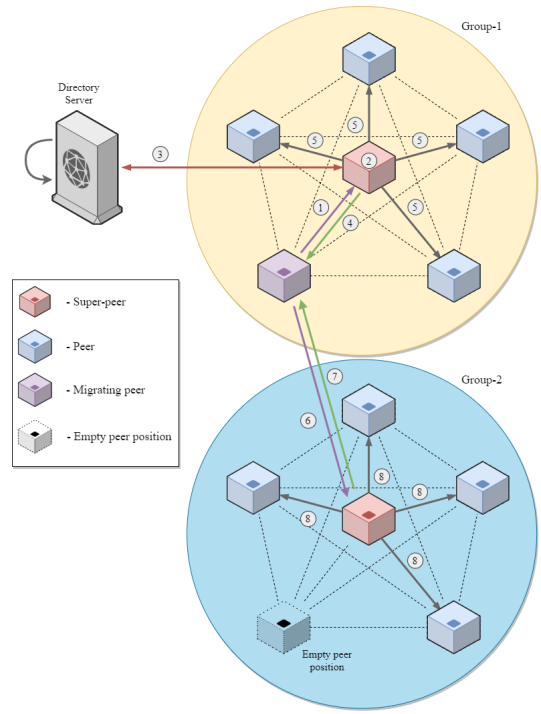

Fig. 8. Nomad group migration mechanism

Similar to normal peers leaving the network, super-peers can also leave the network due to network churn. As super-peers are representative of a group, a new leader is required, in order to retain group functionality. Nomad uses redundant super-peers to streamline the super-peer re-election

process. Figure 7 illustrates Nomad's super-peer leave procedure and the process of re-electing a new super-peer.

(1) Every peer in the group has a super-peer server running in the background. The super-peer mechanism is disabled and the redundant super-peer servers do not receive any traffic. The directory server caches all active and redundant super-peer hostnames per group, and uses these for leader selection. Each peer in the group also listens for a promotion message, which will indicate that it is the new super-peer.
(2) If the super-peer ends its session and wishes to leave the group, it notifies all peers in the group that it is about to leave.
(3) Every peer has to acknowledge the super-peer leave before the handover can be finalised. If a peer does not acknowledge the super-peer leave command, the request will be re-sent to the peer for a maximum of 5 times. If the peer still does not respond, it will no longer be able to participate in the group.
(4) After the previous super-peer has left the network, the directory server automatically selects a new leader. Each peer contacts the directory server, requesting the id of the new super-peer. Since Nomad's leader selection is facilitated by the directory server, peers are dependent on the directory server for new super-peer information.
(5) The directory server responds with the id of the super-peer.
(6) The new super-peer is notified by its internal listener that it has been selected as the new super-peer and immediately takes lead of the group. The peer stops all group related services and activates all super-peer related services.
(7) Every other peer in the network will send a simplified join request to the new super-peer, to ensure that a good connection is formed with the new super-peer. The re-join strategy also ensures that all peers maintain a consistent view of group changes and objects that might have been added during the leader re-selection.
(8) The new super-peer accepts all the previous group peers and sends its group ledger to all peers to ensure ledger consistency. All conflicts are overridden by the super-peer's ledger.

5.7 Group Migration Mechanism

Peers move around in a VE, which mean they can cross group AoI boundaries frequently. This requires an efficient migration mechanism that will not severely affect the user's experience. One of the weaknesses observed with the Pithos migration mechanism is that it relies heavily on the directory server in order to migrate peers to the correct geographical groups, which at some point will become a bottleneck. To solve this, Nomad delegates the group migration responsibilities to the super-peer of each group. This simplifies the required logic of the directory server to a simple cache mechanism. Because Nomad's directory server is simply a caching mechanism for important group and peer information, no additional logic is required server-side to facilitate group migration. Migration logic is implement client-side, and is the responsibility of each super-peer in the network. Figure 8 illustrates Nomad's group migration procedure, which allows peers to migrate from one geographical group to another.

(1) A peer routinely sends positional updates to its super-peer.
(2) The super-peer server evaluates the peer's position and responds with the super-peer ID for the group within which the peer's position falls.
  (a) If the super-peer ID matches the peer's current group super-peer ID, the super-peer simply responds with an ACK. The peer is still in the correct group and does not need to be migrated.

(b) If the super-peer ID does not match the peer's current group super-peer ID, the peer is no longer in the correct group. The peer needs to be migrated.
(3) The super-peer's internal directory server client logic, which makes use of the grouping mechanism, determines which group the peer should be migrated to. If the peer cannot be migrated to a neighbouring group, the peer is promoted to a super-peer.
(4) The super-peer responds with a new group name and super-peer hostname for the group which the peer should migrate to.
(5) The peer sends a leave request to its current super-peer. Once the peer notifies the super-peer that it has left the group, the peer truncates its group ledger and local storage.
(6) The super-peer notifies all peers in the group that the peer has left.
(7) The peer sends a join request to the new super-peer. (The join request procedure is followed)
(8) If the new super-peer accepts the peer, it provides the peer with a list of peers and objects in the group. The peer uses this information to populate its own group ledger.
(9) (Assuming the peer was accepted) The new super-peer notifies all peers within the network that a new peer was added.

5.8 Replication Mechanism

Replication is used in Nomad to ensure object availability and to allow the use of a quorum mechanism to verify objects. Nomad supports a configurable replication factor. For every object stored in the Nomad network, $\mathcal{R}$ object replicas are created. Replicas are randomly distributed amongst all the peers within the group in order to promote fairness amongst peers in terms of resource usage.

5.9 Repair Mechanism

As Nomad is intended to be used in a high churn environment, peers leaving the network will lead to object replicas being destroyed. Nomad therefore requires a repair mechanism to ensure $\mathcal{R}$ object replicas are always available. Nomad's repair mechanism is identical to Pithos' repair mechanism, and consists of *periodic (scheduled) repair* and *leave repair*.

5.9.1 Periodic Repair. During scheduled repair, a super-peer periodically checks that $\mathcal{R}$ object replicas are stored in the group, by checking its group ledger. If an object is found to have insufficient replicas, a repair request is sent to peers that do not already store the object.

5.9.2 Leave Repair. Leave repair is initiated when a peer leaves the group, and object replicas are destroyed. During leave repair, super-peers pre-emptively create new replicas of all objects stored by the peer leaving the group. Using leave repair ensures that $\mathcal{R}$ object replicas are always available if peers leave the group gracefully. The repair mechanism can therefore be seen as an extension of the leave mechanism.

5.10 Quorum Mechanism

Nomad's quorum mechanism is similar to Pithos' quorum mechanism and can be described by the simple quorum formula, $(\mathcal{R}/2)+1$, where $\mathcal{R}$ is the replication factor. The mechanism is implemented within the group storage module, and when active, Nomad executes multiple object retrievals from unique peers. The quorum mechanism compares objects and verifies that the object replicas are consistent. The object value that occurs the most amongst all responses is considered to be the true object value.

5.11 Nomad Storage/Retrieval Procedure

This section provides the required detail to describe how Nomad satisfies the implementation use cases of object **storage, retrieval, modification and updates**. Similar to Pithos, Nomad provides a representational state transfer (REST) API to satisfy all use cases. This means user traffic is generated by HTTP requests.

5.11.1 Store. Nomad has two storage modes, namely *fast* and *safe* storage. In fast storage mode, Nomad's peer-storage module sends a storage request to multiple group storage peers and to overlay storage. The first successful response is propagated to the higher layer. Fast storage is much more responsive than safe storage and no less reliable, however, it does lack security in the presence of malicious peers. In safe storage mode, Nomad's peer-storage module sends a storage command to multiple group storage peers and to overlay storage. Only after all responses are received from both group and overlay storage is a decision made whether the request was a success or failure. If the majority of responses are successful, the response is determined as successful. Safe storage is much less responsive compared to fast storage, but in the presence of malicious peers, safe storage is more secure. Figure 9 illustrates Nomad's storage procedure:

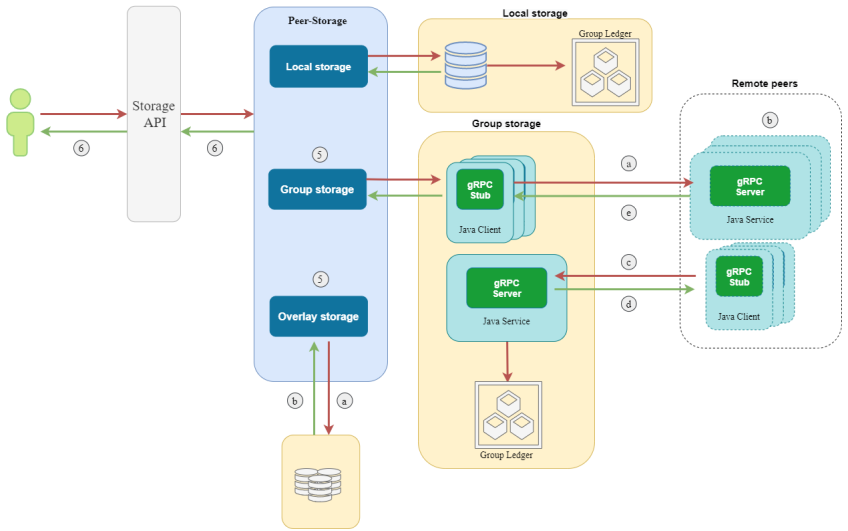

Fig. 9. Nomad object storage procedure

(1) The higher layer VE logic uses Nomad's peer storage API to execute an *add* operation.
(2) The object is stored to local storage. The peer adds the object to its authoritative object store.
  (a) The storage operation is relayed to the peer's local storage module (H2/RocksDB).
  (b) Only if the operation succeeds, does the peer add the object ID, coupled with a TTL, to its group ledger.
  (c) The local storage module responds with success.
(3) Group storage is used to notify peers of the new ledger entry.
(4) Nomad propagates the storage request to multiple peers within the group for replication purposes. Requests are made in parallel to improve responsiveness.
  (a) The request is propagated to a subset of the group peers, depending on the replication factor ($\mathcal{R}$).

(b) Each peer receiving the request ensures that it does not already store the object, by checking its group ledger. If the object is not stored on the peer, the peer will initiate its own storage request.
(c) Once the object is stored successfully, an *addObjectReference* request is made to all group peers.
(d) Peers add the object reference to their local group ledger and respond with an *ACK* message.
(e) The peer informs the original requesting peer that the object replica has been stored successfully.

(5) Nomad simultaneously propagates the storage request to its overlay storage component.
(a) The storage request is made to the overlay storage component.
(b) The overlay storage component completes the storage request and responds with success or failure.

(6) Nomad responds to the higher layer:
(a) **Fast storage:** Nomad responds with the successful result from either overlay or group storage.
(b) **Safe storage:** Nomad monitors responses from multiple peers in group storage and overlay storage. If the majority of responses are successful $[(\mathcal{R}/2) + 1]$, the response is deemed successful.

5.11.2 Retrieve. Nomad has three retrieval modes, namely *fast*, *safe* and *parallel* retrieval. Fast retrieval mode is similar to fast storage mode. In fast retrieval mode, Pithos sends a retrieval command to a single group storage peer and to overlay storage. In order to select a group peer to send the request to, the originating peer queries its group ledger and filters out all peers not storing the requested object. From the filtered peers, a random peer is selected and is sent the retrieval request. The first successful response object that it receives, either from group or overlay storage, is then propagated to the higher layer. Fast retrieval is generally very responsive and requires minimum bandwidth, but it is more susceptible to malicious peers and object manipulation. In safe retrieval mode, Nomad's peer storage component sends a retrieval command to multiple group storage peers and to overlay storage. After a sufficient number of response objects are received, a quorum mechanism is used to determine the original object. If the majority of responses are successful, the response is deemed successful. Safe storage is resilient against malicious peers, however similarly to safe storage, it is less responsive compared to fast and parallel retrieval. In parallel retrieval mode, Nomad sends a retrieval command to multiple group storage peers, and to overlay storage. Similar to fast storage, parallel retrieval uses the first successful response to serve the higher layer. Parallel retrieval is generally more responsive and reliable than fast storage, but is more susceptible to malicious peers than safe retrieval. Figure 10 illustrates Nomad's retrieval procedure.

(1) A retrieval request is sent from the higher layer to the Nomad storage interface.
(2) Nomad first determines whether the object is hosted within the group by checking its group ledger.
(3) If the object is stored locally, it is retrieved from local storage and used to determine the appropriate response.
(4) If the object is stored within the group, Nomad propagates the retrieval request to both group storage and overlay storage peers.
(a) Nomad only sends a retrieval request to peers that, according to its group ledger, are storing the required object.

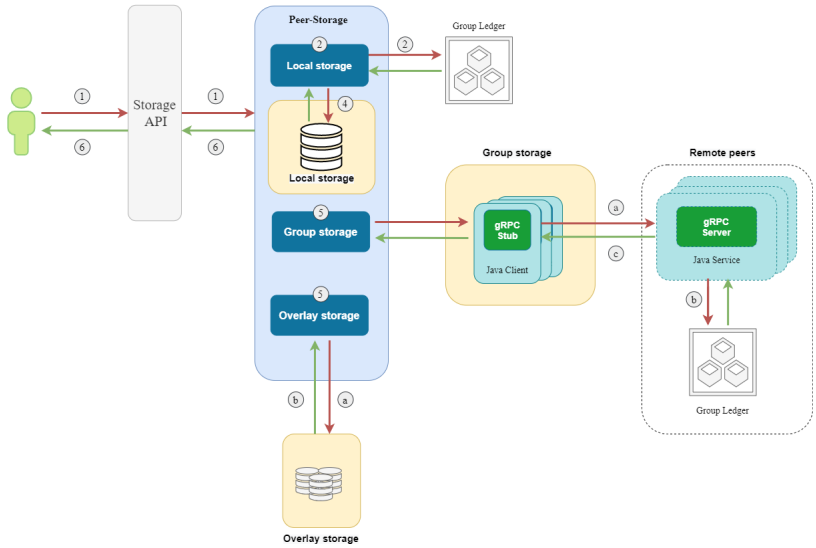

Fig. 10. Nomad object retrieval procedure

(b) Peers that receive the retrieval request, verify that they are storing the requested object, by using their own group ledger. If the peer does store the object, it is retrieved from the peer's local storage.
(c) The retrieved object is returned to the originating peer, along with the appropriate response message.

(5) Nomad simultaneously propagates the storage request to its overlay storage component. If the object is not stored within the group, a retrieval request is only sent to overlay storage. This is known as an out-of-group request.
(a) The retrieval request is made to the overlay storage component.
(b) The overlay storage component completes the retrieval request and responds with the resulting object.

(6) Nomad responds to the higher layer:
(a) **Fast retrieval:** Nomad responds with the resulting object from either local, overlay or group storage, whichever successful response it receives first.
(b) **Safe retrieval:** Nomad monitors responses from local, group storage and overlay storage. A quorum mechanism $[(\mathcal{R}/2) + 1]$ determines if the request was successful and which object should be sent to the higher layer.
(c) **Parallel retrieval:** Nomad responds with the resulting object from either local, overlay or group storage, whichever succeeds first.

5.11.3 Modify. Within Nomad, modify requests are treated similarly to store requests in safe mode. Similar to all other storage operations, modify requests are received from the higher layer. The peer receiving the modify request from the higher layer verifies that the object is stored in the group. If the object is not stored within the group, the object is only modified in overlay storage. If the object is stored within the group, the object is modified within group storage and overlay storage. Only peers storing the object will receive an object update request. Only if all object modifications succeed is the modify request deemed successful.

5.11.4 Delete. Similar to the Pithos architecture, Nomad's storage API does not explicitly support object removal. Instead it relies on an object's TTL. When an object is created in Nomad, it is assigned a TTL that dictates the object's "expiry" time. When an object's TTL period expires, the object is removed from group storage, overlay storage and the group ledger. In order to ensure objects are removed from a peers local storage and group ledger after their TTLs expire, a scheduled maintenance task periodically checks the local storage for expired TTLs. Objects that have expired TTLs are removed from local storage and the group ledger.

6 NOMAD EVALUATION

Nomad's evaluation served a dual purpose, firstly to verify the simulated Pithos results and secondly to verify that Nomad meets the storage requirements of: (1) Responsiveness, (2) reliability, (3) scalability, (4) security and (5) fairness/load-balancing.

6.1 Experimental Setup

6.1.1 Eperiment parameters overview. Table 1 contains the various configuration settings (test variables) used during the system evaluation test iterations.

Param.	Nomad Configuration	Pithos Configuration
Group Size	25 (24 peers, 1 super-peer)	25 (24 peers, 1 super-peer)
Network Size	50 (40 peers, 10 super-peer)	2,600 (2500 peers, 100 super-peer)
Simulation Length	3600 s	10,000 s
No. Requests	350,000 (avg.)	5,000,000 (avg.)
No. Generated Objects	330,000 (avg.)	600,000 (avg.)
Object Size	1024 bytes	1024 bytes
Overlay	TomP2P (Kademlia)	Chord (medium)
$\mathcal{R}$	6 replicas	6 replicas
Network Churn	On	On

Table 1. Summary of group storage evaluation variables for Nomad vs Pithos

6.1.2 The Test Virtual Environment. For the purpose of this experiment, the geography of the map is not important. The application merely requires x and y limits. The map limits ensure that all player movements are restricted to the virtual plane. During the experiment, x and y limits of 10.0 were used for the virtual world. Figure 11 illustrates the virtual world, segmented into Nomad groups, using a Voronoi grouping mechanism. The image was generated by super-imposing the Voronoi map generated during one of the test iteration onto the fortnite map [22]. The visualisation of the Voronoi map was generated using an online tool [1]. Black dots represent super-peer positions, which act as Voronoi site points.

6.1.3 Player Movement Model. In order to simulate realistic player movement within the test virtual world, a player movement model was required. We made use of recent work by Moll et al. [14], who present extrapolated player movement patterns and network traffic, based on analysis of a popular battle royal game, *Fortnite* [20]. The FortniteTraces [13] dataset was used as the player movement model for each Nomad storage peer. Voronoi-grouping split the virtual world into geographical segments. This facilitated group migration of a peers, enabling their movement across multiple Voronoi segments within the virtual world. Movement traces were contained within the x and y map limits mentioned earlier. Each Nomad instance was started with a movement file provided by the FortniteTraces dataset. The movement files provided every peer with positional

updates , i.e. x, y coordinates, every 10 seconds, which represented the peer's movement within the virtual world.

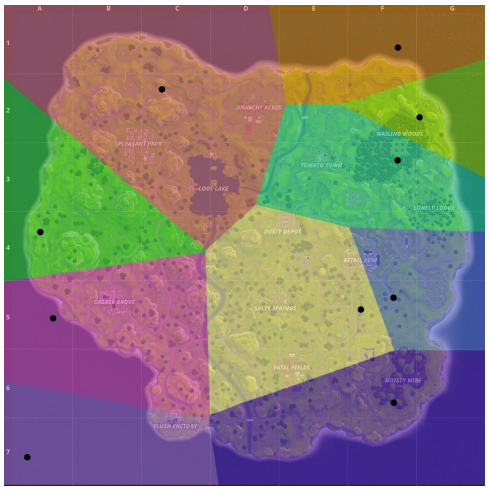

Fig. 11. Example of Voronoi divided map and super-peer positions

6.1.4 Network Churn. At the beginning of each test iteration, 50 Nomad peers were created, then at 1 to 2 minute intervals, an instance was either added or removed, representing a game session ending or starting. For peer removal, random instances were selected and shut down, either gracefully or forcefully. Both peers and super-peers were affected by network churn, in order to test both peer and super-peer leave scenarios. The network size was kept above 45 peers.

6.1.5 Iteration Length. Two test iterations were performed. Each ran for 3600 seconds, generating on average 350,000 storage & retrieval requests. During storage tests, an average of 330,000 objects were generated.

6.1.6 Request Rate. No higher layer application implementing Nomad exists as of yet. A standalone higher layer application was therefore required to generate storage requests on Nomad peers. To satisfy this requirement, Artillery [17] was used to generate HTTP requests to each peer's storage API. A load test of 100 RPS distributed over the entire network was executed in each iteration, using a 60 seconds warm-up phase, increased incrementally from 10 to 100 RPS. The request rate was kept stable at 100 RPS for the rest of the experiment.

6.2 Responsiveness and Reliability

Table 2 shows a comparison between Nomad and Pithos in terms of overall responsiveness and reliability per component. Overall, Nomad was found to be slightly more reliable and responsive when retrieving objects from group storage compared to Pithos. When using parallel retrieval, Nomad provides response times of under 20 ms and 100% reliability, which is a better than Pithos' 60 ms response times and 99.98% reliability. The same observation was made for fast retrieval.

In terms of object storage, Nomad was found to be more responsive, but not as reliable as Pithos. Using safe storage, Nomad provides response times of 1.2 seconds and reliability of 83.30%, whereas Pithos provides slightly slower response times of 1.554 seconds, but better reliability at 97.05%. This is due to Nomad's inefficient overlay storage component.

System	Entity	Reliability (%)		Responsiveness (s)		Bandwidth (Bps)
		store	retrieve	store	retrieve	
Pithos (safe store)	Chord (med)	96.90	93.20	1.214	1.582	2,380
	Kademlia	45.53	35.13	0.908	4.604	1,010
	Group Storage (fast retrieve)	97.05	99.77	1.554	0.189	2,750 (363)
	Group Storage (parallel retrieve)	97.05	99.98	1.554	0.0859	3,972 (1,592)
Pithos (fast store)	Group Storage (fast retrieve)	100.00	99.70	0.0665	0.192	2,743 (370)
	Group Storage (parallel retrieve)	100.00	99.98	0.0665	0.0846	3,899 (1,519)
Nomad (safe store)	Kademlia	100.00	47.40	1.630	2.949	8,500
	Group Storage (fast retrieve)	83.30	100.00	1.231	0.017	21,450 (12,950)
	Group Storage (parallel retrieve)	83.30	100.00	1.231	0.0192	22,150 (13,650)
	Group Storage (safe retrieve)	83.30	100.00	1.231	0.254	22,900 (14,400)
Nomad (fast store)	Group Storage (fast retrieve)	100.00	100.00	0.0271	0.017*	16,550 (8,050)
	Group Storage (parallel retrieve)	100.00	100.00	0.0271	0.0192*	17,250 (8,750)
	Group Storage (safe retrieve)	100.00	100.00	0.0271	0.254*	18,00 (9,500)

Table 2. Pithos versus Nomad performance for different overlay implementations and storage and retrieval modes [4] (Retrieval operations marked with * were tested using safe storage)

When comparing the two systems' overlay components, Pithos' clearly has a better overlay implementation. Pithos' overlay was found to be more reliable and more responsive under network churn. Chord provides much better retrieval reliability of 93.20%, compared to TomP2P's 47.40%, whilst simultaneously providing better responsiveness.

Nomad's bandwidth requirement is much higher than that of Pithos. Overall bandwidth usage is calculated as the sum of retrieval, storage and overlay bandwidth usage. On average, Nomad's bandwidth usage is almost 7 times that used by Pithos. This is caused by Nomad's higher bandwidth usage for both group and overlay storage. Additionally, Nomad's overlay storage was found to be extremely bandwidth inefficient.

6.3 Security

Table 3 provides a comparison between Nomad and Pithos in terms of security against malicious peers. It shows that Nomad's reliability follows a similar trend to that of Pithos, with safe storage being the most reliable of the retrieval methods. Nomad's safe storage implementation does, however, decay faster in terms of security than that of Pithos. Nomad's lower security is due to the fact that it does not implement a certification mechanism, which Pithos does. The development of a certification mechanism for Nomad is deemed future work.

Malicious Peers (%)	Retrieval Mode	Pithos Security (%)	Nomad Security (%)
10	Fast	90.00	90.00
	Safe	99.00	99.00
	Parallel	-	95.00
25	Fast	78.00	77.00
	Safe	95.00	81.00
	Parallel	-	79.00
50	Fast	55.00	45.00
	Safe	65.00	52.00
	Parallel	-	50.00
75	Fast	28.00	21.00
	Safe	28.00	26.00
	Parallel	-	23.00

Table 3. Overview of Nomad vs Pithos security against malicious peers for different retrieval modes (Safe Retrieval use 4 compares for Quorum)

7 CONCLUSION

7.1 Responsiveness

Nomad achieves responsiveness by splitting the network into fully connected groups of peers. The underlying architecture uses a distance-based group-based storage mechanism to store, retrieve, distribute and replicate objects. The assumption is made that peers are more likely to request objects from their own group, which helps Nomad to achieve highly responsive storage and retrieval for in-group requests.

Nomad's evaluation confirmed that group storage is extremely responsive in performing storage and retrieval operations. In terms of object storage, fast mode was found to be the most responsive and to require the lowest amount of bandwidth compared to safe storage. In terms of object retrieval, parallel retrieval was found to be more responsive than fast and safe retrieval, however required more bandwidth than fast retrieval.

Creating object replicas proved to be costly in terms of resource usage, although it does improve the responsiveness of storage and retrieval requests as some peers may be geographically closer than others, leading to lower RTLs. In parallel retrieval and fast storage mode, Nomad sends identical requests to multiple peers and only awaits the first successful response, thereby taking advantage of the distribute nature of object replicas.

In terms of responsiveness, Nomad's evaluation produced similar values to those measured during Pithos' simulation. It can therefore be concluded that Pithos' measured response times are accurate and that Pithos satisfies the P2P MMVE storage requirement of responsiveness.

7.2 Reliability

Reliability is achieved in Nomad by making use of various modules and mechanisms. An overlay storage component in the form of a DHT increases reliability, as it is able to serve any in-group or out-of-group retrieval request. Object replication ensures that sufficient replicas exist in case of storage peers leaving the network. Replicas are maintained by object repair mechanisms, which ensures a set number of object replicas are always available.

Nomad's evaluation confirmed that the overlay storage impacts the overall performance of the network. During Nomad's evaluation, a Kademlia based DHT was used as overlay component,

whereas during the Pithos simulation a Chord based DHT was used. Kademlia proved to be far less reliable and responsive compared to Chord, and led to a lower overall reliability.

Nomad's overlay component was far less responsive under network churn compared to group storage. Object storage requests were reliable, but retrieval requests far less so, and also less responsive. Nomad can be described as less reliable than the Pithos simulation, as out-of-group requests are likely to fail in Nomad. As such, Nomad's overlay storage is one of the system's weaknesses and requires further investigation.

The use of object replicas improved overall reliability, as replication provides high availability for objects in the presence of group migration and network churn. Nomad's scheduled and leave-repair mechanisms proved to consistently maintain object replicas, but had a significant impact on performance.

Nomad's evaluation provided similar results to those of the Pithos simulation. It can therefore be concluded that Pithos satisfies the P2P MMVE storage requirement of reliability.

7.3 Security

Security is achieved in Nomad by making use of object replication and a quorum mechanism. Nomad also supports *safe* reads and writes, which inherently increases reliability and security.

Object replication diminishes the impact of malicious peers in an MMVE. As multiple peers store object replicas, a maliciously altered version of the object is easily identifiable. Multiple object replicas also enable the use of quorum mechanisms on parallel responses.

When parallel requests are made, depending on the retrieval mode, a quorum mechanism can be used to verify object state. This way, a peer can decide to select only the most frequently occurring version of an object. This effectively diminishes the impact of malicious peers in the system and improves security. Nomad's quorum mechanism can be described by the quorum formula $(\mathcal{R}/2)+1$, where $\mathcal{R}$ is the replication factor.

Pithos additionally implements a certificate authority (CA) and a certification mechanism, in order to assign IDs and certificates to peers, and to sign object changes. Nomad does not make use of this certification mechanism and therefore, in its current state does not support Secure Sockets Layer (SSL) communication between peers or signing of object changes. These certification mechanisms are considered future.

Nomad's implementation lacks the security features of the Pithos architecture, but does provide core security functionality for verifying object integrity. Nomad's evaluation proved that Pithos' results are accurate and that Pithos satisfies the P2P MMVE storage requirement of security.

7.4 Fairness

To ensure fairness (load-balancing), peers and super-peers are not selected on any discriminating factors but purely based on their location and time of joining the group. Load is distributed evenly across all peers in the network by making use of randomly distributing group storage requests. Group storage and overlay storage are inherently considered to be fair.

The group storage mechanism distributes object replicas in a uniformly random fashion, ensuring that no one peer is favoured for storage purposes. Peers use their own copy of the peer ledger to choose random peers in the group when (a) requesting objects or (b) storing object replicas.

Similarly, the overlay storage maps objects and peers to the same identifier space. Objects are stored on peers with the closest ID match. This way, if IDs are assigned in a uniformly random fashion, storage load is inherently distributed in a uniformly random fashion.

During Nomad's evaluation, fairness was not directly measured, but observation of resource usage confirmed that all peers in a group had similar memory footprints. From Nomad's evaluation, it can be confirmed that objects were evenly distributed amongst peers. This means that Pithos'

measured object distribution results can be considered accurate and that Pithos satisfies the P2P MMVE storage requirement of fairness.

7.5 Scalability

In Nomad scalability, is achieved by making use of modules that are scalable in themselves. Nomad uses either a size-based or Voronoi-based grouping mechanism, which splits the network into fully connected groups. Groups have a fixed size limit, as fully connected networks scale quadratically in terms of messages required per transaction. Group size is therefore the limiting scaling factor. For its overlay storage component, Nomad makes use of a DHT, which is by design scalable.

One possible system bottleneck is the directory server, since it facilitates network bootstrapping, and provides peers with a local group cache which is used for various purposes. For Nomad's directory server, a scalable and highly consistent distributed coordination system called ZooKeeper is used. Nomad's evaluation proved that as the number of nodes in the network increased, performance was not severely affected.

8 RECOMMENDATIONS FOR FUTURE WORK

(1) *Scalability Evaluation* - Since Nomad could only be evaluated with up to 50 peers, it is recommended that a scalability evaluation be executed using a higher number of nodes.
(2) *Implement a fully functional certification mechanism* - Nomad currently does not make use of any certification mechanism, which means that peer validation and tracking of object changes are not possible. A certification mechanism should be implemented in order to allow for additional security in Nomad.
(3) *Replace Nomad's overlay storage component* - Nomad currently makes use of TomP2P, a Kademlia based DHT, as overlay storage. During Nomad's evaluation, the overlay storage component was found to be extremely unresponsive under network churn, and generally unreliable. It is therefore recommended to either replace Nomad's overlay component with a more suitable DHT implementation, or to implement a Chord based DHT from scratch.
(4) *Research alternatives for directory server implementation* - Nomad currently makes use of ZooKeeper, for its directory server. In its current state, the directory server is completely functional and capable of providing the required logic. The longevity of ZooKeeper is, however, somewhat of a concern. Projects like *etcd* [19] and *consul* [7] are very promising alternatives that are widely used in the industry and have the prospect of longevity.
(5) *Improve safe storage quorum mechanism* - Nomad's current quorum mechanism relies heavily on Java POJO (Plain Old Java Object) comparison, which is robust, but prone to human error. It is recommended to implement a more secure and robust quorum mechanism for Nomad, in order to ensure the correct function of safe retrieval and updates.
(6) *Improve fast storage response times* - In order to make fast storage completely independent of group size, additional software improvements are required. In Nomad's current state, a peer will notify all other group peers of an object added before reporting success. This can be improved by returning storage results *before* notifying peers of new objects. This means that once the object is stored locally, a successful storage result is returned.
(7) *Allow super-peer movements* - To simplify Nomad's Voronoi grouping logic, once a super-peer is selected and it sends its VE location to the directory server, it can no longer receive any positional updates. This means that super-peer positions are static in Nomad's current implementation. Additional logic is required to allow super-peers to move within the VE. This means that groups will have a velocity and that all peers and super-peers need to recalculate their Voronoi maps regularly to ensure a consistent view of the VE. Calculating the Voronoi

map too frequently causes excessive system load, especially for large VEs. It is therefore recommended to recalculate the Voronoi map at a reasonable interval.

(8) *Solving NAT challenges* - Docker containers and Kubernetes deploy scripts already exist for Nomad, but in order for Nomad to be used in the cloud, it is required to first solve the problem of NAT traversal. NAT traversal is a problem, due to Nomad's use of random ports for its services, since multiple Nomad instances are executed on a single machine. If static ports are used, the problem will become significantly easier to solve.

REFERENCES

[1] Alex Beutel. [n.d.]. *"Interactive Voronoi Diagram Generator with WebGL"*. http://alexbeutel.com/webgl/Voronoi.html vistied on 2021-04-12.

[2] Thomas Bocek. [n.d.]. *"TomP2P"*. https://github.com/tomp2p/TomP2P vistied on 2021-07-22.

[3] Mark Claypool and Kajal Claypool. 2006. Latency and player actions in online games. *Commun. ACM* 49 (11 2006), 40–45. https://doi.org/10.1145/1167860

[4] Herman A. Engelbrecht and John S. Gilmore. 2017. Pithos: Distributed Storage for Massive Multi-user Virtual Environments. *ACM Trans. Multimedia Comput. Commun.* 13, 3 (2017), 31:1–31:33.

[5] John Gilmore and Herman Engelbrecht. 2012. A Survey of State Persistence in Peer-to-Peer Massively Multiplayer Online Games. *Parallel and Distributed Systems, IEEE Transactions on* 23 (2012), 819–825. https://doi.org/10.1109/TPDS.2011.210

[6] John S. Gilmore and Herman A. Engelbrecht. 2013. *A State Management and Persistence Architecture for Peer-to-Peer Massively Multi-user Virtual Environments.* Ph.D. Dissertation. Stellenbosch University, South Africa.

[7] HashiCorp. [n.d.]. *"Service Discovery and Health Checking"*. https://www.consul.io/use-cases/service-discovery-and-health-checking vistied on 2021-09-03.

[8] Shun-Yun Hu, Shao-Chen Chang, and Jehn-Ruey Jiang. 2008. Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games. *5th IEEE Consumer Communications and Networking Conference*, 1134–1138.

[9] IGN.com. 2012. *"World of Warcraft Reaches 12 Million Subscribers"*. https://www.ign.com/articles/2010/10/07/world-of-warcraft-reaches-12-million-subscribers visited on 2021-06-29.

[10] Eberspächer Jörg, Rüdiger Schollmeier, Stefan Zöls, and Gerald Kunzmann. 2021. Structured P2P Networks in Mobile and Fixed Environments. (2021), 3.

[11] Petar Maymounkov and David Eres. 2002. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Lecture Notes in Computer Science, vol. 2429*, Vol. 2429. https://doi.org/10.1007/3-540-45748-8_5

[12] MMO-population.com. [n.d.]. *"NEW WORLD Game population, subscriber counts, daily players and trends for New World."*. https://mmo-population.com/r/newworldgame visited on 2021-10-20.

[13] Philipp Moll, Mathias Lux, Sebastian Theuermann, and Hermann Hellwagner. [n.d.]. *"FortniteTraces"*. https://github.com/phylib/FortniteTraces vistied on 2021-04-12.

[14] Philipp Moll, Mathias Lux, Sebastian Theuermann, and Hermann Hellwagner. 2018. A Network Traffic and Player Movement Model to Improve Networking for Competitive Online Games. *16th Annual Workshop on Network and Systems Support for Games (NetGames)*, 1–6. https://doi.org/10.1109/NetGames.2018.8463390

[15] Aline Normoyle, Gina Guerrero, and Sophie Jörg. 2014. Player Perception of Delays and Jitter in Character Responsiveness. *Proceedings of the ACM Symposium on Applied Perception, SAP 2014* (2014), 1–10. https://doi.org/10.1145/2628257.2628263

[16] Amazon Games. [n.d.]. *"New World"*. https://www.newworld.com/en-gb visited on 2021-10-20.

[17] Artillery.io. [n.d.]. *"Why Artillery?"*. https://artillery.io/docs/guides/overview/why-artillery.html vistied on 2021-08-04.

[18] Blizzart Entertainment Inc. [n.d.]. *"World of Warcraft"*. https://worldofwarcraft.com/en-gb/ visited on 2021-08-19.

[19] Cloud Native. [n.d.]. *"What is etcd?"*. https://etcd.io/ vistied on 2021-09-03.

[20] Epic Games. [n.d.]. *"Fortnite"*. https://www.epicgames.com/fortnite/en-US/home vistied on 2021-08-07.

[21] Facebook. [n.d.]. *"RocksDB: A Persistent Key-Value Store for Flash and RAM Storage"*. https://github.com/facebook/rocksdb vistied on 2021-07-04.

[22] Fortnite Wiki. [n.d.]. *"Map Gallery"*. https://fortnite.fandom.com/wiki/Map_Gallery vistied on 2021-04-12.

[23] gRPC Authors. [n.d.]. *"Introduction to gRPC"*. https://grpc.io/ vistied on 2021-07-22.

[24] H2. [n.d.]. *"History and Roadmap"*. https://www.h2database.com/html/history.html vistied on 2021-07-24.

[25] The Apache Software Foundation. [n.d.]. *"ZooKeeper"*. https://zookeeper.apache.org/doc/r3.7.0/zookeeperOver.html vistied on 2021-07-22.

[26] Amir Yahyavi and Bettina Kemme. 2013. Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey. *ACM Computing Surveys, vol. 46, no. 1* (2013), 39:9–39:10. https://doi.org/10.1145/0000000.0000000

9 NOMAD REPOSITORY

The Nomad implementation repository is openly available on gitlab.com. The repository provides a useful wiki, demo videos and a README with instructions on how to run the Nomad application locally. View the repository at https://gitlab.com/iggydv12/nomad.