The Design of a Contemporary Infrastructure for Scalable and Consistent Virtual Worlds

Umar Farooq

School of Computing Sciences
University of East Anglia

February 2012

To my parents, brothers and sisters.

Abstract

Virtual worlds have recently emerged as interactive and collaborative virtual spaces that have unique features. Existing mechanisms originally developed for games or simulation environments are currently used to handle scalability, load distribution, and consistency issues for virtual worlds, but they have key performance issues. This dissertation examines a novel infrastructure based on inherent properties of virtual worlds targeting these issues. It uses a constrained hierarchical approach for managing the resources of a system and a constrained P2P communication model that consults only the adjacent neighbouring nodes to maintain the temporal order among events. It presents simulation work for scalability, load distribution, and consistency as well as an extension to the current architecture of OpenSimulator to incorporate scalability and consistency. It also describes a prototype developed to implement our scalability and load distribution methods.

The Joint Hierarchical Nodes Based User Management infrastructure was developed to scale virtual worlds and applies both splitting and merging to adapt to system load. It deals with issues in both static and dynamic infrastructures. It aims to simplify the management of a hierarchical virtual world while maintaining user experience. To minimise resource utilisation and communication overhead, we developed an aggregation algorithm using a number of aggregation strategies. This maintains regular and contiguous spaces for assignment and distributes the load so it is as balanced as possible between two simulators. We also present a fully decentralised synchronisation method to maintain a consistent view of a virtual world represented as a complex hierarchical model. It reduces communication overhead and maintains local causality.

We investigate the capabilities of OpenSimulator and develop a load model that determines when to initiate a split or a merge operation. We presented an abstract framework for scalability which is implemented by building on the basic capabilities of OpenSimulator. This work is demonstrated through experiments on both Windows and Linux platforms. It obtains the same level of scalability as static configurations but with a reduced number of resources. It further improves over current dynamic approaches by transferring regions in an aggregate in turn. For evaluation purposes, we used a number of timing and system statistics. We developed significantly more efficient algorithms for removing a region from a simulator compared with the basic methods of OpenSimulator. Overall, we effectively developed a system that dynamically expands and contracts the set of servers used to support a virtual world based on load estimated by tracking the number of active players.

Acknowledgments

IN THE NAME OF ALLAH, THE MOST MERCIFUL AND BENEFICIAL! Many thanks goes to almighty Allah, who gave me the strength and ability to finish this work.

I would like to take this opportunity to say special thanks to my supervisors Prof. John Glauert and Dr. Rudy Lapeer for their continuous guidance, support and advice.

Dear John! Thanks for your hospitality, help and guidance. I am much more confident in accepting challenges and providing viable solutions to industry problems due to the training you provided during the past few years. Thanks for listening to me whenever I was in trouble. I can never forget your support and the opportunities you provided me for training and presenting my work.

Special thanks to my parents, brothers and sisters for all the support they have given over the course of the last few years.

This acknowledgment is incomplete without mentioning all my friends around the world who shared some of the best memories of my life while studying at UEA. I can never forget you all. Special thanks to Dr. Rafat Nabi, Mr. Muhammad Atif, Dr. Jason Yu Liu, Dr. Osama Dorgham, Dr. Fabiola Lopez Gomez, Mr. Wayne Henry, Mr. Adil Zaman, Miss. Hang Nguyen, Miss. Paulina Glowacka, Miss. Saima Laiq, Miss. Kim Findlay Cooper, Dr. Khalid Shah, Dr. Sadaqat Ali, Engr. Faheem Ullah Khan and Mr. Khaled Alotaibi for their support and company, and, to

Catherine Baker for proof reading this thesis and helping in preparing the final manuscript.

Finally, I would like to say thank you to the University of Science and Technology Bannu for funding my PhD studies and, to the OpenSim community for their help and support. **The research presented in this thesis was tested on the High Performance Computing Cluster supported by the Research Computing Service at the University of East Anglia.** Special thanks go to Mr. Chris Collins and Dr. Miah Wadud from the E-Science project for their help and providing special arrangements for this different but exciting project.

Contents

List of Figures

List of Tables

Publications

The following are publications related to this work by the author:

- Umar Farooq and John Glauert. Scalable Virtual Worlds: An Extension to the OpenSimulator Architecture. In *1st IEEE International Conference on Computer Networks and Information Technology (ICCNIT'2011)*, pages 29–34, Abbotabad, Pakistan, July, 2011 [67]. [Online]: Available at: `http://dx.doi.org/10.1109/ICCNIT.2011.6020902`

- Umar Farooq and John Glauert. A Hybrid Infrastructure for Scalable and Consistent Virtual Worlds. In *Winter Simulation Conference (WSC'10)*, Baltimore, USA, 2010 [65]. [Online]: Available at `http://www.informs-sim.org/wsc10papers/prog10.html`

- Umar Farooq and John Glauert. Time Management for Virtual Worlds based on Constrained Communication Model. In *Proceedings of the 9th ACM/IEEE International Workshop on Network Support for Games (NetGames'10)*, pages 19:1–6, Taipei, Taiwan, 2010 [66]. [Online]: Available at `http://dx.doi.org/10.1109/NETGAMES.2010.5679667`

- Umar Farooq and John Glauert. A Decentralised Synchronisation Approach for Complex Hierarchical Models of Virtual Worlds. In *Proceedings of the IASTED Parallel and Distributed Computing and Systems (PDCS'10)*, pages 218–224, California, USA, 2010 [63]. [Online]: Available at `http://dx.doi.org/10.2316/P.2010.724-024`

- Umar Farooq and John Glauert. A Dynamic Load Distribution Algorithm for Virtual Worlds. In *Journal of Digital Information Management*, Volume

8, Number 3, pages 181–189, June, 2010 [64]. [Online]: Available at `http://www.dirf.org/jdim/v8i3.asp`

- Umar Farooq and John Glauert. Managing Scalability and Load Distribution for Large Scale Virtual Worlds. In *Proceedings of the UEA School of Computing Sciences Symposium*, pages 20–27, Norwich, UK, Obtober, 2009 [62]. [Online]: Available at `https://www.uea.ac.uk/polopoly_fs/1.133529!Farooq.pdf`

- Umar Farooq and John Glauert. ARA: An Aggregate Region Assignment Algorithm for Resource Minimisation and Load Distribution in Virtual Worlds. In *NDT '09: Proceedings of the first IEEE International Conference on Networked Digital Technologies*, pages 404–410, Ostrava, Czech Republic, July, 2009 [60]. [Online]: Available at `http://dx.doi.org/10.1109/NDT.2009.5272118`

- Umar Farooq and John Glauert. Joint Hierarchical Nodes based User Management (JoHNUM) Infrastructure for the Development of Scalable and Consistent Virtual Worlds. In *DS-RT '09: Proceedings of the 13th IEEE/-ACM Symposium on Distributed Simulation and Real-Time Applications*, pages 105–112, Singapore, October, 2009, IEEE Computer Society [61]. [Online]: Available at `http://dx.doi.org/10.1109/DS-RT.2009.32`

Extension to our work

Our work on designing scalable virtual worlds is explored and compared with static and dynamic strategies by a group of master students at the University of Aalborg Denmark. They extended it for 3D environments and used these concepts to develop a test game called Rock Pounder. Based on a number of different scenarios, and tests, they suggested that JoHNUM infrastructure performs better than both static and dynamic strategies. The complete report (Master Dissertation) under a title "To Infinity and Beyond: Scaling Massively Multiplayer Games" and the relevant material (source code and videos) are available at `http://www.ejlersen.info/index.php?page=studyprojects&id=11` for the reference.

Acronyms

AD	Agent Domain
ARA	Aggregate Region Assignment
BG	Butterfly Grid
CM	Child Merge
CMS	Client Multi Server
CoS	Cluster of Servers
CR	Create Region
CR Time	Create Region Time
CS	Client Server
CSCW	Computer Supported Co-operative Work
CsR	Close Region
CVE	Collaborative VE
DIS	Distributed Interactive Simulation
DR	Delete Region
DSG	Distributed Scene Graph
DVE	Distributed VE
FPS	Frames Per Second
GALT	Greatest Available Logical Time
HLA	High Level Architecture
JoHNUM	Joint Hierarchical Nodes Based User Management
LBTS	Lower Bound on Time Stamp
LC	Load Content

LC Time	Load Content Time
LOAR	Load OAR
LP	Logical Process
MMOG	Massively Multiplayer Online Game
MUVE	Multi User VE
NVE	Networked VE
OAR	OpenSim Archive
OS	OpenSimulator
P2P	Peer-to-Peer
PADS	Parallel and Distributed Simulation
PandASs	Prims and Active Scripts
PDES	Parallel Discrete Event Simulation
PM	Parent Merge
RAd	RemoteAdmin
RCT Time	Region Content Transfer Time
RD	Region Domain
RMT	Resource Management Tree
RR	Remove Region
RR Time	Remove Region Time
RT Time	Region Transfer Time
RTI	Run Time Infrastructure
RUnAv Time	Region Un-availability Time
SC	Store Content
SC Time	Store Content Time
Sim	Simulator
SL	Second Life
SLG	SL Grid
SOAR	Save OAR
STDEV	Standard Deviation

T2R Time	Transfer to Region Time
T2T Time	Transfer to Transit Time
TClient	TestClient
TM	Time Management
VE	Virtual Environment
VR	Virtual Reality
VW	Virtual World
WoW	World of Warcraft
WWW	World Wide Web

Chapter 1

Introduction

This chapter presents a brief introduction to the concept of Virtual Environments (VEs) and highlights their conflicting terminologies used in the Literature. It introduces scalability, load distribution, and synchronisation issues in recently emerged interactive and collaborative social spaces called Virtual Worlds (VWs). It outlines applications of VEs and presents the research methodology used for this research work. It also lists our contribution and gives the organisation of this thesis.

1.1 Introduction to VEs

Virtual Reality (VR) is a interesting area of research that changes rapidly. It provides the foundation for defining a number of VEs with different concerns and goals. The Literature has, therefore, a number of conflicting terminologies to define VEs. VR has no common definition either, but the majority of these environments share common characteristics with specialised parameters that differentiate them from each other [12, 205]. According to Baladi et al. [12], these parameters include application, technology, number of users and the underlying infrastructure. VR is all about manipulating a 3D computer-generated space based on real or imaginary content and navigating through it. Different navigation styles such as walking, running, or flying are used to even explore different situations (viewpoints) that could be dangerous and expensive in the physical world as a real operational system [222]. A VR can also be used to observe parts of a system which are not observable [84]. According to Vince [222], “the real benefit of VR is the ability to touch, animate, pickup and reposition virtual objects and create totally new configurations”. VR systems try to mimic the real world and provide immersion in a 3D space [205]. Fujimoto defines a VE as a simulation that often executes a set of dynamic computer-generated entities and requires real time execution so that it appears realistic and evolves as rapidly as an actual system [84]. In these systems, time advance in simulation time is paced to occur in synchrony with an equivalent advance in wallclock time [82].

With the passage of time, VEs gained sophistication with the introduction of new concepts and I/O methods to represent and access a shared digital space [14, 35]. The concept of avatars re-defined the landscape of online interaction away from text towards a more complex visual presentation of interactive and collaborative tasks. However, this category mostly represents entertainment environments such as games that follow pre-defined set of rules for different activities. Techniques such as sharding are used to make these environments scalable. A shard is a copy of a part of a virtual space [140]. Most recently, people have shown a keen interest in interactive and collaborative social environments called Virtual Worlds (VWs). These worlds have a number of unique features and allow users to design their content according to their desires. Therefore, it is much more difficult to manage these worlds compared with game environments [165]. This thesis targets VWs and explores techniques to make them scalable and consistent while achieving a fair distribution of load.

1.2 Conflicting terminologies defining VEs

The Literature provides a large number of confusing terminologies to define VEs including VR, VR System, VEs, Distributed VE (DVE), Networked VE (NVE), Collaborative VE (CVE), Multi User VE (MUVE), and VW being the most recent one [12]. According to our observations, a number of terminologies exist as this area is investigated from a number of different perspectives including application, technical infrastructure, number of players, resource orientation, social abilities, and degree of co-operation among the players. Furthermore, the same applications are subsequently re-defined by using more advanced utilities and capabilities. VR, VE and VR Systems are often used interchangeably in the Literature but there is a significant difference between the three. VR is a concept while a VE is the outcome of an application developed by using this concept. According to Baladi et al. [12], VR can be classified on the basis of technological standpoint and functionality. Pont [55] defines VR as a set of computer technologies including an interface to a computer through which users believe they are actually in a computer-generated world. For Bryson [23], VR is the use of various computer graphics systems combined with various display and interaction devices to provide the effect of immersion in an interactive 3D computer-generated environment where a virtual object (an avatar) has a spatial presence. An early definition of VE by Barfield et al. [14] describes a VE as being a computer-simulated world consisting of the software representation of real or imagined agents (avatars), objects and processes. Human computer interaction is an important part of the display and interaction with these models. However, Vince [221] states that a VR System comprises four components: a VE, a computing environment, VR technology and interaction modes.

Andreu et al. [6] explore the early developments of co-operative and collaborative work. According to them, a number of geographically distributed concurrent users utilise groupware or collaborative environments to achieve common goals with co-operation. These environments are called Computer Supported Co-operative Work (CSCW) environments that provide shared scenarios and interaction patterns. Andreu et al. argue that both cooperation and collaboration involve task distribution. The former divides tasks into a number of independent subtasks and the latter into entangled subtasks. A number of CSCW environments exist, such as Basic Support for Cooperative Work (BSCW) [74] and Groove [91]. The concept of these environments is integrated with VEs to develop more advanced collaborative spaces. VEs can also be categorised by using resource orientation and their underlying infrastructures, resulting in DVEs and NVEs respectively. DVEs are distributed in nature at both physical and application levels. However, both categories utilise network resources to provide a unified user view. Moreover, these environments are extended to CVEs by employing complex collaboration patterns.

CVEs allow physically distributed users to collaboratively create and manipulate content in a shared workspace. They also provide means for co-operative and social behaviours between the users. Co-operation is based on community concerns, and social behaviours are the outcome of complex interaction patterns. Common community interests lead us towards the construction of social spaces over the networks that allow globally distributed users to create and share content with each other. According to Thomas et al. [216], the World Wide Web (WWW), MySpace, and YouTube are the outcomes of common community concerns and goals.

VW is a relatively new terminology and includes distinguishing features over VEs, including collaboration and persistence. VWs and VEs are generally used alternatively by a number of researchers, but we believe that VWs are more sociable and provide in-place experience, in that the inhabitants live their lives according to their wishes. Moreover, VWs are generally multi-user imitating real worlds (such as Second Life (SL) [201]), whereas VEs can be single-user.

1.3 Applications of VEs

Due to the tremendous popularity of VEs and the availability of communication and computation facilities at affordable prices, many people are motivated to use different virtual applications for both fun and creativity over the Internet. Games, Defence, Graphics, Internet, High Performance Computing (HPC) and Social communities have leading roles in research into this area. A wide range of applications having a virtual existence, but the most widely employed applications include Games, Simulation, Augmented Reality, Training, Distributed Collaboration, Education, Design, Web, commerce, and visualisation [84, 205].

The **HPC community** work has roots in synchronisation algorithms [22, 36, 120] with a major concern to reduce execution time.

Early work of the **Internet and gaming community** includes a role playing game (Dungeons and Dragons) and a textual fantasy computer game (Adventure) [84]. Advanced computer graphics technologies have converted the initial efforts of this community into a huge video game industry today.

Early work in **simulation** was led by the military with the development of computer generated synthetic environments to simulate specific applications, and train individuals for complex and dangerous tasks. However, these environments were unable to handle group activities and interaction among Simulators (Sims) targeting specific applications [84]. The Literature shows a number of solutions to these problems including Distributed Interactive Simulation (DIS) [85] and High Level Architecture (HLA) [48], concentrating on the development of shared synthetic environments and interoperability among individual Sims with an emphasis on reusability. The DIS and HLA are well-known standards for distributed simulation that have been used and extended for diverse applications by different research communities.

Education has also gained much attention from VR communities. Early systems, using simple online methods, have recently been replaced with 3D virtual classrooms with an emphasis on interaction and collaboration. Currently, education systems are using distinctive features of VR including immersion [241] and presence [227] to help support in conceptual learning and problem solving. Roussou et al. [190] conducted experiments by simulating a school playground; children were asked for different activities, and both qualitative and quantitative analysis was performed. Their results show that full interactivity helped children in problem-solving. These results also lead to indications of conceptual change.

In architectural design, the VR enables users to visualise a design before scale models are built. Workplace design [50] and interactive design of manufacturing processes [119, 196] have been employed as VR systems. VR has a promising role in design visualisation and its inclusion in participatory design methodology. Mobach [150] has explored the use of VR in architectural design and organisational space design. He has also studied the integration of VR with participatory design. Besides design changes, the use of VR technologies improves staff satisfaction, and reduces costs [190].

This present study is based on interactive and collaborative social spaces that allow users to create and develop their content according to their wishes [165, 189], for example SL [201] and OpenSimulator (OS) [177]. Currently a number of communities, organisations and individuals are developing their virtual existence using VWs; this includes colleges and universities, businesses, charity and community services, and informative initiatives. Berger et al. [16] developed a collaborative virtual space for tourism, which adopted a community-driven approach with the basic aim of promoting a dynamic society of travelers. Travelers share travel experiences, recommend tourism destinations or just listen to each other for fun. The infrastructure is a game-like e-Business application where the e-Tourist is posed as an avatar. Furthermore, these worlds are persistence in nature.

1.4 Introduction to VWs

The introduction of the latest communication and computation technologies, the availability of personal computers at affordable prices, and the provision of impressive online content are major factors in the success of VEs, thus encouraging people to participate in a number of social and collaborative activities over the Internet. Currently, there is a great interest in recently emerged interactive and collaborative social spaces called Virtual Worlds (VWs) that imitate the real world and give a sense of place. Users in a VW are represented by virtual characters usually called avatars, and VWs allow users to develop content according to their wishes on virtual land that is purchased by spending inland virtual currency [189][165][201]. Users have many ways to become involved in creative activities while having fun. The great success of these environments has motivated researchers from different communities to introduce novel applications into these exciting infrastructures [165, 189].
Second Life (SL) and OpenSimulator (OS) are metaverse-like worlds that allow users to explore and create a dynamic 3D world. A metaverse is a shared space in which people interact and communicate through virtual avatars [37]. VWs are real time, interactive, and persistent in nature. They provide graphical user interface and manage a common shared space. The whole space is divided into square-shaped regions, typically 256m$\times$256m and each is managed by a separate Sim process. However, each Sim can potentially handle multiple regions. In these worlds, regions are placed adjacent to each other to create larger virtual spaces.
SL [201] is state-of-the-art in commercial VWs and unique in the sense that it is completely designed by its inhabitants. IBM (representing the WWW) and Linden Lab (hosting SL) are working together to achieve the goals of 3D Web. Their

basic aim is to make the WWW more attractive and interesting for the users [95]. In SL, each Sim is tied to a specific region of land and cannot be re-partitioned to cope with a changing workload. Using static assignment strategies, and due to its commercial background, it has a number of limitations. OS, an open source alternative to SL, has recently attracted much attention from developers and practitioners. The main reasons for their increased interest include the availability of complete source code and allowing different organisations to host their own content on their own infrastructures without paying for the service. Furthermore, it allows a Sim to run an arbitrary number of regions; therefore, making it more flexible and a better candidate for future 3D web. However, it has no means of scaling dynamically. Similarly, it compromises on consistency and cannot be used directly for applications of a conservative nature, such as for a business. Conservative applications require their activities to be processed in a correct temporal order.

Our work uses the following definition of VWs based on the description of Rosedale et al. (a pioneer of the SL Project) [189]:

A Virtual World is an integrated and persistent virtual space based on real and imaginary content. Users represented as virtual characters (avatars) feel immersed through tele-presence in a shared workspace which is geographically distributed both at infrastructure as well as application levels. Users collaboratively create and manipulate the content of world they inhabit.

Current VWs use a simulation-centric architecture, presented in Figure 1.1, that usually distributes simulation work to a set of Sims with homogeneous functionalities, where each Sim owns a shard or a region plus the complete set of simulation and communication work [140]. The key component of a VW Sim is the software which runs the simulation and the components that apply operations to objects and scene called actors [132]. These actors include storing objects, handling user interaction, simulating physics, running scripts, maintaining persistence, and generating updates for connected users. According to Lake et al. [132], the user experience is degraded if any of these becomes overloaded. According to them, actors are the main limitation in scaling a world and simulation-centric systems do not scale with additional resources.

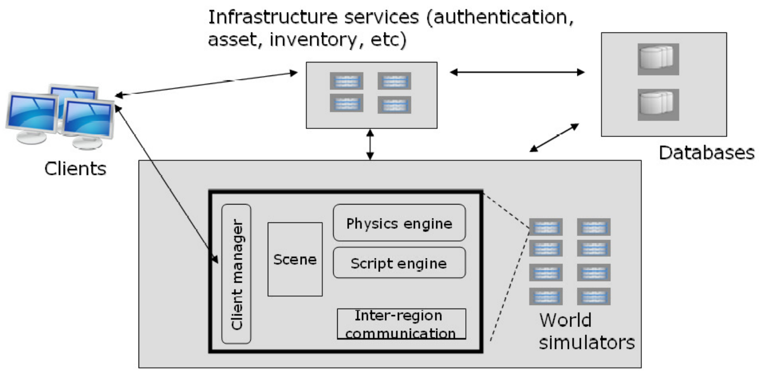

Infrastructure services (authentication, asset, inventory, etc)
Clients
Databases
Client manager
Scene
Physics engine
Script engine
Inter-region communication
World simulators

Figure 1.1: Simulation Centric Architecture [139].

Current VWs are much more sophisticated and immerse users using rich 3D graphics, detailed models of objects and realistic interactions. With this increase in richness and complexity of VWs, the required computation is also greatly increased. Online games like World of Warcraft (WoW) feature large virtual spaces, but the environment remains largely static and interactions are pre-determined. Game-specific optimisation techniques are applied to them and load per user is relatively low. On the other hand, general purpose VW such as SL and OS put fewer constraints on real-time content creation and interaction for services such as social networking and collaboration, scientific experimentation, e-commerce, marketing and games. Therefore, the computation load is much higher and it is difficult to predict it in advance. As the number of users, or scene complexity, increases, the computation and communication load on per-user basis grows with it [132]. Current approaches used for VW try to accommodate the limitations in architecture that could limit the number of concurrent users.

To resolve these issues, a specialised infrastructure is required to handle these kind of worlds. The infrastructure should be able to dynamically add and remove resources based on load, thus solving resource under-provisioning and over-provisioning problems. A number of VWs with similar themes exist with different concerns targeting different audience such as Kaneva [124], ActiveWorlds [211], Barbie Girls [212], Club Penguin [213], There [214], Forterra Systems [208], Gaia Online [166], Habbo Hotel [98], Neopets [157], Whyville [233], and Zwinktopia [215]. A brief introduction and a comparison of these worlds is presented in [40].

1.5 Scalability, Load Distribution and Consistency Issues

Scalability and Load Distribution

Chang et al. [37] argue that a significant problem in designing a 3D virtual world is how to develop a scalable architecture that can manage a large number of concurrent users in an interactive 3D environment. Scalability, therefore, requires powerful computation and communication infrastructures. VWs must scale in the following dimensions: number of concurrent users, scene complexity, and fidelity of user interaction [140]. However, all of them can quickly overwhelm the system and need to be carefully handled without degrading overall performance and interactive user-experience. VWs can be scaled by a flexible and scalable architecture, dynamic load balancing, and reducing redundant communication and computation in delivering views to users [140]. According to Lee et al. [134], the design of a scalable network architecture for VW needs consideration of a Communication Architecture, Interest Management, Concurrency Control, Data Replication, and Load Distribution. Moreover, while improving scalability, a VW still need to deliver its unique features [204].

To deal with scalability and improve performance of a VW, a number of parallel and distributed infrastructures are currently in practice. A typical approach is to partition a space into a number of regions and execute it with the help of a number of systems typically called a Parallel and Distributed Simulation (PADS) system. Common approaches to divide a VW workload across multiple servers use either sharding or spatial partitioning. In sharding, different replicated copies of the same virtual space are created on different servers and there is no interaction among the players in different shards. In spatial partitioning, each server is statically assigned a part of the whole world. SL and OS normally run spaces based on 256m×256m square spaces called regions. Each server is responsible for everything that happens in the space that it is handling [132]. The Literature shows a number of approaches dividing a space into regions of different shapes, such as circles, triangles, rectangles or hexagons. However, the selected pattern needs to cover the whole content with minimum effort and without overlapping.

Region-based schemes using spatial partitioning can be categorised as static or dynamic. Both Butterfly Grid (BG) [103] and SL Grid (SLG) [197] exploit region-based static assignment strategies. To eliminate problems in static configurations, a number of local, global and adaptive dynamic strategies have been developed for load distribution that have their benefits and limitations [136, 143, 159]. Vleeschauwer et al. have achieved better load distribution but with a significant increase in communication between cells [223]. Shirmohammadi et al. [203], Chertov and Fahmy [39], and Morillo et al. [153] have also attempted to address the issues of scalability and load distribution. A number of hierarchal infrastructures can be found in the literature addressing the same issues [13, 24, 51, 127]. However, they have major performance issues when used to host a VW because they are basically developed for games. A detailed analysis and comparison of the existing mechanisms is given in [61]. It is believed that VWs are more sophisticated than games that lack pre-defined rules, and both latency and consistency are prime concerns for these environments.

Consistency Management

Since PADS systems are executed with the help of a set of dedicated computers, they are scalable and provide a better interactive experience. However, they suffer from another challenging issue of consistency and their performance is degraded when conservative approaches are used for time synchronisation [83]. Time Management (TM) is the process of maintaining the temporal order of events in a system. Synchronisation mechanisms can be broadly classified as being either conservative or optimistic. According to Fujimoto [84], both of these have their benefits and limitations, and selecting an approach depends on a target application. TM can be implemented by either a centralised or distributed approach. Further, a TM algorithm can be either synchronous or asynchronous [178, 179]. Existing algorithms have shown great success for their target applications but they have major performance issues when used for VWs. In particular, they have potential bottlenecks when used with complex hierarchical models representing worlds based on partitioning algorithms. The existing VWs, therefore, restrict their application domain and rely on traditional methods for conservative applications.

To cope with scalability and consistency issues both in current static and dynamic systems, a contemporary infrastructure has been developed which targets these issues using the inherent properties of VWs. This is introduced in chapter 2 after a detailed analysis of both scalability and consistency frameworks.

1.6 Research Approach

This study focuses on scalability (load distribution) for VWs and TM for consistency in VWs.
In the first phase, the existing scalability and consistency infrastructures for VWs are investigated and their potential bottlenecks are identified.
In the second phase, the following methods are developed to resolve the identified limitations and they are validated by conducting proof-of-the-concept simulation studies.

- A Joint Hierarchical Nodes Based User Management (JoHNUM) infrastructure comprising of partitioning, assignment, and merging to dynamically scale VWs.
- An Aggregate Region Assignment (ARA) algorithm to obtain a fair distribution of load among Sims using a set of aggregation strategies.
- A decentralised synchronisation approach to maintain a consistent view of the VW.

In the third phase, JoHNUM infrastructure and ARA algorithm were implemented as a Plug-in for OS. First these were implemented on a Windows platform using a small network, and later on, they were extended to scale them further using a cluster environment.

1.7 Contribution

This thesis aims to make the following contributions:

- Development and Simulation Studies
  - Development of a JoHNUM infrastructure to dynamically scale VWs and resolve issues in current static and dynamic methods. A simulation study was conducted to compare it with a game middleware Matrix [13]. JoHNUM strategies showed an improved performance over Matrix.
  - To minimise resource utilisation and reduce implementation and communication costs, an ARA algorithm was developed which uses a number of aggregation strategies to obtain contiguous areas for assignment. It is compared with hierarchical models for complexity and delay based on intermediate points using an abstract communication model. A simulation study shows that ARA algorithm and aggregation strategies perform well for worlds of all sizes.
  - A fully-decentralised TM approach was developed adopting a constrained Peer-to-Peer (P2P) communication model using the inherent properties of VWs to make them consistent and conducted a simulation study that validated the correctness of our approach. It is compared with hierarchical models for complexity and delay based on intermediate points using an abstract communication model.
- Implementation and Evaluations
  - The current architecture of OS framework was examined and an extended architecture was presented to incorporate features for scalability, load distribution, and consistency based on our methods.
  - The ARA algorithm was extended by adding a flood-fill algorithm to make it capable of determining valid combinations for worlds with various shapes. Merge operation also utilises it to maintain continuous spaces.

- Based on our investigation, a load model was developed which is used to determine the points when a split or a merge operation is initiated. A plug-in for OS framework is developed that implements our scalability and load distribution algorithms based on an abstract framework.
- This work is evaluated both on a simple Windows network and a Linux cluster environment. A detailed analysis and comparison of our work on both environments is provided, using a number of time statistics. Two merging strategies have been developed and compared for trade-offs between number of resources and transferring the same content multiple times.
- Two improved strategies have been developed to remove a region from a Sim with direct database access for cleaning up the data. These take advantage of using OpenSim Archive (OAR) functionality to achieve updated content, and they significantly improve a region transfer time.
- A reasonable set of experiments was conducted to compare this work with traditional methods. It showed significant improvements over both static and dynamic approaches. A number of bugs in the OS framework were also fixed by writing explicit methods.

1.8 Thesis Organisation

This thesis comprises 8 chapters and is organised as follows.
Chapter 1 briefly introduces the background of this research and the scalability, consistency, and load distribution issues for VWs. It provides a description of the research methodology adopted for this work and of the contributions made to the field of VWs in this thesis.
Chapter 2 examines the current scalability and consistency frameworks handling VWs and presents their key limitations. It provides an analysis of the underlying technical infrastructures for VWs. The scalability, load distribution and consistency methods developed in this work are introduced in this chapter. It also briefly describes a set of open source frameworks that are used for the development of VWs, including OS which is used for the implementation of our work.

Chapter 3 presents the JoHNUM infrastructure that was developed to scale VWs. It provides proof-of-the-concept simulation performed in MATLAB. It further explores the ARA algorithm and aggregation strategies that are used to determine a fair distribution of load and contiguous spaces for load distribution. It illustrates the aggregation strategies and provides simulation results for experiments carried-out in MATLAB. To show the potential benefits of contiguous spaces, it presents an abstract communication model.
Chapter 4 presents the novel decentralised consistency mechanism that adopts a constrained P2P communication model. It illustrates the concepts and describes a simple simulation model that tested the validity of this method.
Chapter 5 discusses the OS framework in detail and provides its limitations. It illustrates the extended architecture of SL as a reference. It examines the extended OS architecture, which adds scalability and consistency features to current OS framework based on our mechanisms.
Chapter 6 reports on OS capabilities with an extensive set of experiments for different concerns. It presents a load model that is developed and used for taking split and merge decisions on both Windows and Linux platforms. A scalability model that identified the components required to implement our work is introduced. An informal time analysis model was used to determine the capabilities of both Windows and Linux environments for transferring regional content. It also examines two improved strategies to reduce the time taken by removing a region and compares them with basic OS operations. Bug fixtures are also presented in this chapter.
Chapter 7 provides an abstract framework describing the way both JoHNUM strategies and ARA algorithm are implemented. It presents the limitations in the basic ARA algorithm and proposes an extension to fix them. Merging strategies are also presented and compared for trade-offs between resources and multiple transfers. A number of experiments are detailed to illustrate both scaling and merging processes and to compare this framework with traditional systems. Worlds of varied sizes were used and a range of statistics were used to compare this work with existing mechanisms.
Chapter 8 concludes this study and gives future directions for this work. It also outlines the strengths and limitations of this work.

Chapter 2

Background and Motivation

This chapter explores the existing underlying technical infrastructures, and current scalability and consistency mechanisms that are used to manage existing VWs. It examines a number of mechanisms targeting load distribution in a multiuser environments and gives the motivation and goals of the work presented in this thesis. It also gives a brief analysis of the open source frameworks that are used to develop VWs, including the OS framework which is used to develop a prototype of our work.

2.1 Underlying Technical Infrastructures

A VW infrastructure encompasses three parts: a service provider, the communication infrastructure (the Internet), and client software. A service provider needs a flexible and robust server architecture to run a scalable and consistent VW. The Internet is used for communication purposes and a client module implements a number of computationally intensive tasks such as rendering. Currently, the server infrastructures are becoming a bottleneck after a tremendous increase in bandwidth, communication speed, and the availability of cheap commodity systems. Video streaming servers such as "YouTube" and "Google Video" respond slowly during peak hours, highlighting this fact about servers. Therefore, more advanced, flexible and resilient server infrastructures are required. The existing infrastructures can be classified broadly into three categories: Client Server (CS), Cluster of Servers (CoS) and Distributed systems.

CS environments guarantee Quality of Service (QoS) and provide persistent, consistent and secure worlds. However, these provide limited services to a restricted number of users and are, therefore, static and not scalable. Moreover, they are prone to single point failure. [161] discusses a number of measures for handling these issues. As a result, the most successful highly interactive and computation intensive applications use CoS infrastructures as an alternative. However, consistency and intra-server communication are the main problems of these infrastructures, which also suffer from load balancing problems in static configurations that are simple and easy to implement. [70] outlines a number of dynamic strategies to solve these problems. A CoS environment provides a centralised user interface that becomes a bottleneck in case of failure. Generally speaking, CoS systems consider closely coupled resources of an organisation [103, 188, 189, 226] and usually adopt static configurations. Zihui et al. [87] presented a dynamic CoS architecture for multimedia applications, where they proposed a category-based dynamic coalition of servers that achieves a high level of scalability and the efficient utilisation of resources. Since the popularities of video files are normally skewed and unpredictable, the infrastructure needs to be highly dynamic in terms of resource allocation in time varying workloads. Being adaptive in nature, it also exploits temporal locality and users are served by a server already serving similar requests.
Distributed infrastructures leverage geographically distributed resources and provide an integrated seamless VW. According to Wang et al. [226], distributed systems are further classified as P2P and Client Multi Server (CMS) architectures. CMS systems are alternatively called Grid infrastructures (we call them static grids, for dynamic grids, see Appendix A). P2P systems are generally based on commodity PCs and not used for computationally intensive applications such as games. These are more scalable and affordable but experience a highly dynamic topology that introduces the well known neighbour discovery problem. Therefore, topology management in P2P systems requires extra effort. The complexity for the development and management of P2P architectures for large scale VWs is very high. It also requires handling issues such as data security, inter-operability, and

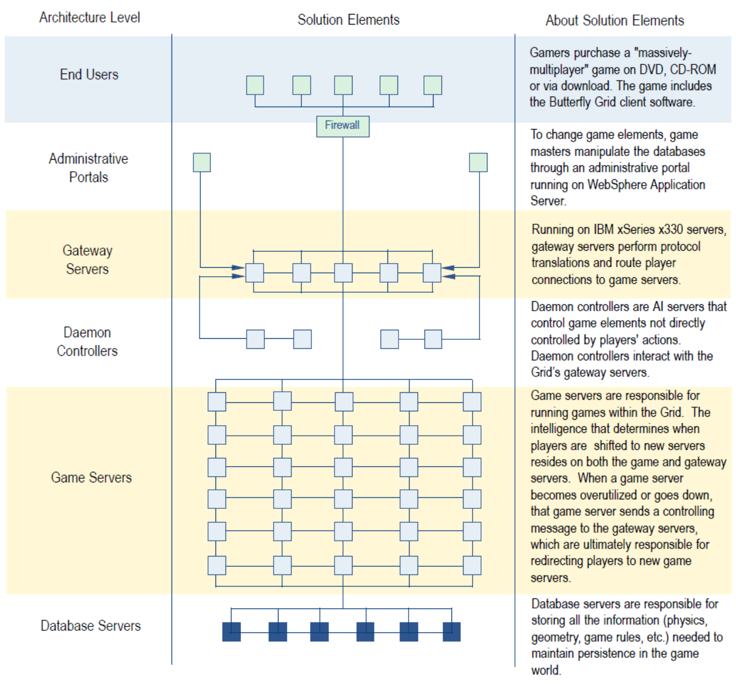
network control [152]. The Literature shows a number of P2P architectures that are developed for network games [17, 18, 93, 105, 128]. Similarly, the variations of P2P infrastructures targeting their inherent problems are presented in [35, 101]. P2P networked game infrastructures generally focus on synchronisation of game states across the hosts. These techniques normally utilise the concept of Distributed Hash Table (DHT) for this purpose. Knutsson et al. [128] proposed a Massively Multiplayer Online Game (MMOG) model using a publish-subscribe mechanism. Similarly, Castro et al. [34] proposed a large-scale and decentralised application-level multicast infrastructure called Scribe. Bharambe et al. [18] developed a routing protocol similar to DHT for multi-attribute queries called Mercury. They claim that MMOGs are less restrictive in terms of latency so that multi-hop latency of underlying DHT look-ups can be tolerated. In order to speed up searches, they subsequently proposed another mechanism called Colyseus [17], where they employed the principles of locality and prediction in accessing data patterns. Hu et al. [101] proposed a Voronoi-based Overlay Network (VON) that maintains a fully distributed topology. It experiences low latency and efficiency in messaging. Chan et al. [35] used a contemporary P2P approach as an alternative solution for MMOGs called Hydra. It utilises a message consistency protocol and is based on an augmented CS programming model in conjunction with a set of protocols for realising interfaces. According to Chan et al., the adaptation of a hybrid approach solves issues such as billing and persistence. Hydra is scalable and introduces little message overhead.

CMS/Grid infrastructures are loosely coupled environments that use the resources of different organisations and individuals based on common agreements over the Internet. CMS architectures generally follow the grid model. Grid infrastructures are more dynamic and resilient, and the majority of successful MMOGs and VWs including SL [197, 201] are hosted by these infrastructures. Multiserver architectures are developed for scalability but they experience significant workload imbalance due to the dynamic and unpredictable nature of humans. They also suffer from latency and delay, and require sophisticated dynamic load-sharing and transferring mechanisms to achieve quick responses and maximise throughput [27, 35, 189]. Butterfly Grid (BG) solves a number of problems of conventional systems by providing a flexible, scalable, and resilient MMOG infrastructure for game developers and service providers [103]. The underlying infrastructure of SL [201] called SL Grid (SLG) [197] customises the BG model with a constrained communication model.

2.2 Current Scalable and Load Distribution Mechanisms

A number of approaches have been developed which investigate scalability, consistency and load balancing issues. Major contributions are based on splitting a VW, balancing the load, and developing communication models to minimise network flow. These mechanisms consider different parameters such as computation power and network bandwidth. BG [103] and SLG [197] are commercial Grid infrastructures running different MMOGs and SL [201] respectively. BG is comprised of two clusters where game and database servers are fully meshed over high speed fiber-optic lines for transparent movement of users between servers. It divides a game world into a number of mutually exclusive sectors known as locales and assigns each to a specific server on the grid. It utilises the grid model to seamlessly route contents and players to the nearest available server in case of failure or excessive load [103]. Figure 2.1 shows the basic architecture of BG highlighting different levels of services. It also gives a description of the activities and services at each level. SLG is a customised form of BG that restricts the interaction of a server to four neighbouring servers [197]. However, both decrease the interactive user experience and the migration of users incurs processing burdens. Furthermore, no means are provided to cope with under-provision and over-provision of resources. In case of over-utilisation, these systems fail quite frequently.

Load distribution strategies are categorised as static and dynamic. Static strategies do not provide any means to cope with excessive load and are not generally scalable. Both BG and SLG use static assignment strategies for content. Dynamic load distribution strategies generally belong to one of the three classes: local, global, and adaptive. Local approaches consider neighbouring servers to share excessive load and minimise communication between the servers. Ng et



Gamers purchase a "massively-multiplayer" game on DVD, CD-ROM or via download. The game includes the Butterfly Grid client software.

To change game elements, game masters manipulate the databases through an administrative portal running on WebSphere Application Server.

Running on IBM xSeries x330 servers, gateway servers perform protocol translations and route player connections to game servers.

Daemon controllers are AI servers that control game elements not directly controlled by players' actions. Daemon controllers interact with the Grid's gateway servers.

Game servers are responsible for running games within the Grid. The intelligence that determines when players are shifted to new servers resides on both the game and gateway servers. When a game server becomes overutilized or goes down, that game server sends a controlling message to the gateway servers, which are ultimately responsible for redirecting players to new game servers.

Database servers are responsible for storing all the information (physics, geometry, game rules, etc.) needed to maintain persistence in the game world.

Figure 2.1: Basic architecture of Butterfly Grid [103].

al. [159] have developed a region based DVE called CyberWalk by exploiting a local strategy. These are simple to implement but perform well only with a minimum number of servers and do not scale well. Functionality decreases greatly if neighbouring servers become overloaded. Global strategies utilise global information to re-distribute the overall load uniformly among the servers as one server becomes overloaded. Lui and Chan [143] have proposed an efficient partitioning algorithm for DVEs using a global strategy. Migration of users and re-partitioning overhead rises with an increase in content and number of servers that degrades interactive user experience. Furthermore, an exponential increase in complexity renders them unsuitable for large scale real time interactive systems. To resolve these issues, adaptive strategies re-partition the load of selected servers only. These are more effective and introduce less overhead. The communication costs in adaptive approaches are greater than for local, but less than for the global strategies. In adaptive approaches, an overloaded server balances its load by utilising a set of servers beyond the neighbouring servers if the neighbouring servers are busy. Lee and Lee [136] have devised an adaptive strategy for load distribution in DVEs.

They presented an architecture that divides the world into a rectangular grid with cells representing virtual spaces. A graph partitioning algorithm is used to allocate cells to the servers. This approach is suitable for those simulations with known scale, but difficult to adapt to an un-constrained simulation environment such as a VW. The importance of minimum latency and the utilisation of fewer resources is highlighted by Lee et al. [135] in their investigation of the client assignment problem. It increases overhead by implementing the dynamic concepts described as 'zoom in' and 'zoom out'. According to Ta et al. [209], position-based client assignment strategies degrade interactive experience and they proposed greedy algorithms to cope with client assignment. However, greedy approaches are unable to serve real time systems as these require periodic re-execution that increase system load.
According to Vleeschauwer et al. [223], the larger cell assignment strategies suffer from player density in peak hours. They proposed a split of the VW into a large number of interacting micro cells. These cells are dynamically assigned to a set of servers, thus achieving better balance of the load against static and dynamic mechanisms. However, it greatly increases communication among the cells. Chertov and Fahmy [39] have proposed a load balancing layer that facilitates server co-ordination for quick convergence to a best possible distribution of load. However, this is not scalable as they adopt a centralised approach. Shirmohammadi et al. [203] have developed a large scale collaborative architecture that divides the VW into multiple adjacent hexagonal regions. However, in case of further re-partitioning, it considers the whole VW to divide it into smaller regions. The use of a global partitioning strategy increases complexity and degrades interactive user experience. Moreover, the re-partitioning could occur frequently, thus making their approach unsuitable for large scale real time interactive and collaborative systems. According to Ahmed et al. [3], CS architectures are not scalable and they proposed a dynamic interest management and collaboration model using P2P model for games. Furthermore, they presented a load balancing model over this infrastructure [4]. However, we believe that P2P environments are not suitable for real time interactive applications. RING [86], NetEffect [49] and CittaTron [96] split a VW into subregions to make them scalable. RING uses a static assignment strategy and the rest adopt dynamic split strategies at servers. Ac-

cording to Morillo et al. [153], region based schemes do not need synchronisation but require intelligent load balancing mechanisms to achieve a reasonable response time and maximise the throughput. These systems are less scalable due to highly unpredictable user patterns in MMOGs and VWs. RING [86] provides no means to handle excessive load. To balance the load, NetEffect [49] utilises a centralised approach that greatly degrades interactive user experience. CittaTron [96] also degrades performance because it allows run time resizing and transfer of users. Moreover, factors such as object density and locality are not considered.

The Literature shows a large number of solutions for large scale simulations [84], simulations with distributed interactions [130], and real time simulations [57, 154] to cope with these issues. However, the VWs combine the challenges of all these hard simulation problems including scale (order of magnitude larger than traditional PADS), being perpetual (continuous existence) and real time in nature, and need to produce rapid responses to user inputs. According to Liu et al. [140], the existing VWs usually adopt a simple synchronisation model to accommodate large scale environments and therefore events are not processed in a correct order. VWs perform both simulation and visualisation of viewpoints for a large number of users simultaneously, and put an enormous burden on both computation and communication infrastructures. To address these challenges, researchers have developed solutions based on Interest Management [210] and visibility computation [131]. They also incorporate a number of heterogeneous engines ("the actors") to simulate and evolve the work, each having a different scope of operation, resource requirements, performance constraints, and operational characteristics [180].
According to Liu et al. [139], the dynamic load balancing mechanisms such as distributed binary space partitioning (BSP) hold the potential to scale a VW. They are simple and effective, and have the capability to resolve hotspot issues. However, they introduce excessive burden due to workload migration and communication between the regions. VWs have both computation and communication bottlenecks in terms of scale. In addition, the migration of scene objects has been overlooked by most of the previous studies. Static assignment methods avoid it by using a static and strict partitioning method, and academic studies focus on the migration of client connections but ignore object migration. According to Liu et

al. [139], the current VWs need to scale beyond the existing capabilities to cope with rich user experiences, greater realism, and new dimensions. They presented a new architecture called Distributed Scene Graph (DSG) to overcome the limitations that are introduced due to the use of simulator centric architectures [139]. This architecture is presented in Figure 2.2. The key idea is to break the simulation centric architecture used for the current VWs and detach data structure from simulation engines [140]. This also separates the client interaction from the scene, thus reducing the scene complexity that greatly helps to scale the world. DSG allows multiple servers to host a scene where each server is managing part of the scene. Scene is no longer a centralised and monolithic process that manages simulation as well as data management. It is left to focus on data management, state synchronisation, event distribution, and persistence of the world content. DSG prototype is developed as an extension to OS framework and it has greatly improved the capacity compared with a basic Sim of OS. However, it introduces an additional layer that handles an increasing number of client managers based on load [132] that could potentially degrade interactive user experience and increase delays. Furthermore, the overall system is more complex to implement because it requires the provision of additional interfaces between the scene and actors [132].

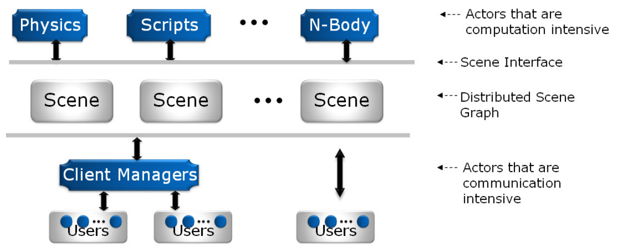


Figure 2.2: Distributed Scene Graph (DSG) Architecture [132, 140].

The most common approaches to dividing a VW workload across multiple servers use either sharding or spatial partitioning. [132]. The capacities of SL [201] and WoW [164] are limited to well below 100 interacting users [92]. Games such as WoW [164] and Ultima Online [217] use the concept of sharding to scale a VW, but at the price of user interaction. It is the most popular approach that broadly partitions the user base into disjoint copies of the world. Replication in this model is easy as users in different shards cannot interact with each other due to the lack of means of interaction with each other. VWs such as SL and OS use the concept of spatial partitioning. Lu et al. [142] have presented a load balancing technique for a CoS infrastructure. It is simple and effective and maintains the flexibility of the cluster systems; however, grid infrastructure and dynamic strategies are considered to be more resilient and not prone to single point failure issues. Moreover, the system spends a great deal of time in balancing the load, and the issues regarding frequent migrations of load and communication are overlooked. Chen et al. [38] have presented a dynamic architecture for MMOGs that hides the division of a world from the players. Interactions between players is, therefore, not limited to the objects in a single region or server. Their major contribution is also a load balancing algorithm using locality awareness to improve average response time for users. Varvello et al. [219] conducted a detailed survey about SL to determine the worth of this exciting social collaborative environment and to investigate how well it is utilised by users and which kind of content is favourite among the players. Based on their findings, they have presented techniques and ways of further enhancing these worlds.

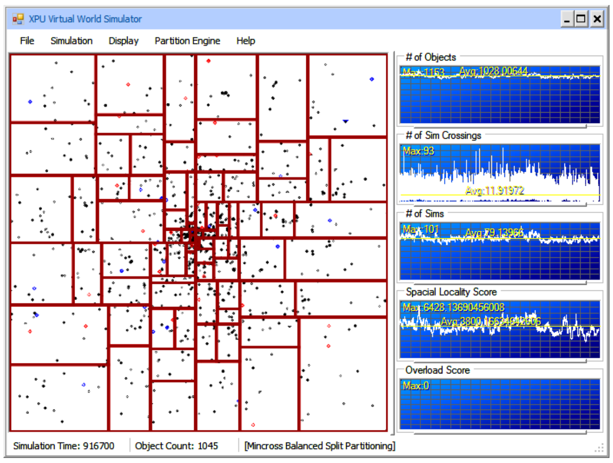
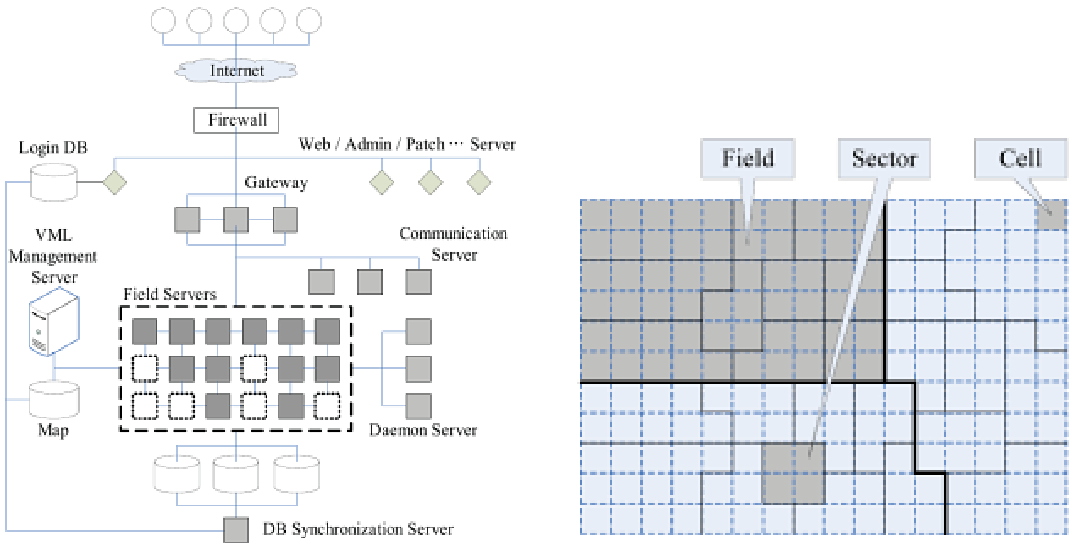


Figure 2.3: A screenshot of the XPU simulator. The dots represent objects, and solid lines represent boundaries between partitions [37].

Francis et al. [37] presented a hierarchical CS architecture called Extremely Partitioned Universe (XPU) for the development of highly scalable metaverses using a spatial sub-division algorithm. It dynamically partitions the world and manages network and computing resources. The basic goals of this architecture include the use of CS environment for security reasons, using a varied number of resources based on current load, and to manage a large and unpredictable population. It has the ability to handle both flash crowds as well as vast unused or sparsely populated spaces. This architecture only addresses the problems of managing 3D virtual space and the objects contained in it. XPU architecture uses the XPU tree, which is very similar to a k-dimensional (k-d) tree where the leaves represent virtual 3D spaces instead of objects. A k-d tree is a space partitioning data structure for organising points in a k-dimensional space [15]. The root node in the XPU tree represents the whole simulation process managing an entire XPU universe. Splitting and merging are the two significant operations in managing XPU systems. When a simulation system is overwhelmed in terms of system load, it can choose to split its workload between two child Sims by selecting them from the available pool of resources. Merging is much simpler as, when two sibling leaf Sims have a smaller workload, they can choose to simply synchronise their states and revert processing to a single node. Partitioning of borders in XPU is dynamic and reactive to load. If work is unbalanced, a child always exchanges load to balance it. The main flaw in this system is that it takes a great deal of effort to balance the load, thus degrading the interactive user experience and increasing complexity. There is no limit on the levels in a Resource Management Tree (RMT). On the other hand, a hierarchical structure could face severe synchronisation issues when used for conservative applications. Figure 2.3 shows a snapshot from the XPU simulator in action. It allows partitions of any size based on load that greatly increases communication and crossings between the Sims. Furthermore, no prototype has been developed to determine the actual worth and limitations of this work.

Croquet [44] is a decentralised approach that exploits a P2P synchronisation protocol to manage the content of a virtual space spread over a number of Croquet worlds. It allows access between different worlds, but no dynamic mechanisms are provided to extend the number of concurrent users in an individual world and it is, therefore, not scalable. Similarly, it is difficult to manage the environment using a P2P model. A number of other open source VW development frameworks are provided in section 2.4 that are briefly analysed against the requirements established in this study to obtain the best possible framework for the implementation of our work. Active Worlds [211] is another type of metaverse that allows dynamic content to be created and, for this reason, provides a simplified scripting interface. An active world universe can host hundreds of worlds that can be traversed by a user where each world is hosted by a different server. However, neither does it provide any dynamic solution to cope with under-utilisation or over-utilisation of resources and is not scalable as each world can only have a limited number of players. A number of other similar VWs are presented in section 1.4 of chapter 1. Presetya et al. [181] compared a number of topologies for fixed grid spatial subdivisions such as triangular, square, and hexagonal with their own method called brickworks for gaming environments. However, these systems are not as scalable as spatial subdivision approaches using hierarchical approaches. They either miss the dynamic allocation of resources or involve moving server processes around so that unloaded servers can share a single CPU, thus degrading interactive user experience. The Sun Gamer Server technology (called the Darkstar Project) [207] framework adopts a different approach to traditional mechanisms. It does not utilise spatial sub-division; rather, it uses a high speed centralised database holding the whole virtual space. Each server has access to world objects and rights to modify the objects in this database. For each operation, a server retrieves an object, modifies it and then stores it again in the database. This architecture introduces extra delays, especially when there are many objects involved in an interaction. A centralised database might easily become a bottleneck as the system scales. The ALVIC (Architecture for Large-Scale Virtual Interactive Communities) approach for metaverse design uses quad tree subdivision for partitioning logic servers and employs many proxy servers to hide network topology from the servers [184]. These introduce an additional layer, thus increasing delays and complexity.

Kim and You [125] have proposed a hierarchical map partitioning method for MMOGs by introducing a Virtual Map Layer (VML). Figure 2.4(a) shows the traditional network server architecture with an additional component called VML management server. It checks the loads of field servers and tries to divide or merge the fields based on current load. VML is an overlaid version of map that keeps information about hierarchy of sub-areas. It splits the VW into a hierarchy of Fields, Sector Groups, Sectors, and Cells as shown in Figure 2.4(b). A field is what is assigned to a server and is comprised of sectors or group sectors. The VML management server divides or merges fields based on server load of its sectors and sector groups. A sector is the smallest unit for partition that is comprised of the smallest unit of VML hierarchy, called cells. A sector is a combination of adjacent cells and greatly varies in size based on client capacity. However, it increases overhead by introducing an additional mapping layer, and a single management server is prone to a single point failure. Furthermore, the maintenance of the hierarchy is complex and requires additional resources.



(a) (b)

Figure 2.4: Illustration of hierarchical map partitioning [125]. (a) VML based MMOG system. (b) Hierarchical structure of VML.

Burlamaqui et al. [24] presented a communication infrastructure by extending the CS model to a hierarchical one in order to scale large scale collaborative VWs named H-N2N (hierarchical N to N). It scales the world by geographically partitioning it into groups where each group is managed by a server that keeps it together. It starts with a single application server and the clients constitute the first group hosted by the first server at level 0, as shown in Figure 2.5. When this server becomes populated, a new group is created, which is assigned to a new server that is placed at level 1, which establishes a link to the first server.

To balance the load between servers, the clients are often re-grouped. The localisation matrix is used to keep track of users and the proximities among the users. H-N2N architecture is comprised of 4 components: ApplicationServer, GroupServer, SlaveServer, and UserClient. The ApplicationServer component awaits client connections and the GroupServer component handles message exchange among clients. The SlaveServer is responsible for communication with users and UserClient represents a user. These components and their placements at different levels are illustrated in a simple configuration adopted from [24] in Figure 2.5. A single application server might become a bottleneck in case of failure and re-grouping of space can greatly degrade the interactive user experience. In addition, sending messages across groups might introduce longer delays. Furthermore, a client might not be able to cope with as great a massive number of exchanged messages as a GroupServer. Similarly, it puts no restrictions on the levels in a resource hierarchy. A similar approach is proposed by Oliveira and Georganas [51], which manages servers in a parent child hierarchy. It also redistributes the load among the servers to balance the load, thus degrading the overall performance.

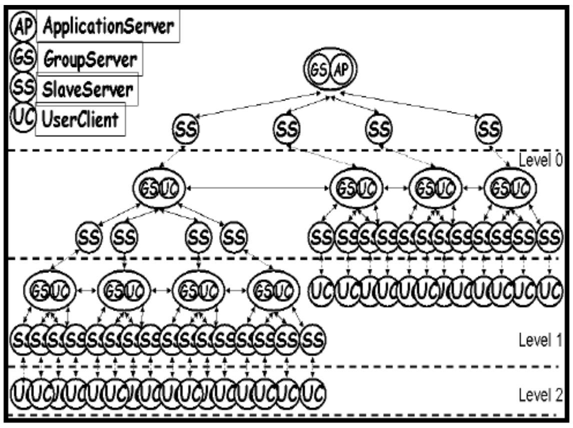


Figure 2.5: Hierarchical N to N (H-N2N) architecture [24].

Wang et al. [226] proposed a novel, dynamic idea of a multi-server model based on a grid structure. It uses the concept of "gamelet" that provides execution logic for a server. The logic is divided into data and processing. Data include the current state of the simulation work and performance parameters, such as

CPU and network load. Processing provides the logic that is executed to perform activities and its control functionalities. A gamelet provides a complete scenario that could be based on space or time, such as a meeting or a football match. Novelli et al. [162] presented a technical mechanism that utilises the grid concept for content distribution of multimedia applications. Grid infrastructure is used for both Replica Storage and the computationally intensive task of transcoding that changes content format for compatibility. Bruneo et al. [21] proposed a grid middleware to integrate distributed computation and storage resources, thus hiding the locations of power and data. This work also targets the utilisation of grid for multimedia applications.

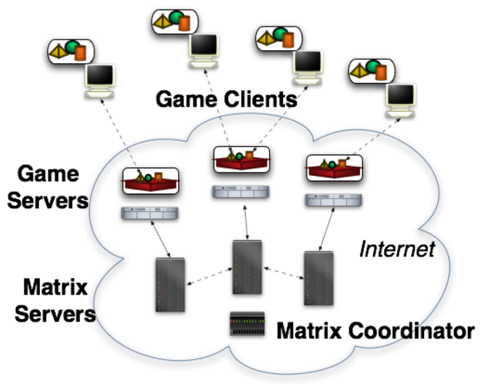


Figure 2.6: Matrix architecture [13].

Matrix is the outcome of a major contribution towards the scalable MMOGs by Balan et al. [13]. It is a game middleware that achieves low latency while providing localised consistency based on dynamic workload compared with P2P and static infrastructures. The authors assumed that a game can be decomposed into different stages and only localised consistency can achieve better results. Matrix changes a region size and the number of serving nodes at run time. The architecture of Matrix comprises three layers: game clients, game servers and matrix servers with a matrix coordinator as shown in Figure 2.6. Matrix simply takes local decisions for partitioning. Therefore, when an overloaded server is detected, a new matrix server is selected which further selects a new game server and shares the load with it. The new matrix server becomes a child of the matrix server that initiated the split. In case of under-utilisation, a matrix server reclaims the parti-

tion and game state from the child matrix server which releases the game server. It concentrates on achieving low latency but compromises on consistency. It degrades interactive experience and yields an RMT of many levels. In our opinion, to achieve a consistent VW, the levels in RMT need to be minimised. Further, the Matrix Coordinator (MC) is prone to failure, though its complexity is negligible. Similarly, it uses two different levels of servers (Matrix and Game) which increases system complexity.
The motivating factors, together with the ways to achieve the goals set for the JoHNUM infrastructure and ARA algorithm, are presented in the next section.

2.2.1 Motivation and Goals of JoHNUM Infrastructure

This work investigates current systems using static, dynamic and hierarchical approaches to make VWs scalable and to distribute load among the participating servers.
Current CS, CoS, and Distributed systems (classified as P2P and CMS/Grid infrastructures) have their strengths and limitations, which are presented in section 2.1. Grid infrastructures are the most favourable choice for hosting applications that have high computation and communication demands, such as online games and VWs. However, static assignment strategies limit their capabilities by introducing both resource under-provisioning and over-provisioning problems. Dynamic load distribution strategies (local, global and adaptive with a flat orientation) are used to overcome these issues to some extent. Local strategies (in which a server shares its load with only adjacent servers) are not scalable, and global strategies are too complex and introduce a significant burden of player migration and re-partitioning. Similarly, adaptive strategies increase the implementation overhead. Dynamic hierarchical strategies (such as Matrix [13]) overcome many issues in static and dynamic methods but put no restrictions on an RMT, which could potentially introduce longer delays in certain situations (especially when used with conservative applications, which are further examined in section 2.3).
Game infrastructures are currently used to host VWs that have major performance issues. Performance issues in current static, dynamic and hierarchical strategies and the un-availability of a specialised framework to scale VWs motivated us to

develop a Joint Hierarchical Nodes Based User Management (JoHNUM) infrastructure. Similarly, to overcome the complexity and implementation issues in load distribution mechanisms, an Aggregate Region Assignment (ARA) algorithm is presented that significantly reduces resource utilisation and communication overheads.
VWs (the target application in this work) are more advanced environments than games, and both latency and consistency require special attention while scaling a world. Therefore, game specific techniques such as sharding are of no use for VWs as they conflict with the basic aim of these environments. Similarly, other approaches for scalability using different aspects of a system (such as DSG architecture [132, 140]) introduce additional layers and increase complexity. Spatial partitioning is the most promising way to partition and transfer the content to other systems for general purpose VWs [132]. However, it is an expensive operation because it transfers the contents as well as players and, therefore, requires better strategies to reduce the time taken by different activities. The OS framework (used for the prototype development) and its modular design greatly motivated us towards using the traditional spatial partitioning to scale the worlds. It enabled the transfer of regions in a delegated space in turn, which greatly reduces the content un-availability time. It further helped to minimise the total time and number of players that suffer from a transfer.

Currently, there is no project that dynamically scales the OS worlds using spatial partitioning and it is believed that it would be a genuine contribution to introduce this feature to OS. According to our knowledge, two attempts have been made to extend the OS framework for scalable worlds. The first project [132, 140] targets a different aspect (scene graph) to spatial partitioning and the second project [141] for load balancing, is no longer maintained. This work is the only current project extending the OS architecture for scalability using spatial partitioning which transfers both the content and players. It targets and resolves the issues in both static and dynamic approaches. Results of a survey conducted by Vervello et al. [219] also motivated the combination of a number of OS regions to start a parent Sim with a bigger world in the present study. According to their observations, a limited number of regions in these worlds are highly populated,

and content creation and destruction happens quite rarely. There are many regions that are never visited or visited by very few people, thus greatly leading to resource under-utilisation. The findings of traditional studies based on spatial partitioning further motivated us to restricting players during a transfer to certain simple activities, rather than freezing them. Players are temporarily moved to an intermediate region called a "transit region" that allows them to move around or keep themselves busy with simple activities until the transfer is complete and they are moved back to the original region.
Most of the relevant academic studies are evaluated through simulation and consider transferring only client connections. To determine the actual strengths and limitations of our work motivated us for the development of a prototype of our work. In order to validate our research, first proof-of-the-concept simulations for scalability and load distribution were conducted, and then a prototype was developed to implement them. The prototype was then tested on both Windows and Linux platforms. Split and merge operations are found to happen rarely and, by developing improved strategies, the time taken by a region transfer was greatly reduced.

None of the existing scalability mechanisms target all the requirements of VWs and consider one or the other aspect of these environments using either the existing games or simulation systems. Indeed, it seems the development of a robust system requires the consideration of a number of parameters based on the inherent properties of VWs. The following goals were set and expected to be achieved through the JoHNUM infrastructure and ARA algorithm:

- to develop a simple and flexible but highly scalable infrastructure,
- to use a localised and decentralised, but dynamic, approach,
- to improve interactive user experience and overall system performance,
- to minimise resource utilisation and communication overhead,
- to solve the resource under-provisioning and over-provisioning issues,
- to minimise complexity, delays, and implementation cost,
- to minimise the time parameters for spatial partitioning,
- to obtain a fair distribution of load, and

- to reduce the number of levels in an RMT and help accommodate conservative applications.

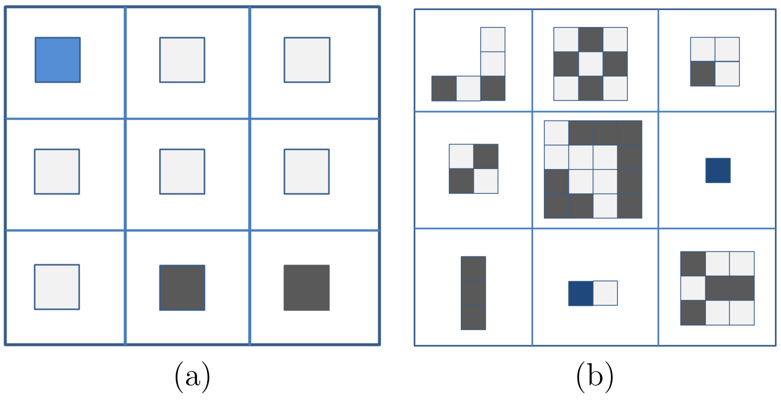


Figure 2.7: Illustration of static partitioning for: (a) SL Grid (normally a single region per Sim but possibly a small fixed number); (b) OS Grid (arbitrary number of regions per server).

We propose a hybrid approach (a dynamic hierarchical Grid) that combines the strengths of both Grid infrastructures and hierarchical dynamic methods. Initially, it uses a static view of grid (used by both SL and OS as shown in Figure 2.7(a) and Figure 2.7(b)) where each Sim (called parent Sim) is assigned a bigger continuous space made of a number of regions in a flat orientation (as discussed for OS in [175]). Based on excessive load, the dynamic provision of additional resources (child Sims) at lower levels share the load with a parent Sim to scale the world as shown in Figure 2.8. It is believed that this will potentially overcome most of the issues. The hierarchy is managed by a parent-child relationship, and the levels in the hierarchy could be greatly reduced by splitting a space into more than two sub-regions and assigning a child to a parent based on the initiation of split. The concept of grid computing could be used to obtain resources on the Internet if an organisation has limited resources (see Appendix A).
Unlike current strategies, a fair distribution of load with a localised and decentralised split and merge operations is believed to improve performance and reduce the potential degradation of interactive user experience by avoiding frequent content and player transfers between servers. It can potentially reduce system complexity and implementation cost, as well as other limitations of local, global and

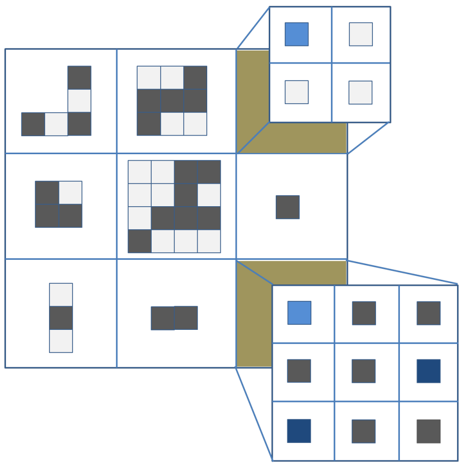


Figure 2.8: Illustration of the proposed hybrid Grid infrastructure with an additional layer of resources.

adaptive dynamic load management strategies. It is capable of solving the issue of single point failure in existing mechanisms. The use of additional resources solves the under-provision of resources while the merging process overcomes the over-provision of resources. Moreover, keeping the infrastructure simple and compromising a little on uniform distribution of load would help to develop a flexible system that scales well. Communication overhead could be reduced by assigning adjacent sub-regions to a single server, and degradation of performance could be avoided by adopting a relaxed merging strategy.

2.2.2 Scaling and Distribution of Load

This section explains how the system developed in this study scales a VW and obtains a fair distribution of load. The system comprises of JoHNUM infrastructure and ARA algorithm.
This work uses a grid model and each grid server (called a parent server in our hybrid model) initially runs a larger part of the whole world. It is assumed that each parent is running a square shaped space and is normally assigned against system capacity. In our implementation model, this is achieved by placing OS regions side by side in multiple rows. Since the basic aim is to dynamically allocate resources against load, JoHNUM infrastructure is applied to cope with increase

and decrease in load, as explained next. It uses the ARA algorithm developed here for load distribution purposes.
Each server (both parent and child) continuously monitors its load against a threshold value, named SplitCapacity (based on a load model presented in section 6.3 of chapter 6). It accepts client connections and initiates the split process to share its load with a newly added server that is selected dynamically from the available pool of resources when it exceeds the SplitCapacity. This process continues until each server is hosting a smaller region that cannot be further divided. All the decisions are taken locally and the framework is quite simple to implement. It splits a space handled by a server into n^2 sub regions with an appropriate value of n for all n > 1. Starting with minimum value 2, the most appropriate value of n is determined by increasing the value of n by 1 each time the prospective regions based on a current value of n are unable to ease the load. However, it combines consecutive regions to obtain larger and continuous regions for assignment by ARA algorithm using aggregation strategies. Sub-regions keep their identity in this case, and system based on increased load re-assigns part of the current load to additional servers until each server is serving a single region. Child servers are always assigned to a parent server that initiated a split, thus greatly minimising the number of levels in an RMT. The aggregation process reduces the number of resources used to simulate the world. It also greatly minimises the cost and complexity of communication and implementation. Merging, on the other hand, integrates the regions, thus sparing some resources to minimise resource under-utilisation. When a child notices that its current load is under a threshold, named MergeCapacity, it determines whether it can merge its load with its parent server. MergeCapacity is assumed to be somewhat smaller than the SplitCapacity to avoid frequent splits after merge operations. However, the integrated larger space is required to be contiguous. The JoHNUM infrastructure and ARA algorithm are examined in chapter 3.

2.3 Existing Synchronisation Mechanisms

Synchronisation or Time Management (TM) is an integral part of PADS systems that ensures the execution of timestamped events in a correct temporal order called a local causality constraint. According to Fujimoto [84], TM algorithms usually treat a simulation as a collection of Logical Processs (LPs) that communicate by exchanging discrete events in time. The goal is to achieve exactly the same results as a sequential computer that precisely ends in time sequence. Each LP maintains a list of events (both internal and external) and in each iteration removes the smallest timestamped event from the list, and processes it. Each LP also maintains a simulation clock that is used for message generation. A simulation clock is normally advanced in response to an event processing.

Initial work for synchronisation is based on conservative approaches [84]. Synchronisation approaches proposed by Bryant [22] and Chandy and Misra [36] are among the initial attempts to ensure local causality; however, they are prone to deadlock. These methods maintain a queue for each incoming link, and when they find the corresponding queue of a link with smallest timestamp empty, they block the process. To resolve this issue, the concept of null messaging is used to advance the simulation clock of an LP. Null messages are sent repeatedly between neighbouring LPs. The timestamp of a null message is the sum of current clock time plus a constraint value known as Lookahead. The Lookahead value is used by a federate (a simulation entity [2]) to determine a minimum value it might be using for a timestamped event in future [80]. It is application dependent and has a dramatic performance effect on a TM algorithm [80]. According to Pan et al. [178], asynchronous TM algorithms with small Lookahead values have the "time creep" problem. The main drawback of the null message algorithm is that it generates an excessive number of null messages, thus introducing longer delays. In a typical synchronous algorithm, the LPs share their Lookahead values and timestamps of smallest events with each other to solve this issue [83]. LPs determine the smallest event and calculate a Lower Bound on Time Stamp (LBTS) allowing events with timestamp less than, or equal to, LBTS to process. The LBTS guarantees that a process will never generate a message with smaller timestamp than this value [82]. It is called Greatest Available Logical Time (GALT) in IEEE 1516 [2] (an IEEE open standard). Since an LBTS value is calculated based on next unprocessed events, it has no time creep problem. The main drawback of synchronous algorithms is that time advancement might be blocked by an LP sending information with a low frequency. Similar synchronous approaches are presented in [160, 206]. Synchronous algorithms also need to cope with transient messages, and the traditional approaches such as message counters and flush queues are used for this purpose [81, 83].

DIS [85] is a standard networked infrastructure developed by the US Department of Defense (DoD) for inter-connecting thousands of synthetic training and simulation environments. The basic aim was to develop an integrated and collaborative environment for group training and activities as a successor of an influential technology called Simulators Networking (SIMNET) [149]. DIS is an easy and lightweight protocol but it does not allow interest management, and load balancing is applicable only to real time simulations [85]. It is restricted to the military domain, and limitations lead to custom modifications and implementations that cannot be re-used. High Level Architecture (HLA) [48] was basically developed as a common interoperability architecture to integrate different classes of simulations. It is generalised and builds upon the results from DIS and similar approaches, such as Aggregate Level Simulation Protocol (ALSP) [69]. HLA TM services provide a mechanism that allows federates (an HLA compliant simulation entity [2]) to send and receive timestamped data and advance their logical time. This allows different approaches to maintain consistency among Sims and, therefore, handles simulations with varied types of ordering and delivery requirements [82]. HLA provides both real time and as-fast-as-possible simulations, and its specifications are flexible enough to accommodate a number of internal TM mechanisms generally used for the applications such as analysis, training, and test and evaluation.
Conservative TM is the main strategy of HLA that maintains timestamp order (TSO) delivery of temporal messages. TM for a federation is realised jointly by a Run Time Infrastructure (RTI) and federates. RTI is a software that coordinates the operation of federates and data exchange during the execution based on HLA - Federate Interface Specifications [1]. RTI is responsible for the TSO delivery and, for that reason, each federate requires an explicit time advance request to it. RTI grants permission if it can guarantee that no messages would be received in the federate's past. To realise this guarantee, an RTI calculates a LBTS value for each federate giving a minimum bound on messages a federate might receive in future. It maintains a TSO queue, and safe intended messages are delivered

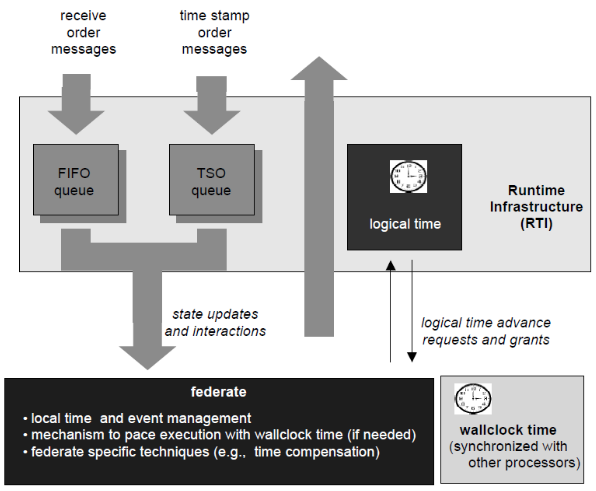

receive order messages
time stamp order messages
FIFO queue
TSO queue
logical time
Runtime Infrastructure (RTI)
state updates and interactions
logical time advance requests and grants
federate
• local time and event management
• mechanism to pace execution with wallclock time (if needed)
• federate specific techniques (e.g., time compensation)
wallclock time (synchronized with other processors)

Figure 2.9: Logical view of Time Management in HLA [82].

to a federate in a non-decreasing order on a time advance request before a grant is issued. For an event driven simulation model, the HLA provides a routine called Next Event Request (NER) that is used to advance a federate time to T. RTI grants a time advance to T, if no TSO messages exist in response of time advance request. Otherwise, it delivers the smallest TSO message destined for the federate, and advances federate time to the timestamp value of the delivered message. Each federate in HLA uses Lookahead value in conjunction with federate time for deadlock avoidance among the federates. The Lookahead value of a federate promises other federates that the earliest timestamp it uses would be greater than, or equal to, its current time plus its Lookahead value. However, the Lookahead value depends on an application and might be changed dynamically during execution. To keep a simulation consistent, the time advance value T must not be greater than the LBTS value at any time. A federate's LBTS value is calculated with the help of time information of those federates that can generate a TSO message (called time regulating federates). The RTI also consider timestamps of messages in RTI and interconnection network to compute an LBTS value.
The concept of LBTS used for Parallel Discrete Event Simulation (PDES) in HLA 1.3 (a US DoD standard) [151] is replaced with a terminology of Greatest Available Logical Time (GALT) in IEEE 1516 (an IEEE open standard) [2]. Liu et al. [138], argue that TM in HLA is a crucial factor that restricts the size of distributed

simulation and, therefore, RTI is required to efficiently handle large number of federates. They also argue that TM in HLA using GALT is more complex than TM in PDES using LBTS in two major aspects. Firstly, a federate time advance in HLA is under the control of an RTI, but a PDES entity advances its time without an underlying infrastructure. Secondly, PDES supports modular and hierarchical structures, but HLA lacks this feature. Our consistency work targets complex hierarchical models and uses the concept of LBTS for PDES in the synchronisation approach presented in chapter 4. Liu et al. [138] presented an efficient GALT algorithm and developed a prototype over a cluster system that successfully manages thousands of federates. However, it is complex and not tested for very large scale distributed federates. Furthermore, it manages all the federates using a centralised approach. Pan et al. [178] presented a hybrid HLA TM algorithm based on both conditional and unconditional information to resolve the drawbacks of both synchronous and asynchronous algorithms. According to them, synchronous algorithms use conditional information while asynchronous algorithms use unconditional information. Both of them have limitations, and neither of them has the ability to manage varied types of federation scenarios. They incorporated three algorithms into an RTI and effectively achieve the combined advantages of both algorithms.

HLA has shown itself to be a great success for military applications; however, it has a number of limitations regarding interoperability, scalability, and complexity. It does not provide load balancing and is poorly scalable, providing only syntactic interoperability. It is complex, difficult to learn, and difficult to adopt and use [48]. The basic HLA standard does not support multi-level or hierarchical federations but implements all federates at one level as a single federation. It includes no means of inter-federation communication, and the conversion of a complex hierarchical structure into a flat one introduces several issues regarding data exchange, security, and re-usability [127]. It does not deal with the issue of information hiding, and the flat structure is not adequate to model complex system with hierarchies. Some federates might not require certain information, and due to subscription of federates for common data, it is difficult to distinguish among different copies of data [32]. Extra checks are needed to validate the source of data that greatly degrades the performance of a federation. Renaming data can easily resolve this situation but requires modifications in code that damage re-usability. Kim and Kim [127] argue that these issues can be easily resolved if modular and hierarchical modeling methods are adopted. The Literature has a wide range of both flat and hierarchical HLA extensions to cope with these issues. However, in this work the discussion focuses on hierarchical solutions that are used to manage complex models.

According to Zhang et al. [244], in a traditional HLA TM as shown in Figure 2.10(a), an RTI computes the LBTS values for all the participating federates by collecting their logical times. It explicitly grants permission to each federate for its time advance. Being a centralised component, RTI might become the bottleneck of the system in terms of both computational complexity (with an increase in number of federates) and overhead of messages. Therefore, RTI based federations are not scalable and could suffer from performance degradation. Zhang et al. [244] presented a two-level TM mechanism for HLA based DVEs to overcome these issues, and it is presented in Figure 2.10(b). It divides the federates in to several Federate Groups (FGs) where each FG has an additional component called FGTimeManager that is responsible for the time advance of its federates. RTI provides communication between FGTimeManagers [244]. It greatly decreases the computation load and intensive communication in an RTI, but it introduces an additional level to TM that further increases complexity. It still depends on a centralised RTI to obtain an integrated simulation environment.

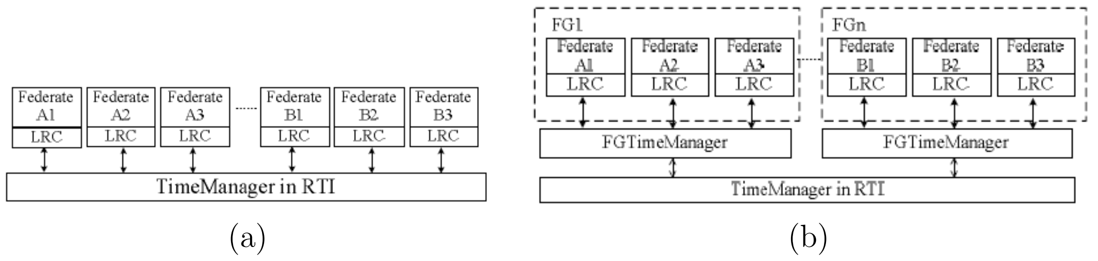


Figure 2.10: Illustration and comparison of [244] (a) traditional Time Management, and (b) two level Time Management;

According to Myjak et al. [155], a federation community is a group of federations and RTIs working together to achieve a common goal. A hierarchical federation is a special type of a federation community in which federations are organised in a hierarchy, and a federation acts as a federate in an upper-level federation. The Literature shows a number of attempts to provide interoperability between federations to form federation communities, including Federation Gateway, Proxy/Bridge Federate, RTI Broker, and RTI-to-RTI Protocol [32, 156]. The first two approaches provide solutions at the application level while the other two may require system level modifications. Only the first two approaches are examined here.

A **Federation Gateway**, as shown in Figure 2.11(a) is a separate process interconnecting two or more federates of different federations and performs translation among federations. It can be used for both interoperability among HLA compliant federations, or between an HLA federation with a legacy simulation protocol such as DIS and ALSP [32]. It has the capability to provide state information from one federation to another and filter out sensitive information, thus performing information hiding [68].

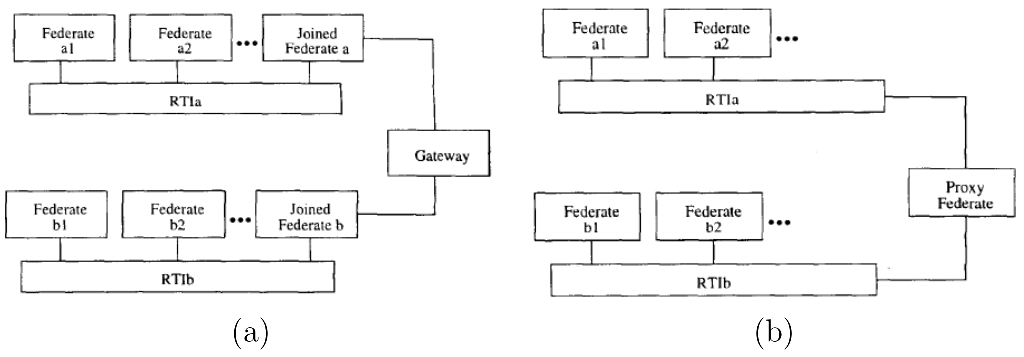


Figure 2.11: Illustration of [32, 156]: (a) Gateway Architecture; (b) Proxy Architecture.

A **Proxy Federate**, as shown in Figure 2.11(b), is a federate that is associated with more than one federation simultaneously. It requires multiple interfaces to communicate with different RTIs. It performs some similar functions to a federate gateway such as data transformation. However, it does not provide information hiding and cannot be trusted by the federations.

Cai et al. [32] presented a hybrid approach called the hierarchical federation ar-

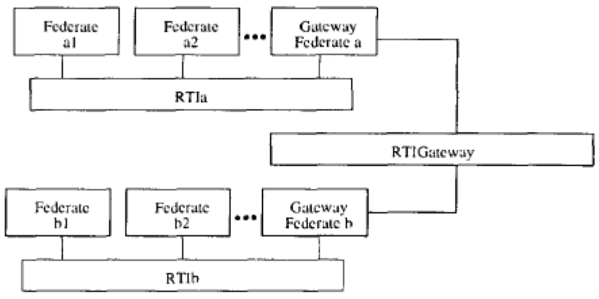


Figure 2.12: Hierarchical federation architecture [32].

chitecture by combining both gateway and proxy mechanisms and it is presented in Figure 2.12. Federates in a federation interact with each other by a dedicated RTI session. However, they interact with federates of other federations through gateway federates. The gateway federates constitute a gateway federation where their interaction is managed by a dedicated RTI session which filters confidential information and improves security and interoperability [31, 32]. However, it requires additional interfaces and the structure is very complex. It has potential performance issues because it places no restrictions on levels in a hierarchy.

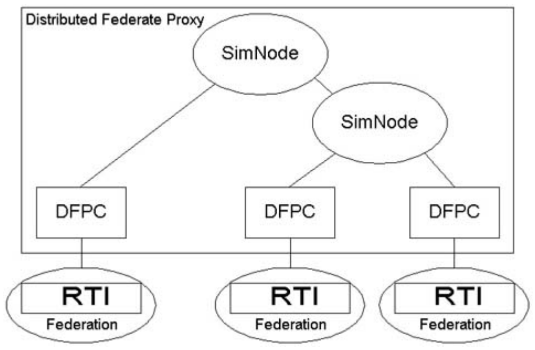


Figure 2.13: Distributed Federate Proxy architecture for hierarchical federation communities [41].

According to Cramp et al. [41], a Federate Proxy (FP) is the simplest architecture for inter-federation communication. However, it might not be an ideal data filter for geographically distributed simulation, being a single local process. Furthermore, it handles a single federation with a flat orientation. To minimise the traversal of potentially large distances, Magee et al. [144] presented the concept of Distributed FP (DFP). Cramp and Oudshoorn [42] further extended it for the hierarchical federation communities. They proposed splitting a FP into a number of different components where each is assigned and processed local to a federation. These components are called DFP Components (DFPCs) which are linked together with the help of tree nodes called SimNodes, thus forming a hierarchical federation community. Each SimNode represents the root of a sub-federation community and manages communication between its child nodes. It also communicates with its parent node if one exists. This architecture is presented in Figure 2.13. However, hierarchical architectures impose additional LBTS constraints on TM services to determine the correct order of execution for temporal aspects of a system [41]. These constraints are applied to system components that are federates, DFPCs, RTIs, SimNodes, and a root SimNode (see Figure 2.13). Cramp et al. [41] concluded that the LBTS value of the root SimNode is the earliest possible timestamp assigned to a message generated by any federate in the entire hierarchical community. This implies that time advance decisions are made by the ultimate root SimNode. We believe that this mechanism is too complex and might introduce longer delays, making it unsuitable for real time systems in that it might block the whole system for a time advance. The whole simulation might be stopped if the root SimNode crashes.

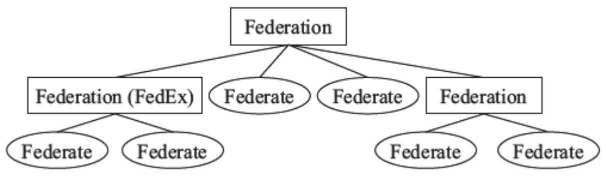


Figure 2.14: Components of the extended HLA architecture [126].

Kim and Kim [126] argue that hierarchical models are essential to simulate large complex systems, although the methods described above are temporary solutions. These require additional interfaces that are not part of the RTI specification and, therefore, it is difficult to achieve interoperability among the RTI systems developed by different companies. An interoperability protocol based on an open messaging is under development but its target implementation is flat. According to Kim and Kim [126], to improve overall performance of an RTI, hierarchical federations need to be supported by an RTI itself. They presented a hierarchical extension to HLA to incorporate hierarchical federations (see Figure 2.14) among federations based on an inspiration from the formalism of PDES [243]. The PDES characterises hierarchical and modular specifications of discrete event systems. The same authors, developed extended HLA services called Federation Execution (FedEx) processes to manage hierarchical federations [126]. Each federate communicates with its parent FedEx process and a FedEx process communicates with both a FedEx (which might be either a child or a parent) and the federate processes as shown in Figure 2.14. A FedEx process acts as a federate in an upper-level federation and is called a representative federate. The current time of a representative federate (current LBTS) is the minimum of federate time plus lookahead values among the participating federates. Therefore, a message generated by any federate would have a timestamp value greater than, or equal to, the federation time. For the realisation of a hierarchical HLA, they proposed two possible implementations of hierarchical RTI: FedEx Processes, and fully distributed federations. FedEx processes handle federation-related services that include synchronisation of known federates and data exchange. The functionality and relation of federates is simple, but it can potentially introduce longer delays because no limits are set on the depth of FedEx processes in a hierarchy. The second approach distributes the RTI functionality among federates and avoids FedEx processes and extra levels in a hierarchy. However, it complicates the design of RTI libraries and is difficult to manage [126, 127].

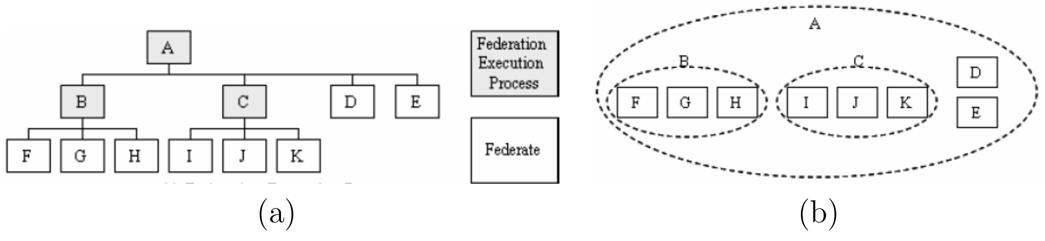


Figure 2.15: Illustration of implementation for [127] (a) Federation Execution Processes, and (b) fully distributed federates.

It is believed that by using the inherent properties of VWs, the second implementation option could be easily utilised, with a restriction on the number of interacting federates. This eliminates the management issues in existing P2P systems. Motivating factors, together with mechanisms to achieve the goals set for our synchronisation approach, are presented next.

2.3.1 Motivation and goals of the consistency approach

This work also studied and investigated the current TM systems that are used to get a consistent view of the overall virtual space.
PADS systems basically developed for games and simulation environments are scalable and provide better interactive user experience. They have a successful history, but they put limits on the application domain and compromise on consistency to achieve improved performance. Therefore, they face issues with applications of a conservative nature such as e-business applications. The majority of existing synchronisation mechanisms are implemented with either a centralised or a distributed approach. They perform well with small and medium scale environments, but are not suitable for large scale continuous spaces such as VWs. The existing hierarchical mechanisms are flexible but complex, and introduce longer delays because of dependencies among their components when they are used to get a global consistent space using conservative approaches. Centralised approaches to manage hierarchical structures block the activities of the whole space during certain operations such as a time advance. The complexity and, thus, the implementation cost is too high for hierarchical structures.

Current VWs rely on conventional web techniques for e-business applications on the Internet. The limits on the application domain greatly weakens the claim of developers of these worlds that they are evolving towards a future 3D web. The nature of VWs is different to games and simulation environments, and both latency and consistency are of prime concern. An individual or an activity is only affected by the events in their neighbourhood. According to our knowledge, there is no specific synchronisation method dealing with VWs of this nature. To cope with the issues with existing mechanisms and to incorporate conservative appli-

cations in VWs, we developed a decentralised consistency management approach in this work. It is important to mention that this consistency approach applies to hierarchical models based on the JoHNUM split strategies [61].
The following goals are set and expected to be achieved through our decentralised synchronisation approach:

- to use a decentralised and local control,
- to use a constrained communication model,
- to remove intermediate points, and thus reduce delays and system complexity,
- to reduce implementation cost and improve performance,
- to avoid blockage of the overall system, and
- to incorporate conservative applications in VWs.

It is believed that a simple but flexible synchronisation approach can be developed by adopting a constrained P2P communication model (based on inherent properties of VWs) to maintain a consistent view of dynamic but constrained hierarchical models [61]. By adopting decentralised control with a limited number of interacting servers, a system potentially outperforms both centralised and distributed systems. A server with additional functionality can take purely local decisions (such as a time advance) in direct consultation with the servers that host adjacent regions. It has the potential to maintain the traditional constraints and guarantee that all events are processed in their temporal order. This potentially solves the issue of system blockage, thus allowing different groups of people to carry out their activities. It greatly reduces communication overhead, complexity, and delays by avoiding the intermediate points (hops) compared with traditional distributed and hierarchical mechanisms based on conservative approaches. It is flexible with reduced implementation cost and is very likely to improve performance.

2.3.2 How Consistent Virtual Worlds are achieved

This section shows how our system achieves a consistent state among players based on time information of only immediate neighbours that might possibly be handled by a hierarchical infrastructure at different levels in a hierarchy.

This decentralised synchronisation approach uses the concepts of HLA for conservative discrete event simulation and exploit them to hierarchical models in a P2P fashion with a constrained communication model. The logical time, LBTS and Lookahead values work in a traditional way, but functionality of the RTI is distributed among the federates and each federate has the capability to interact with the federates surrounding it. Each federate processes the events against its LBTS value (thus guaranteeing processing of safe events) which is the minimum among the time information (Current LBTS + Lookahead value) of adjacent federates. Timestamped messages are delivered in an order that guarantees that messages will never arrive in a federate's past. This synchronisation approach and proof-of-the-concept simulation are presented in detail in chapter 4.

2.4 Open Source VW Development Frameworks

Open sourcing has decreased the development time and cost remarkably by reusing existing software modules and components to develop new solutions. It has especially provided a useful way for the research communities to implement and test their novel ideas. This section presents a number of parameters and their corresponding filters that we used during a survey to select the most suitable framework for the implementation of our work. A total of twenty-two frameworks were studied: Second Life (SL) [137, 189, 201], Croquet [43, 44], Maverik [99, 147], Arianne [109, 234], Beyond 2 [5, 110], BZ Flag [228, 235], Quack II [104, 116, 183], Genecys [88, 89], Massiv [145, 146], Crossfire [229, 236], FreeTribes [112], netPanzer [158], Irrlicht Engine [113, 237], WarZone 2100 [117, 239], Janthus [115], Isotope [114], Argentum Online [108, 220], Diamonin [111], WorldForge [231, 240], OpenArena [230, 238], OpenCobalt [169, 170] and OpenSimulator (OS) [172, 177]. Details of these frameworks are beyond the scope of this work and are, therefore, not provided.

Parameters that were used to evaluate these frameworks against our requirements are: Architecture, Concurrency, Status of the Project, Documentation and Help, Operating System, Development Language(s), Level of open sourcing, the efforts required to implement our research, and Persistence. An iterative evaluation criterion is applied to these parameters in the order of presentation with their corre-

Parameter Name	Filter	Iteration Number
Architecture	Allow Client/Server and CMS infrastructures (Grid)	1
Concurrency	Allow Multiuser/Massive Multiplayer systems	2
Project Status	Allow Complete and/or projects under further development	3
Documentation and Help	Allow those having forums, IRCs, Documentation and a Wiki	4
Operating System	Windows and Linux	5
Development Language(s)	Allow any combination of C/C++, C#, Java, and Python	6
Level of Open Sourcing	Allow complete framework and games/demos	7
Efforts Required	Allow those that require minimum or average efforts	8
Persistence	Allow only persistent infrastructures	9

Table 2.1: Parameters and their corresponding filters

sponding filters , as shown in Table 2.1. This process rejected even state-of-the-art projects if they were unable to fulfill one criterion or the other. For simplicity, frameworks eliminated in each step are presented in Table 2.2
The filter for architecture (iteration 1) allowed only those projects that were based on CS and CMS architectures. It rejected frameworks based on P2P architectures, and those that were not applicable to any architecture. For example, Croquet and Isotope are rejected for being P2P environments, and Maverik and Irrlicht Engine for their specialised natures. P2P systems are not suitable for real time graphics applications and the specialised softwares cover very narrow domains. Therefore, a developer needs to design both a server and a client in addition to a communication infrastructure. **The second iteration** (iteration 2) eliminated none of the remaining environments as all the remaining infrastructures are multiplayer in nature. Maverik is the only single user environment but it is already removed in the first iteration. However, some environments, such as WarZone, provide both single and multi-user environments. Though the development of an environment using open source components minimises both time and cost, it is convenient and easy to work with a project if it is alive and supported by a dedicated community of developers. Therefore, **the next two iterations** (iterations 3 and 4) considered the projects having their source code available for the basic components of both the infrastructure and games. Furthermore, environments supported by a number of ways such as Documentation, Forums, IRCs, Mailing Lists and a Wiki are the most favourable choices. Janthus and Genecys

were removed for providing very little documentation, and Massiv for being a dead project. Moreover, these projects offer poor support for the developers. **The operating system filter** (iteration 5) allowed those projects that run over Linux or Windows. Linux is best for development and Windows for easy and friendly interfacing. The remaining projects all run over both or at least one of these operating systems and, therefore, none is removed in this iteration. **The criteria for development language(s)** (iteration 6) considered the integration power, ease, and available expertise and help in our organisation. This step permitted those projects that are solely or partially based on any set of C, C++, C#, Java, and Python. **Level of open sourcing filter** (iteration 7) discarded SL which is state-of-the-art in VWs. It is a commercial infrastructure and most relevant to our intentions; however, only the viewer of the infrastructure is open source. **The required efforts parameter** (iteration 8) filtered out OpenArena for the huge effort required as it provides very basic functionalities, and one needs to design both server and client modules. **Persistence** (iteration 9) discarded most of the game development environments because they are normally not persistent and follow pre-defined rules from the start each time. Some commercial games, such as EverQuest and World of WarCraft provide persistence, but still follow pre-defined narrations. Furthermore, their source codes are not available.

The filtering process returned three frameworks: **Arianne**, **WorldForge**, and **OS**. This process discarded most well-known and relevant frameworks due to their limitations. Isotope implements a similar concept presented by SL extension. However, it is no longer maintained and only a little documentation is available for help. OS is the most relevant framework to this study that implements the extended architecture of SL, and therefore provides the basis required for the implementation of our work. Therefore, it was selected for the prototype development of our work.

2.5 Conclusions and Future Work

This chapter presented a detailed analysis of the existing scalable and consistent VW development strategies. It examined the underlying technical infrastructures

Criteria	Eliminated Frameworks
Architecture	Croquet, Isotope, Maverik, Irrlicht Engine
Concurrency	None
Project Status	Massiv
Documentation and Help	Janthus, Genecys
Operating System	None
Development Language(s)	None
Level of Open Sourcing	SL
Efforts Required	OpenArena
Persistence	Beyond 2, BZ Flag, Quak II, Crossfire, FreeTribes, netPanzer WarZone 2100, Argentum Online, Diamonin, OpenCobalt

Table 2.2: Summary of filtering process showing eliminated frameworks

that are used to host VWs. It determined that Grid infrastructures are best for computationally intensive jobs, but that P2P infrastructures are more scalable. Hierarchical infrastructures are well-suited to model complex environments but they have no restrictions on the levels in an RMT. It makes it difficult for these worlds to perform better in terms of scalability, especially when they are used to manage conservative applications. Existing mechanisms for both scalability and consistency have performance issues when used for the VWs because they are primarily developed for game and simulation environments. This chapter also identified the issues in existing systems and presented the motivation for the work undertaken in this thesis. It also outlined the way to achieve the goals set for this study. It presented a short comparison of a number of VW development frameworks and identified OS as the most appropriate framework for the prototype development of our work.

Chapter 3

Scalable Virtual Worlds

In this chapter, the Joint Hierarchical Nodes Based User Management (JoHNUM) infrastructure for scalable VWs is presented. It describes the proposed partitioning method, a number of assignment strategies for load distribution, and a merging mechanism. Simulation Results in MATLAB compare it with a dynamic game middleware called Matrix [13] for evaluation purposes. It is demonstrated that it reduces the number of levels in a Resource Management Tree (RMT), and decreases the number of times a user is interrupted during a session.
This chapter further discusses the assignment phase of the JoHNUM infrastructure in more detail and presents the Aggregate Region Assignment (ARA) algorithm for achieving a fair distribution of load. The basic aim is to minimise resource under-utilisation, and reduce communication and implementation costs by combining individual regions into larger contiguous areas. Simulation results demonstrate that it works well with small, medium, and large scale worlds. An abstract model is used to identify the communication and implementation overhead reduction through this algorithm. This chapter is based on our published work [60, 61, 64] on this topic.

3.1 The JoHNUM Infrastructure

This section describes the components of the JoHNUM infrastructure and the way it achieves its goals. It also provides a proof-of-concept simulation to evaluate and compare it with Matrix.

3.1.1 Introduction

This work assumes a geographically distributed workspace with a unified user view that is processed by a Grid infrastructure. Initially, each geographic location starts with a server that simulates approximately the same content as other locations. In addition, each server handles no more than a number of users described as Maximum Server Capacity (MSC). Furthermore, the world is approximately square-shaped and a regular square pattern is followed for splitting the overloaded regions. Each server continuously monitors the workload for the assigned region, and splits it into a number of smaller regions in the case of excessive load. Local decisions are made by using a split factor, named the Region Split Factor (RSF). The number of smaller regions are determined by RSF^2 for $RSF > 1$, while the boundaries are calculated by considering the height and width of the VW against the RSF. The proposed algorithm first determines the RSF value that eases the load and then splits the region accordingly. It starts with an initial RSF value of 2 giving four regions, but the value is incremented on the basis of players' distribution yielding more smaller regions as explained later with the help of Figures 3.3 and 3.4. The proposed partitioning algorithm is illustrated in Figure 3.1 showing that Server 1 and Server 4 divide their corresponding regions into four while Server 2 divides its assigned space into nine smaller regions. However, it could divide the world into a larger number of smaller regions (such as sixteen or twenty-five) but against a boundary condition to ease the load in case the crowd assembles at a very small space. The load in Server 3 is normal and, hence, no split occurs in this case.

The JoHNUM infrastructure comprises three components: Partitioning, Assignment (Load Distribution), and Merging. Partitioning is hierarchical in nature and is triggered by a regional server as its capacity exceeds the MSC. Figure 3.2 shows a number of possible splits into grids of 2×2 and 3×3 sub-regions in a two-level hierarchy for different geographic servers. A region representing an un-partitioned but varied size of space is divided during a split operation if it is not the ultimate space that cannot be further partitioned. The concept of aggregate assignment

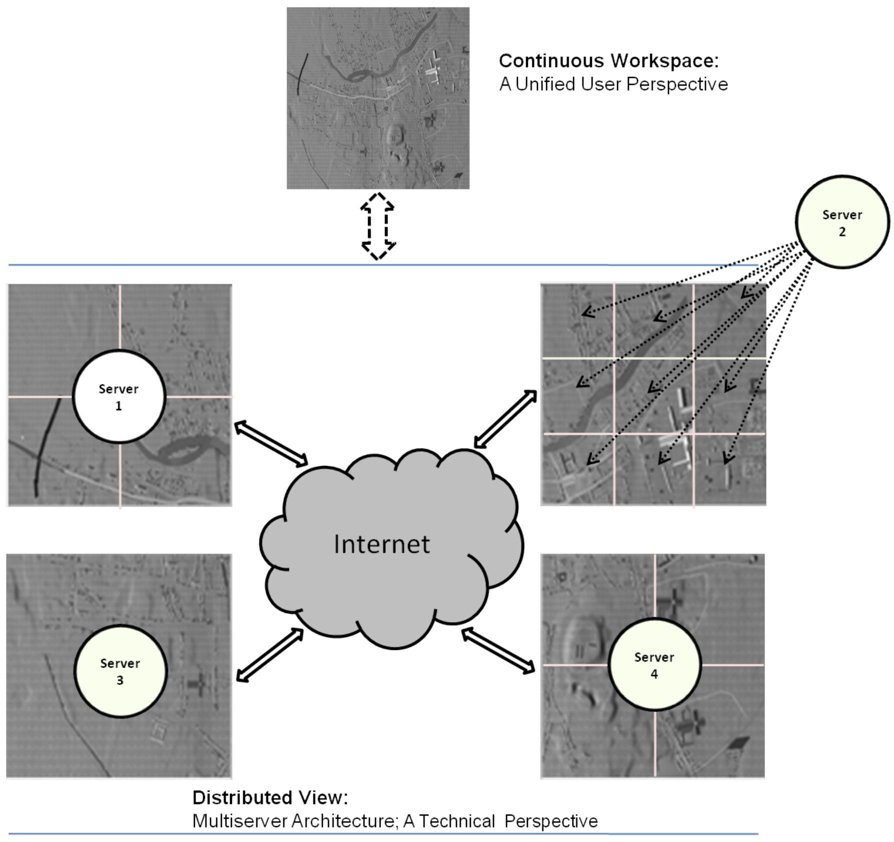


Figure 3.1: Abstract view of JoHNUM partitioning algorithm [61].

is used to minimise resource utilisation and communication between the servers. Moreover, it tries to achieve a fair distribution of load between the two servers by combining adjacent regions and avoiding the diagonal ones. A regional server that triggers a split and assignment becomes the parent of the new server. The partitioning algorithm developed in this study is simple, and better performance is achieved by using intelligent assignment strategies. Partitioning and assignment are two different concepts and two terms are introduced to highlight their applicability. The aggregate assignment is termed **"provisional assignment"**, which maintains the identity of individual smaller regions, and at a later stage, triggered by increasing load, the aggregates are re-assigned until each server is handling a single smaller region termed **"permanent assignment"**. At this point, the smaller regions are further subdivided into yet smaller regions unless these are too small for subdivision. We describe it as a boundary condition. The terms split and partitioning are used interchangeably in this work, and mean dividing a region into a set of smaller regions.

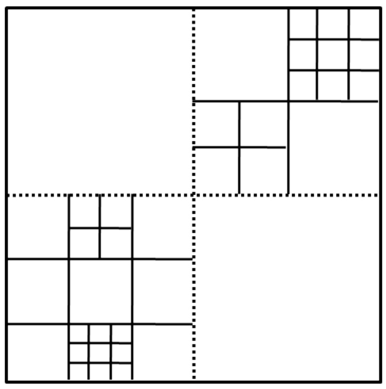


Figure 3.2: Illustrating two-level splits with various combinations at different geographic locations.

3.1.2 JoHNUM Partitioning

The basic JoHNUM partitioning algorithm works as follows. Each server continuously monitors total players against the MSC, and applies a greedy approach that divides the space into RSF^2 smaller regions in case of excessive load. Theoretically, this eases the load but it is not always practically possible due to the unpredictable nature of the users; for example, if we assume that the cases shown in Figure 3.3 consider the MSC of nine players and the server triggers a split as the tenth player enters the region. Moreover, the solid lines show actual splits and the dashed lines represent prospective regions only. The server divides the world into four regions if the players' distribution is uniform, as shown in Figure 3.3(a). However, Figure 3.3(b) highlights a hotspot scenario that fails basic JoHNUM partitioning in order to ease the load.

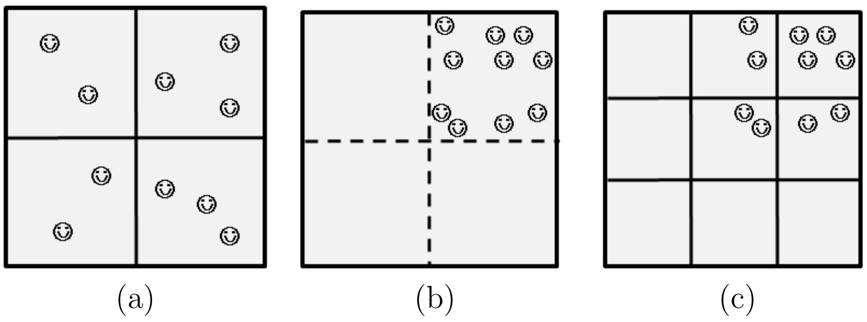


Figure 3.3: Illustration of JoHNUM Partitioning with uniform and hotspot scenarios: (a) a split of uniform scenario into 4 smaller regions; (b) highlighting a hotspot that fails basic JoHNUM partitioning; (c) a split using Players Considered JoHNUM Strategy that splits a region into 9 smaller regions.

Players Considered JoHNUM partitioning solves this issue by considering player density in each prospective region before splitting. It increments the RSF until the player density in each prospective region is below the MSC. The region is split once the final RSF value is determined, which results in more smaller regions. Figure 3.3(c) illustrates the split of hotspot scenario into nine regions instead of four, as shown in Figure 3.3(b). In the remainder of this work, JoHNUM partitioning is referred to as Players Considered JoHNUM partitioning. Figure 3.4 shows how the system splits the space into sixteen smaller regions when it failed to ease the load, by dividing it into four and nine sub-regions for the same example.

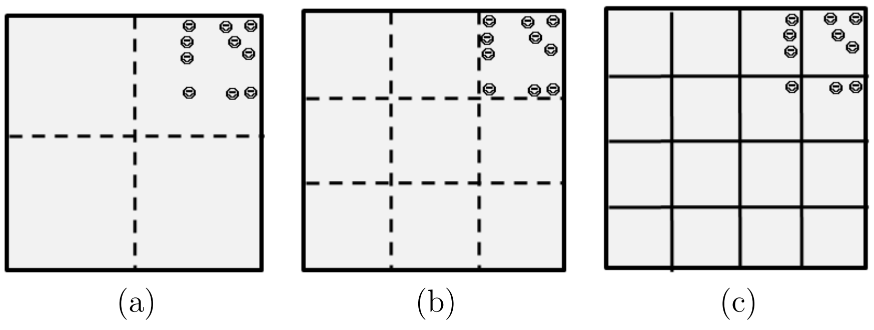


Figure 3.4: Illustration of JoHNUM Partitioning in to 16 smaller regions: (a) highlighting a hotspot that fails to ease the load with RSF value 2; (b) highlighting a hotspot that fails to ease the load with RSF value 3; (c) highlighting a split into 16 smaller regions.

3.1.3 JoHNUM Assignment/Load Distribution

JoHNUM Assignment comprises two basic strategies based on individual and aggregate assignments. Each of these has its benefits and limitations. To eliminate the limitations in basic strategies, a third strategy is devised which achieves better performance, as discussed later. Basic assignment strategies are described as follows:

JoHNUM Assignment Strategy 1 (JAS1): assigns each smaller region to a different server. It selects n-1 servers and assigns n-1 smaller regions to them while keeping the nth with itself.
JoHNUM Assignment Strategy 2 (JAS2): applies aggregation at each step in the assignment and tries to balance the load as much as possible. It first aggregates smaller regions into two groups and then selects another server for sharing its load with it in each iteration.

JAS1 is the most simple and flexible strategy, but it yields significant load imbalance. It might assign regions to servers having no players at all and greatly increases resource utilisation and communication overhead. JAS1 obtains lowest-level RMT, which we believe helps to improve consistency and interactive user experience. JAS2 applies aggregation at each stage during assignment and re-

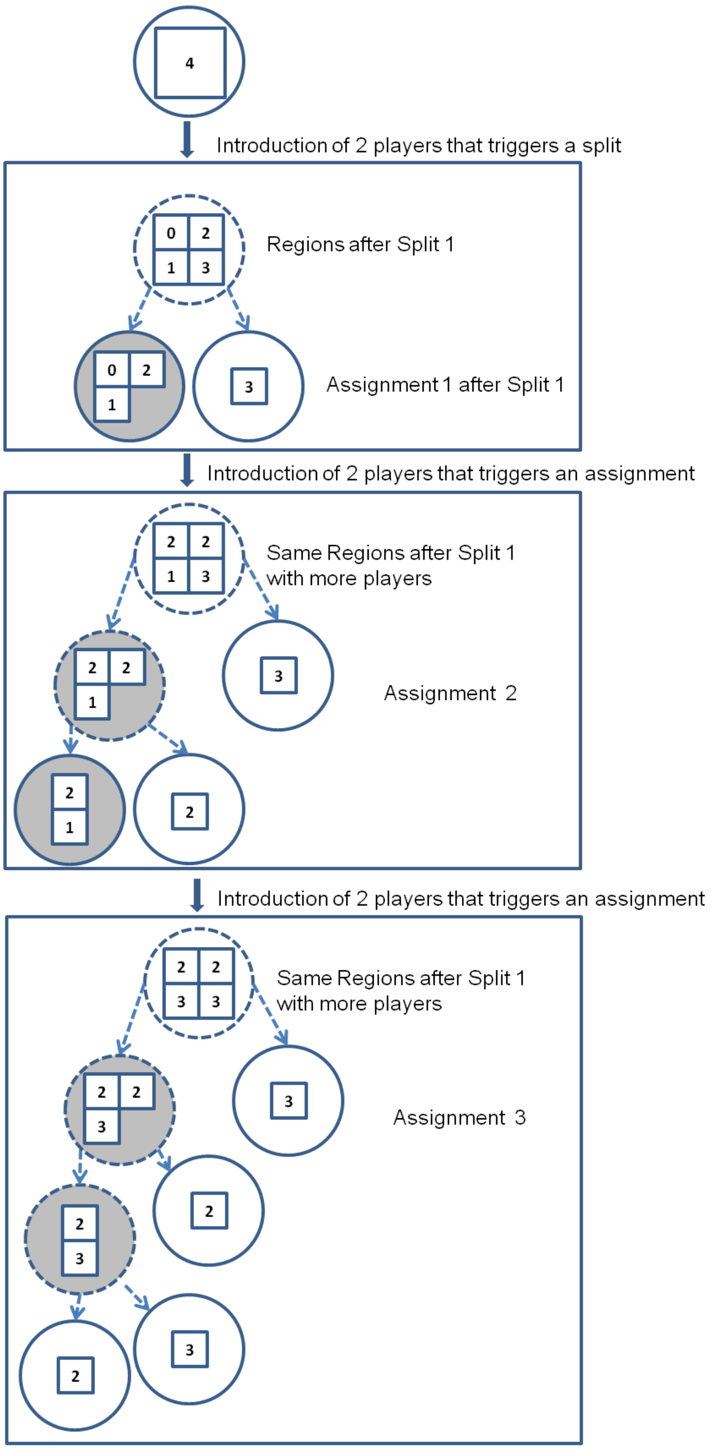

4
Introduction of 2 players that triggers a split
0 2
1 3
Regions after Split 1
0 2
1
3
Assignment 1 after Split 1
Introduction of 2 players that triggers an assignment
2 2
1 3
Same Regions after Split 1
with more players
2 2
1
3
Assignment 2
2
1
2
Introduction of 2 players that triggers an assignment
2 2
3 3
Same Regions after Split 1
with more players
2 2
3
3
Assignment 3
2
3
2
2
3

Figure 3.5: Illustration of JoHNUM Assignment Strategy 2 (JAS2): It can be seen that the levels in each assignment step are increased with the introduction of additional players (3 levels in this case).

quires little effort to re-assign provisionally assigned regions to a new child server in case of further excessive load. It achieves better resource utilisation and reduces communication overhead. However, it increases the number of levels in the RMT, as illustrated in Figure 3.5 with a simple example that assumes the MSC of four players. Here, it is assumed that white and grey coloured solid circles represent permanent and provisional assignments, while white and grey coloured dotted circles characterise a unified view of the world and intermediate logical processes for the assignments, correspondingly. A region is represented by a square in the diagram. Aggregate assignment tries to balance the load as much as possible between the two servers at any point while maintaining contiguous and regular areas for allocation. This issue is investigated in detail further in section 3.2 by presenting a number of aggregation strategies and an aggregation algorithm. JAS2 selects a new server in each assignment phase and makes it a child of the assignment server, thus increasing the RMT levels. RMT levels are greatly reduced if we modify server management by making a new server the child of the server that initiates partitioning instead of an assignment. This strategy (named JAS3) achieves better results and is explained in Figure 3.6 with the same example and assumptions used for JAS2. However, it has suppressed the details of intermediate logical processes for the assignments. JAS3 is similar to JAS2 in partitioning but only different in the assignment phase.

3.1.4 JoHNUM Merging

The JoHNUM merging algorithm implements the reverse process to partitioning. A server triggers merging when it observes a decrease in the assigned players' density. Both parent and child can initiate the process. When a server experiences a decline in its assigned capacity, it asks child nodes for their loads and re-computes the load. It revokes some load and releases extra resources to minimise resource under-utilisation against a MergeCapacity. MergeCapacity is made much smaller than the MSC to avoid immediate splits. The child server asks its children for their load, if any, and the re-computation is performed in a bottom-up fashion. In case a child detects reduction in its assigned capacity, it asks its parent server for the load and determines the cumulative load. It returns its load if the cumulative

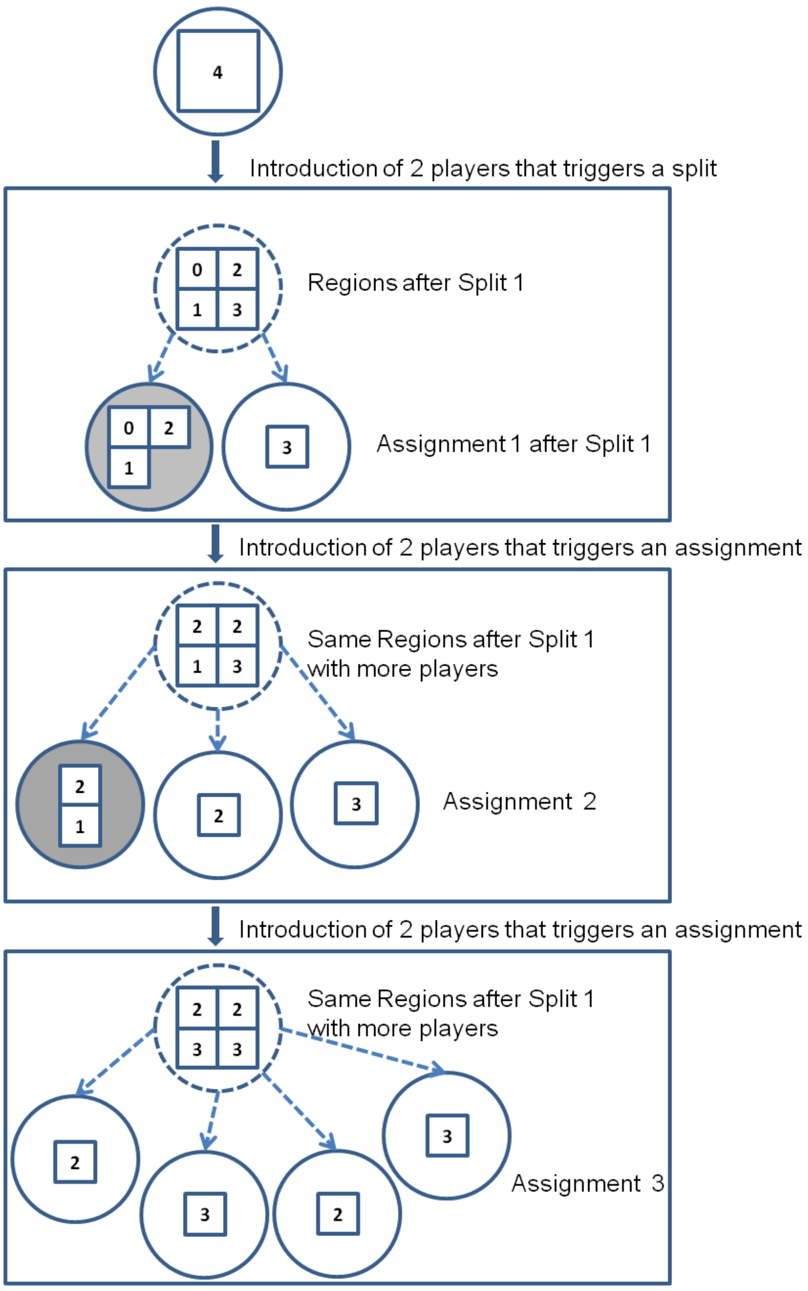

4
Introduction of 2 players that triggers a split
0 2 1 3
Regions after Split 1
0 2 1
3
Assignment 1 after Split 1
Introduction of 2 players that triggers an assignment
2 2 1 3
Same Regions after Split 1 with more players
2 1
2
3
Assignment 2
Introduction of 2 players that triggers an assignment
2 2 3 3
Same Regions after Split 1 with more players
2
3
2
3
Assignment 3

Figure 3.6: Illustration of JoHNUM Assignment Strategy 3 (JAS3): It should be noted that the levels in this strategy remains the same (1 level) for this particular example in comparison with JAS2.

load is less than the MergeCapacity.

3.1.5 Simulation Setup and Assumptions

JoHNUM infrastructure is evaluated through simulation in MATLAB. A world is represented by an n×n matrix. Experiments have been performed with worlds of different dimensions, but this work presents and discusses the results of only a few of those due to their similar outcomes. Table 3.1 summarises the parametric values considered for each experiment including dimension, player distribution, number of players initiating partitioning, and the MSC. Instead of introducing additional players at lower levels, the MSC of each server is decreased to trigger the algorithm. MSC was reduced four times for each experiment and the reduction pattern is provided in the far right column of Table 3.1 from left to right. L0 represents the initial value of MSC. We assumed that the last reduction (represented as L4) reaches the boundary condition and stops further partitioning. It is also assumed that a server triggers the partitioning algorithm as the number of players exceeds the MSC.

Experiment Number	Dimension	Player Distribution	Players that initiate Partitioning	MSC (L0)	Reduction (L2R): L1:L2:L3:L4
1	8*8	Uniform	33	30	20:10:8:4
2	8*8	Uniform	31	30	20:10:8:4
3	8*8	Hotspot	41	40	30:15:8:4
4	18*18	Hotspot	159	150	100:75:50:25

Table 3.1: Experimental assumptions for the experiments

Being a dynamic infrastructure, JoHNUM is compared with a dynamic game middleware, called Matrix, that has demonstrated better performance over previously published static and dynamic mechanisms. The following two strategies of Matrix are used for comparison with JoHNUM strategies:
Matrix Strategy 1 (MS1): assigns minimum load to the newly selected server and the existing server maintains most of the load.
Matrix Strategy 2 (MS2): uniformly distributes the load and shares exactly half of the load with a newly selected server.

The experimental work used a number of metrics for evaluation purposes, which are: Number of Regions, RMT Levels, Resource Utilisation, Degradation of Interactive User Experience, and Communication Overhead.

Number of Regions: provides the total number of regions after splits at different levels for the complete span of an experiment.

RMT Levels: represents the number of levels in an RMT, determined by considering the longest of the shortest paths from root to the leaf nodes.

Resource Utilisation: determines the maximum number of resources utilised to simulate the whole world.

Degradation of Interactive User Experience: demonstrates the average number of times a user suffers from the splits.

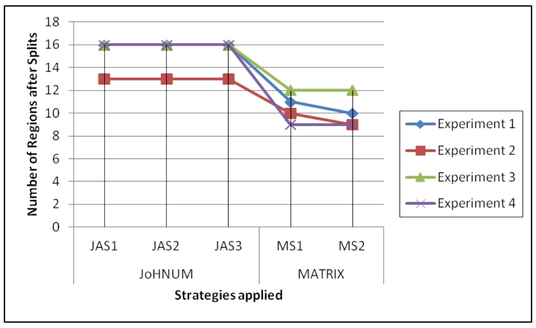
Communication Overhead: outlines increase in communication (over the network) between servers in response to splits.

Exp No.	Algorithm	Strategy	Regions	RMT Levels	Resource Utilisation	Degradation of Interactive User Experience
1	JoHNUM	JAS1	16	2	16	2
		JAS2	16	3	9	2
		JAS3	16	2	10	2
	Matrix	MS1	11	5	11	4
		MS2	10	4	10	4
2	JoHNUM	JAS1	13	2	13	2
		JAS2	13	4	10	2
		JAS3	13	2	10	2
	Matrix	MS1	10	5	10	4
		MS2	9	4	9	4
3	JoHNUM	JAS1	16	2	16	2
		JAS2	16	4	12	2
		JAS3	16	2	12	2
	Matrix	MS1	12	5	12	4
		MS2	12	4	12	4
4	JoHNUM	JAS1	16	2	16	2
		JAS2	16	3	8	2
		JAS3	16	2	8	2
	Matrix	MS1	9	5	9	4
		MS2	9	4	9	4

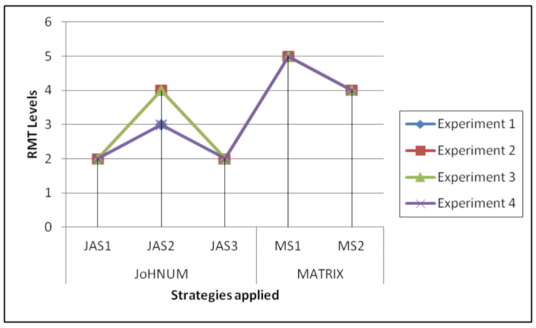
Table 3.2: Detailed evaluation summary of experiments for all JoHNUM and Matrix strategies.

3.1.6 Simulation Results and Gains

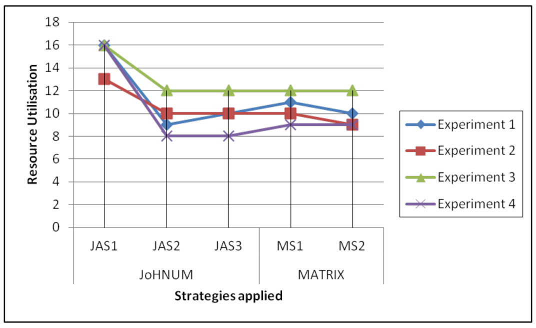
Table 3.2 presents a summary of the experiments for the first four parameters. It is observed that the results are quite similar although each experiment was performed with completely different specifications. Communication cost is discussed but not computed at this stage because both the mechanisms used almost the same number of resources for each experiment. However, it is a major metric during a scalability process and is computed during the real implementation of this work in chapter 7, in terms of increase in number of inter-sim crossings as a system scales. Evaluation results for the number of regions, RMT levels, resource utilisation and degradation of interactive experience between different strategies of JoHNUM and Matrix are graphically illustrated in Figures 3.7(a)-(d) respectively.



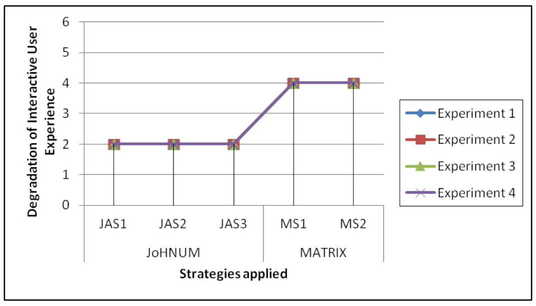
(a)



(b)



(c)



(d)

Figure 3.7: Comparison of all JoHNUM and Matrix strategies for: (a) total number of regions after the splits; (b) RMT levels; (c) Resource utilisation; (d) Degradation of interactive user experience.

JoHNUM strategies divide a world into more regions than Matrix. Individual region assignment (proposed in JAS1) demands more resources and, therefore, results in resource under-utilisation. Furthermore, this strategy might assign regions to servers without players. However, aggregate assignment is adopted to solve this problem. JAS1 improves interactive user experience over Matrix. JAS2 applies aggregate assignment to eliminate the limitations of JAS1, but greatly increases the levels in an RMT. However, the interactive user experience remains the same as JAS1. Experimental results show that JoHNUM is more flexible and performs better than Matrix both in terms of RMT levels and interactive user experience. Moreover, JoHNUM selects fewer resources the same as Matrix by using aggregate assignment and, therefore, also reduces communication overhead (inter-server communication). The management strategy used in JAS2 greatly degrades JoHNUM performance. However, a slight modification to the management strategy (adopted in JAS3) outperforms Matrix and basic JoHNUM strategies. JAS3 not only displays better performance and interactive player experience but potentially keeps the communication overhead almost the same as that of Matrix. The JoHNUM strategies, developed in this study, except JAS2, output RMTs of two levels that show a significant improvement over Matrix strategies. However, JAS2 results in a RMT of four levels at maximum while MS1 and MS2 produced RMTs of five and four levels correspondingly. Similarly, resource utilisation in JAS2 and JAS3 is almost the same as MS1 and MS2. Moreover, JoHNUM strategies improve interactive user experience by 50 percent over that of Matrix due to pre-processed partitions.

Experiment Number	Algorithm	Regions	RMT Levels	Resource Utilisation	Degradation of Interactive Experience
1	JoHNUM	16	2	10	2
	Matrix	10	4	10	4
2	JoHNUM	13	2	10	2
	Matrix	9	4	9	4
3	JoHNUM	16	2	12	2
	Matrix	12	4	12	4
4	JoHNUM	16	2	8	2
	Matrix	9	4	9	4

Table 3.3: The evaluation summary of JoHNUM and Matrix

Summary

Simulation results show that JAS3 is a better strategy among the JoHNUM strategies and MS2 gives better results than MS1 for Matrix. The rest of this work uses the concept of JAS3 to manage a hierarchy of resources for JoHNUM infrastructure. Therefore, from now onwards, we use JoHNUM for JAS3 and Matrix for MS2. Simplified experimental results for JoHNUM and Matrix are reproduced again using the same metrics from Table 3.2, and their summary is presented in Table 3.3. Best available results for number of regions, RMT levels, resource utilisation and degradation of interactive experience for both JoHNUM and Matrix are illustrated in Figures 3.8(a)-(d) correspondingly.

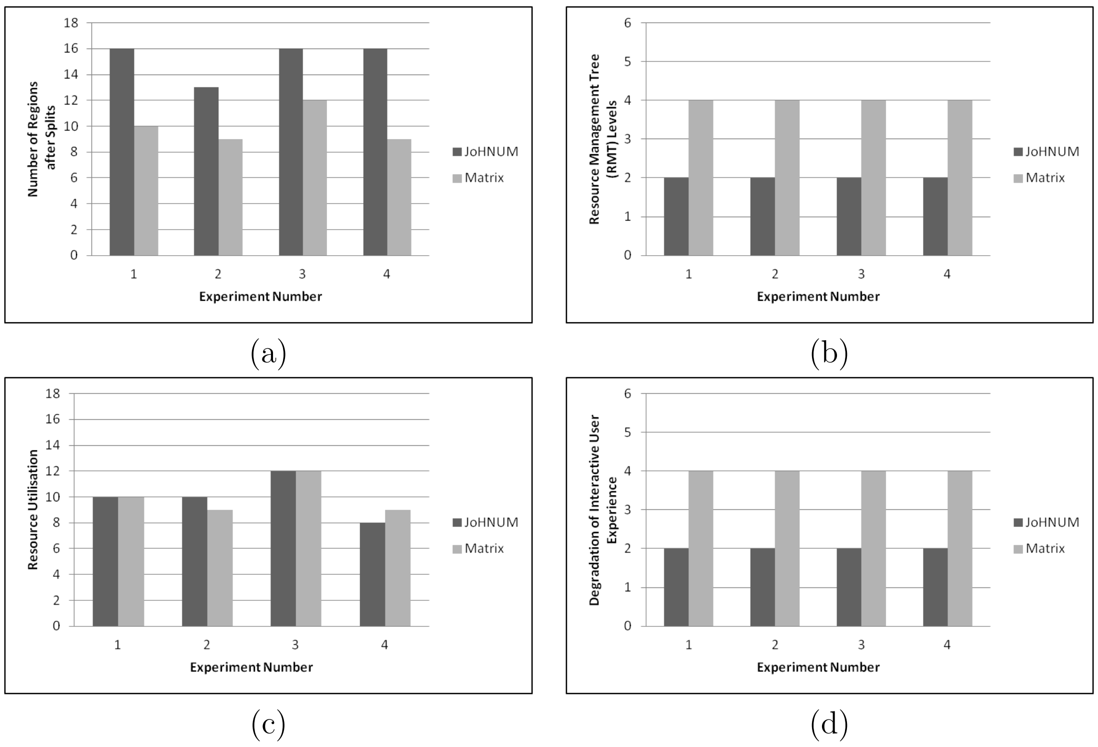


Figure 3.8: Comparison of JoHNUM and Matrix for: (a) Total number of regions after the splits; (b) RMT levels; (c) Resource utilisation; (d) Degradation of interactive user experience.

Figure 3.8(a) shows that JoHNUM divides a world into more regions than Matrix. It utilises almost the same resource as Matrix, as is shown is Figure 3.8(c) based on the concept of aggregate assignment. JoHNUM outperforms Matrix both in terms of RMT levels and degradation of interactive user experience, as illustrated in Figure 3.8(b) and Figure 3.8(d), respectively.

3.1.7 Discussion

Proof-of-the-concept simulation only used splitting into four or nine smaller regions based on the player distribution and obtains improved results over the traditional hierarchical structures such as Matrix. JoHNUM helped to reduce the number of levels in an RMT due to splitting a space into more than two regions, and improved interactive experience due to pre-processed regions. The actual implementation also has the benefit of transferring the smaller regions in an aggregate in turn, which further improves the user interactive experience (this is further discussed in chapter 7). Based on the fact that a region can even be divided into more than nine regions, the levels in an RMT for JoHNUM would be further reduced. It has the potential to reduce delays in hierarchical structures by reducing intermediate points among the components. It would also increase the overall system performance and user interactive experience.

3.2 Load Distribution

In this section, we examine the ARA algorithm and aggregation strategies which try to balance the load between two servers as much as possible. It provides a set of illustrations and simulation to evaluate its benefits.

3.2.1 Introduction

The ARA algorithm uses a number of aggregation strategies to distribute the load in as balanced a way as possible between two servers. It combines only those regions sharing physical boundaries, and no horizontal or vertical lines are allowed to go out of one aggregate, into the other and back again. However, a diagonal region is considered for aggregation if it shares boundaries with regions already in one of the two sets of regions during the aggregation process. The main objective is to avoid islands and peninsulas, and obtain two contiguous areas for assignment.

It minimises transfers between servers when players move between regions. The world is in the form of a tiled grid having square-shaped regions and obtained as an output from the partitioning algorithm of JoHNUM infrastructure. The ARA algorithm works from a starting point, named a root, in the set of regions to be distributed. Keeping these restrictions in mind, the corner regions are chosen as roots.

3.2.2 The Algorithm and Strategies

The Top Left (TL) and Top Right (TR) regions of the grid are selected as the roots for this work. However, any combination of two consecutive corner regions can be chosen which guarantees scanning all possible and unique combinations. However, the strategies might require minor changes regarding directions with different roots. Four strategies are proposed for each root, and these are presented in Table 3.4. Nevertheless, the RSF value determines the number of strategies required in each case. The first two strategies for an RSF value of 2, and all four for an RSF value of 3 or greater, guarantee examining the entire set of unique and valuable combinations when applied to both TL and TR.

Root initiating scanning	Strategy Number	Aggregation strategy	Applied for RSF Value of
Top Left Region (TL)	1	Left to Right, Row by Row (LRRows)	2, 3 and greater
	2	Top to Bottom, Column by Column (TBColumns)	2, 3 and greater
	3	Left to Right and Top to Bottom (LRaTB)	3 and greater
	4	Left to Right and Top to Bottom with Diagonal Region (LRTBwDR)	3 and greater
Top Right Region (TR)	1	Right to Left, Row by Row (RLRows)	2, 3 and greater
	2	Top to Bottom, Column by Column (TBColumns)	2, 3 and greater
	3	Right to Left and Top to Bottom (RLaTB)	3 and greater
	4	Right to Left and Top to Bottom with Diagonal Region (RLTBwDR)	3 and greater

Table 3.4: Summary of roots and their corresponding aggregation strategies [60].

The proposed strategies deal with two sets of regions named Aggregate1 and Aggregate2 during exhaustive aggregations. **Aggregate1 and Aggregate2 are coloured black and grey** for illustration purposes in the diagrams used in this work. Initially, Aggregate1 includes the root region while Aggregate2 holds the rest of the regions. Each successive iteration of a strategy transfers one region from Aggregate2 to Aggregate1, as shown in Figures 3.9(a)-(d) for an input matrix of nine regions and root TL. However, different patterns are followed by the strategies which are explained later for an RSF value of 3. Furthermore, it is observed that some strategies repeat a number of combinations. However, the ARA algorithm exploits a number of techniques to reduce the scanning process. Strategies 3 and 4 scan the rows and columns simultaneously by selecting one region in an iteration from them in turn (see Figures 3.9(c)-(d)) compared with the first two strategies which are visiting either rows or columns in a row-by-row or column-by-column order. They add a number of valuable and unique combinations with (in strategy 4), or without (in strategy 3), considering the diagonal regions, as explained later. Figure 3.10(a) outlines the first two strategies for an RSF value 2 and both roots (TL and TR) with repetitions. Furthermore, a total of six unique combination are possible in this case, as shown in Figure 3.10(b), and the repetitions are marked as R and skipped.

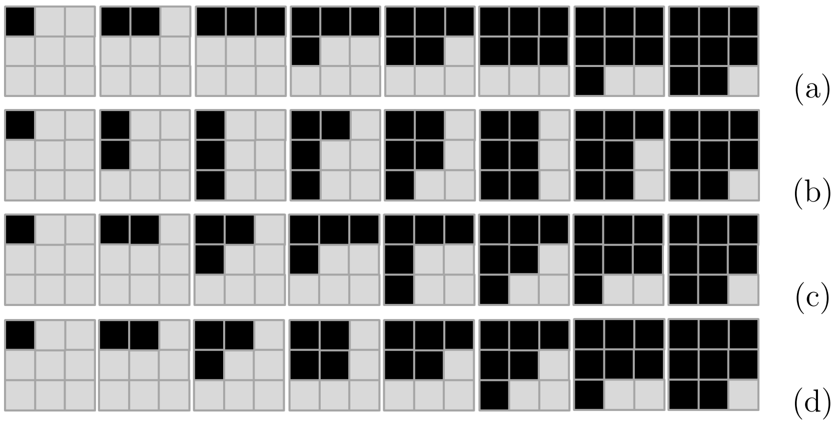


Figure 3.9: Illustration of aggregation strategies of ARA algorithm for root TL and an RSF value 3 for: (a) LRRows; (b) TBColumns; (c) LRaTB; (d) LRTBwDR.

The LRRows strategy for root TL examines the possible combinations by visiting the regions row by row, from left to right. It gives a total of eight unique combinations (see Figure 3.9(a)). Similarly, TBColumns performs a column-by-column scanning from top to bottom yielding six unique combinations (see Figure 3.9(b)) with two repetitions. RLRows and TBColumns strategies for root TR inspect the regions row by row and column by column, from right to left correspondingly. The third strategy for both roots (LRaTB and RLaTB) considers a single region from rows and columns in turn, as shown in Figure 3.9(c), for root TL. It only defines three unique combinations with a number of repeated ones (five in this case). The fourth strategy (LRTBwDR) is similar to the third strategy but it reads the adjacent diagonal regions when the two regions sharing boundaries with it are already in an aggregate. This is depicted in Figure 3.9(d) and it reads only one unique combination with seven repeated aggregates. However, these strategies add more unique combinations for bigger grids. The proposed technique can cutoff a number of repetitions if the aggregates initially consider regions of the first unique combination (skipping the iterations for regions without players). Strategies for the root TR can be visualised with the same approach with one exception, that they move in the opposite direction.

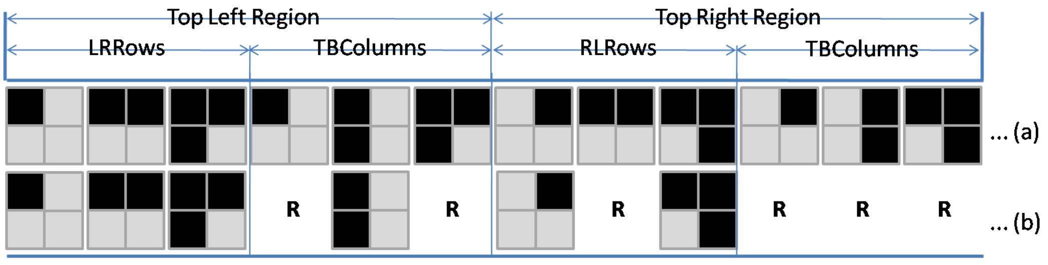


Figure 3.10: Illustration of LRRows and TBColumns strategies for an RSF value of 2 and both TL and TR.

Since a region can be divided into more than nine sub-regions, such as sixteen or greater based on player orientation, we use Figure 3.11(a) and 3.11(b) to show how strategies 3 (LRaTB) and 4 (LRTBwDR) are applied to a grid of sixteen smaller regions. The first two strategies are straightforward and are not illustrated here. The illustrations include repeated aggregates just for understanding purposes and they are skipped by the ARA algorithm. Strategy 3 is illustrated iteration by iteration in Figure 3.11(a), while Figure 3.11(b) illustrates strategy 4. This is not explained further as the illustrations are self-explanatory. Larger regional grids are also processed in the same fashion. However, it is noted that dividing into more regions gives more unique combinations compared with 3×3 grid combinations of the same strategies. The LRaTB strategy gives eleven unique combinations, with only four repeated. Similarly, LRTBwDR also adds nine unique combinations, with six repeated.

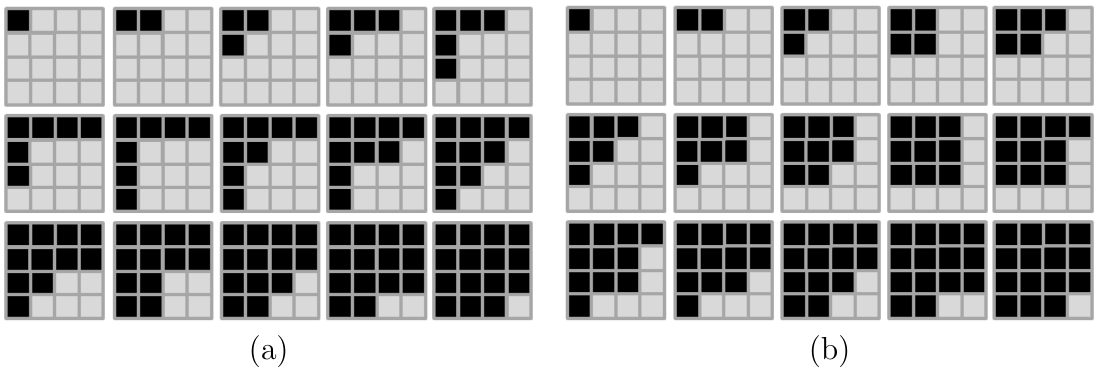


Figure 3.11: The 4×4 regional grid illustration of root TL for (a) LRaTB, and (b) LRTBwDR strategies.

The pseudocode of ARA is presented in Algorithm 1. It takes Roots[], Search-Strategies, Players, RSF, and a Regional Players Matrix (RPM) holding player density for the regions as input. It reads the corresponding strategies for an RSF value and applies them in turn. The ARA algorithm performs an exhaustive search to achieve fair distribution of load, and minimises the scanning process by avoiding repetitions. It investigates the unique combinations and terminates if it finds uniform load at any point. It also skips the remaining combinations of a strategy if the observed difference for a combination is greater than the difference for the previous combination. It is clear that the rest of the combinations for this strategy cannot further achieve a better balance of load. The proposed algorithm maintains BestAggregate1, BestAggregate2 and BestDifference where BestDifference is used for deciding best aggregates, and the rest hold the two sets of regions whose difference is the minimum one. The proposed strategies and conditions greatly reduce the effort of finding the best possible distribution of load, as illustrated later in this chapter.

Algorithm 1 The Aggregate Region Assignment (ARA) Algorithm for Load Distribution

```
Require: Players, Roots[], SearchStrategies, RSF, RegionalPlayersMatrix
Ensure: BestAggregate1, BestAggregate2
 1: Flag ← false
 2: for $i \leftarrow 1$ to Max(Roots[]) do
 3:     BestDifference ← Players
 4:     Strategies[]=Read corresponding strategies for the root considering the RSF Value
 5:     for $j \leftarrow 1$ to $Max(Strategies[])$ do
 6:         Initialise Aggregate1 and Aggregate2
 7:         Difference ← Players
 8:         while (All combinations are not visited) do
 9:             Determine Aggregate1 and Aggregate2 for each successive step as described by the strategy
10:             if (Combination not yet visited) then
11:                 Compute AggregateTotal1 and AggregateTotal2
12:                 if (absolute(AggregateTotal1 − AggregateTotal2) > Difference) then
13:                     break
14:                 else
15:                     Difference ← absolute(AggregateTotal1 − AggregateTotal2)
16:                 end if
17:                 if (Difference < 2) then
18:                     Flag ← true
19:                     BestAggregate1 ← Aggregate1
20:                     BestAggregate2 ← Aggregate2
21:                     BestDifference ← Difference
22:                     break
23:                 else
24:                     if (Difference >= BestDifference) then
25:                         go to the next combination
26:                     else
27:                         BestAggregate1 ← Aggregate1
28:                         BestAggregate2 ← Aggregate2
29:                         BestDifference ← Difference
30:                     end if
31:                 end if
32:             end if
33:         end while
34:         if ($Flag == true$) then
35:             break
36:         end if
37:     end for
38:     if ($Flag == true$) then
39:         break
40:     end if
41: end for
```

The ARA algorithm results in two sets of regions (BestAggregate1 and BestAggregate2) and the server that initiates the split then assigns one set of regions to the child server while keeping the other for its own processing. It minimises the content un-availability time and reduces the number of players that suffer from a transfer by sending multiple regions in an aggregate in turn (as in our implementation). In the worst cases, it must examine the entire set of possible and favourable combinations for the corresponding strategies. The favourable combinations are those that further reduce the difference and, therefore, must be examined while maintaining regular and contiguous spaces.

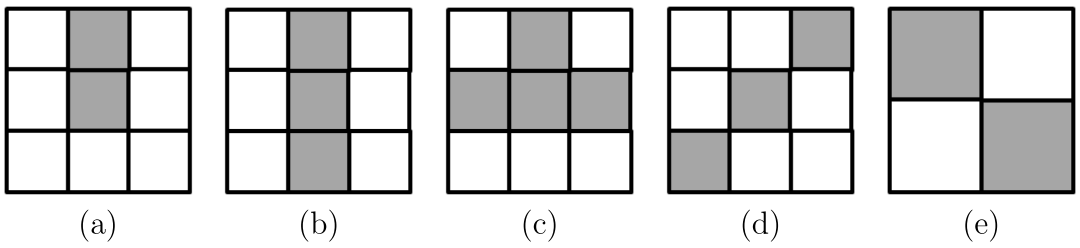


Figure 3.12: Odd cases excluded by the ARA Algorithm. (a) Irregular content distribution. (b) A case splitting an aggregate into two isolated groups. (c) A case splitting an aggregate into three isolated groups. (d) Aggregation with diagonals splitting an aggregate in 2 different isolated groups while having no physical boundaries among the regions of the other aggregate. (e) Aggregation with diagonal for RSF value 2 splitting into two aggregates where regions in both have no physical boundaries (each aggregate has isolated groups of one region each).

Motivation and Discussion

Two consecutive corner regions (TL and TR) have been used being roots that have the potential to determine the complete set of valid combinations with our aggregation strategies. It has a great impact on performance and load distribution. The ARA algorithm uses these for load balancing while obtaining contiguous and regular areas for assignment. The basic reasons for the selection of the proposed roots and strategies include content visibility and disconnection, communication, player migration and implementation concerns. These issues are relevant to each other, and are explained with the help of Figures 3.12(a)-(e) by means of five different cases. Content in these cases is divided into non-contiguous areas that increase communication traffic and player migrations between servers. Moreover, these increase implementation complexity as a server maintains isolated sets of regions which are difficult to manage. A player requiring content information from a neighbouring region needs to pass a request to the server and, in case a player moves, they experience a number of disconnections and connections between the same servers. Keeping these issues in mind, the proposed strategies and roots are selected in order to achieve a fair distribution of load while maintaining regular and continuous spaces. It minimises risks for the issues described above. An abstract model discusses this issue with an assumed set of scenarios in section 3.2.4.

3.2.3 Simulation Results

Simulation Setup and Assumptions

A number of different cases are considered for illustrations in MATLAB and a world is represented by an n×n matrix. The assumptions and parametric values are summarised in Table 3.5, including dimension of the matrix representing a world, player distribution, the MSC, and an RSF value for each case. We have considered worlds of different dimensions and player distributions to obtain fair results. The matrix dimensions and the MSC values which are selected can be used with RSF values of both 2 as well as 3. Hotspot scenarios are common in VWs and are considered for obtaining RSF values greater than 2. It is assumed that as player density exceeds the MSC, the server triggers the partitioning algorithm. Therefore, the number of total players is one more than the MSC in each case.

Case	Dimension of VW Matrix	Distribution	MSC	RSF
1	4*4	Uniform	7	2
2	4*4	Uniform	7	2
3	6*6	Uniform	8	2
4	6*6	Hotspot	8	3
5	12*12	Uniform	30	2
6	12*12	Hotspot	30	3
7	18*18	Uniform	75	2
8	18*18	Hotspot	75	3

Table 3.5: Assumptions and parametric values for the illustrations.

The JoHNUM partitioning algorithm takes a world as an input, and returns an

RSF value and the corresponding regions in the form of an RSF×RSF matrix. This matrix is described as a Regional Players Matrix (RPM) representing the tiled grid of regional players. Figure 3.13 presents the tiled grids for the worlds used in this simulation study. Each grid shows the regions, their boundaries and player distribution in different regions after splitting by the JoHNUM partitioning algorithm. The ARA algorithm takes the RPM as an input. Roots and their corresponding strategies are applied in the order of presentation and step-by-step illustrations are presented in Figure 3.14 and Figure 3.15. It skips repeated patterns and terminates if uniform load is achieved at any stage. If the difference between AggregateTotal1 and AggregateTotal2 is less than 2, it declares the achievement of uniform load and terminates. It also skips the remaining combinations for a strategy when absolute difference for a combination is greater than the difference for the previous combination, because difference between the remaining combinations increases in further iterations. Cases where the uniform load cannot be achieved, it examines all favourable possibilities and determines the best aggregations. It assures the best possible load distribution with contiguous patterns. It is worth mentioning here that better load distribution may be possible, but not with one of these chosen partitions.

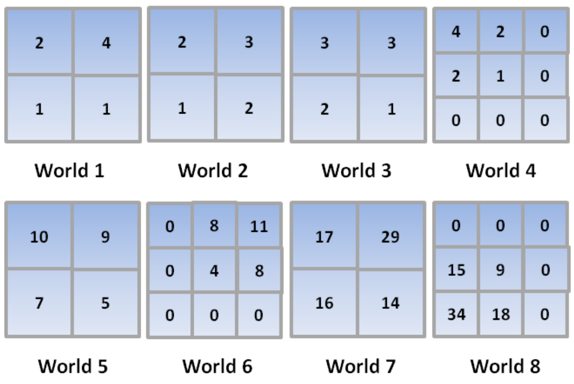


Figure 3.13: Example worlds considered for illustration purposes being presented as tiled grids of 4 and 9 regions with player density.

Illustrations and Outcomes

Figure 3.13 shows the regions and their corresponding players for different example worlds in the form of tiled grids. These worlds have been split into two sets for their step-by-step illustrations. The illustrations of cases for RSF value 2 are depicted against the possible and unique combination in Figure 3.14 for a reference. Similarly, the cases for RSF value 3 are illustrated by Figure 3.15. Figure 3.14 provides comprehensive information while Figure 3.15 skips the details due to the large number of possible combinations. Three tags **Repetition Skipped (RS), Strategy Skipped (SS) and Cannot Improve (CI)** are used to represent skipping combinations at different levels of the algorithm and this starts with root TL. The terms RS, SS and CI are used to show the three cut-off conditions. The best aggregates are modified when an aggregation with better load distribution than the stored one is achieved. Step-by-step illustrations are sketched from left to right in all diagrams.

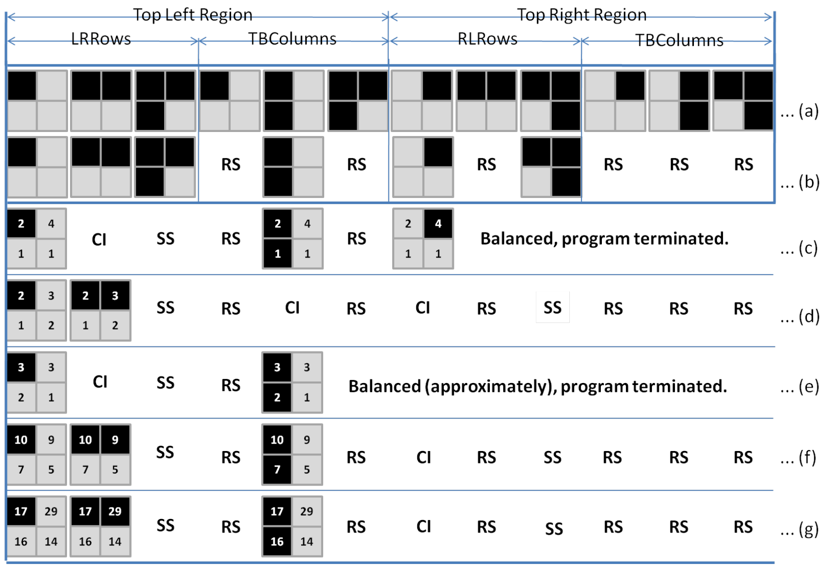
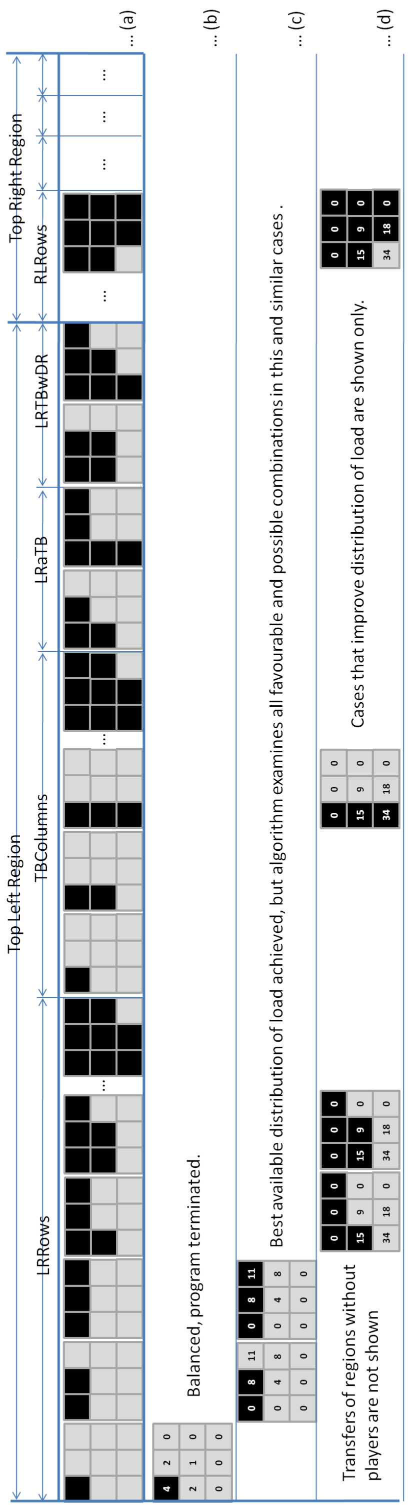

Top Left Region
LRRows
TBColumns
Top Right Region
RLRows
TBColumns
... (a)
RS RS RS RS RS RS
... (b)
CI SS RS RS
Balanced, program terminated.
... (c)
SS RS CI RS CI RS SS RS RS RS
... (d)
CI SS RS
Balanced (approximately), program terminated.
... (e)
SS RS RS CI RS SS RS RS RS
... (f)
SS RS RS CI RS SS RS RS RS
... (g)

Figure 3.14: Illustrations of the proposed combinations and worlds for an RSF value 2: (a) a complete set of possible combinations; (b) a complete set of possible unique combinations; (c)-(g) illustrations of worlds 1, 2, 3, 5, and 7 respectively.

The first set of worlds for 2×2 regional grids are shown in Figure 3.14. World1 is illustrated by Figure 3.14(c) which skips two unique combinations, marked as CI and SS. It achieves uniform load distribution by visiting the fifth unique combination and terminates. World3 (see Figure 3.14(e)) is the same as world1 and finds the uniform load when it scans the fourth unique combination. World2 (see Figure 3.14(d)), world5 (see Figure 3.14(f)) and world7 (see Figure 3.14(g)) cannot achieve uniform load and, therefore, they must search for all possible and favourable combinations. However, the actual computations are greatly reduced by the proposed filters. These cases (world2, world5, and world7) yield load distributions of 5:3, 17:14 and 33:43 for BestAggregate1:BestAggregate2, consecutively. Nevertheless, it is clear that some combinations, such as diagonals in world2 (see Figure 3.14(d)) and world5 (see Figure 3.14(f)), achieve better distribution of load, but these are not considered by the ARA algorithm due to the implementation and performance issues. It can be observed that the cut-off conditions reduce the aggregations considered by almost 50 percent, compared with the unique combinations.

World4, world6, and world8 (second set of bigger worlds) are explained with the help of Figures 3.15(b)-(d) respectively. Figure 3.15(a) provides a partial set of strategies for both roots TL and TR. World4 achieves uniform load distribution on the first iteration and terminates by returning total aggregates of 4 and 5 for the best aggregates. However, the remaining two cases examine the entire set of combinations. Some of the information is not shown in the diagrams for these cases, but the same approach as in Figure 3.14 is used. World6 and world8 return load distribution of 19:12 and 34:42 for BestAggregate1:BestAggregate2 consecutively. These worlds also include combinations that further improve load balancing but they are not considered by our approach. It is observed that the aggregations are further reduced for worlds divided into nine regions. Dividing a world into much smaller regions achieves better distribution of load but introduces a number of different issues that are discussed next.



Top Left Region
Top Right Region
LRRows
TBColumns
LRaTB
LRTBwDR
RLRows
... (a)
Balanced, program terminated.
... (b)
Best available distribution of load achieved, but algorithm examines all favourable and possible combinations in this and similar cases .
... (c)
Transfers of regions without players are not shown
Cases that improve distribution of load are shown only.
... (d)

Figure 3.15: Illustration of the proposed combinations and worlds for an RSF value 3: (a) a complete set of possible combinations shown partially; (b)-(d) Illustrations of worlds 4, 6, and 8 respectively.

Discussion

In this section, the functionality of our ARA algorithm was demonstrated with the help of a set of experiments. We performed additional experiments with different sizes of regional grid with varied distribution of players which identified that the ARA algorithm is flexible and is capable of managing worlds of different sizes with different player distributions. Other than this, they provided no further improvements and are therefore not included.

3.2.4 An Abstract Communication Model

In this section, we present a simple communication model to show how cases excluded by the ARA algorithm introduce extra burden in terms of communication, implementation, and user migrations. This model uses three metric parameters for an assumed and restricted number of cases: total number of shared boundaries between aggregates and the interaction capacity among players of regions in different aggregates; total number of isolated regions to manage; and number of connections and disconnections for a user. The cases discussed in Figure 3.12 are used for evaluation and comparison purposes. The diagonal case is the only excluded case for a tiled grid of four regions (based on an RSF value of 2) and is discussed for fair evaluation. This model assumes that a player of a region interacts with up to four neighbouring regions. Moreover, it considers the following example mobility patterns for a player:
Case1: A player at the top left region moving row-wise with an alternate left-to-right and right-to-left pattern visiting every region.
Case2: A player at top left region moving column-wise with an alternate top-to-bottom and bottom-to-top pattern visiting every region.

The excluded cases are compared with equivalent aggregates having a maximum number of possible regions, but actual scenarios might achieve better results. The evaluation results are provided in Table 3.6 showing that these cases share more boundaries and, therefore, increase inter-server communication among regions in case of communication. Our method significantly reduces communication and interaction compared with the excluded cases. Excluded cases greatly increase the implementation complexity by managing different isolated areas than the equivalent aggregates defined by the proposed mechanism. Furthermore, for the selected mobility patterns, the number of connections and disconnections in excluded cases are more than the aggregates achieved by our aggregation strategies, except for the cases shown in Figure 3.12(a) and 3.12(b). They achieve a slightly better number of connections/disconnections but share more external regions and require implementation of more isolated regions. This communication model, with a small number of cases, shows that the proposed algorithm reduces complexity and greatly reduces communication and implementation cost while achieving load in as balanced a way as possible. This is simply a justification for excluding the cases that will have a bad communication behaviour.

Serial Number	Case	Number of shared boundaries	Number of isolated regions	Number of disconnections Case1/Case2
1	Figure 3.12(a)	5	2	4/2
	Proposed equivalent aggregates	3	2	3/3
2	Figure 3.12(b)	6	3	6/2
	Proposed equivalent aggregates	3	2	3/3
3	Figure 3.12(c)	7	4	4/6
	Proposed equivalent aggregates	4	2	3/3
4	Figure 3.12(d)	8	5	6/6
	Proposed equivalent aggregates	3	2	3/3
5	Figure 3.12(e)	4	4	3/3
	Proposed equivalent aggregates	2	2	2/2

Table 3.6: Evaluation summary of the abstract communication model for cases provided in Figure 3.12.

3.3 Conclusions

This chapter examined the JoHNUM infrastructure (to achieve scalability) that is a vital part of our contemporary infrastructure for the development of scalable and consistent VWs. It is evaluated with a number of experiments, and compared with the game middleware called Matrix. Simulation results show that JoHNUM reduces the RMT levels and increases interactive user experience.

It also examined the ARA algorithm to minimise resource utilisation and communication overhead, which achieves the best possible load distribution while maintaining contiguous and regular spaces for assignments. It constitutes an essential part of our JoHNUM infrastructure which handles its assignment component (also called load distribution). Results from a large set of experiments show that our load distribution algorithm is flexible and can be used with small, medium and large scale VWs. It is seen that uniform load distribution is not always possible due to player distribution, and to our proposed split and aggregation strategies. In the worst cases, it examines the entire set of possible and favourable combinations, although the proposed intelligent techniques greatly reduce the aggregation process. In certain cases, it was observed that excluded cases can balance the load better than the proposed strategies. However, they significantly increase communication between servers as well as complexity for handling isolated areas in an aggregate.

Chapter 4

Consistent Virtual Worlds

This chapter presents a decentralised consistency management approach using a constrained communication model based on the inherent properties of VWs. VWs imitate the physical world where there is no direct effect from arbitrary events on an entity, and entities are mostly affected by activities and events generated in the neighbourhood. It uses a P2P approach in contrast with our JoHNUM infrastructure, which maintains a space in a hierarchical order [61]. The restriction on P2P communicating servers makes it simple and potentially very scalable. This chapter illustrates the relevant concepts, and proof-of-the-concept simulation shows that it maintains the traditional causality constraint. It is based on our published work [63, 66] on this topic.

4.1 Introduction

Parallel and Distributed Simulation (PADS) environments and large scale VWs normally utilise a number of resources to handle a vast amount of content and a large number of interactive players. These infrastructures are normally distributed both at infrastructure and application levels. The scheme of partitioning a virtual space and simulating it with a large number of resources makes these world scalable but, at the same time, introduces key consistency issues. Consistency management (alternatively called synchronisation or time management) is the process of maintaining the temporal order of events to have a uniform view of

the environment. Therefore, scalable systems must carefully design a synchronisation method by considering the application domain of the environment. Games and special purpose VEs use a number of optimisation techniques and normally compromise on consistency. However, VWs that claim to be a candidate for a future 3D web need to accommodate a diverse set of applications including those with a conservative nature. Therefore, using a relaxed synchronisation approach might have potential problems to deal with VWs.
In this work, we propose a fully decentralised synchronisation approach for dynamic, and potentially hierarchical, models of scalable VWs with restricted communication based on our JoHNUM strategies [61]. It is an integral part of our contemporary infrastructure for scalable and consistent infrastructure. It is flexible and considers the dynamic changes happening to the partitions that are handled by the neighbouring servers. The basic aim is to incorporate conservative applications in VWs, thus making them much stronger candidate for the 3D web and adapts the HLA TM for discrete event systems. The assumption of constrained communication in conjunction with a fully decentralised approach potentially reduces complexity and delay with a decrease in the number of interacting servers. It, therefore, improves interactive user experience. It is important to note that the proposed synchronisation approach uses a flat infrastructure for direct communication between the servers that might be handling parts of a world at different levels in a hierarchy, as shown in Figure 4.1 for a hierarchical model presented in Figure 4.2.

4.2 The Proposed Synchronisation Approach

4.2.1 Introducing the time advance mechanism

The basic aim of this work is to obtain a consistent state of a given world. It uses both flat (Figure 4.3 and Figure 4.4) and hierarchical models (Figure 4.5 and Figure 4.6) to illustrate our decentralised approach and its relevant concepts. Figure 4.3, which represents a simple world of 1×4 regional grid, is used to introduce the basic time advance mechanism used by our method. It shows two examples with respect to the regions marked with a star to highlight their neighbouring

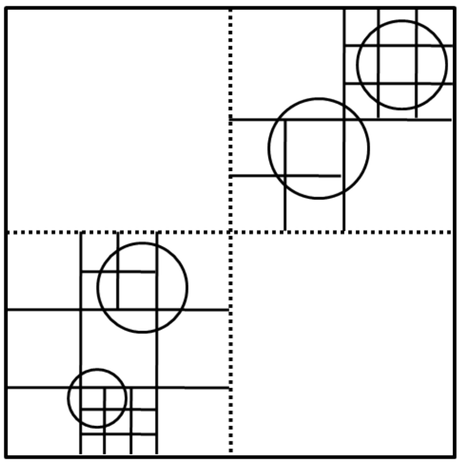


Figure 4.1: Illustrating neighbouring regions for the selected central regions in the hierarchy presented in Figure 4.2.

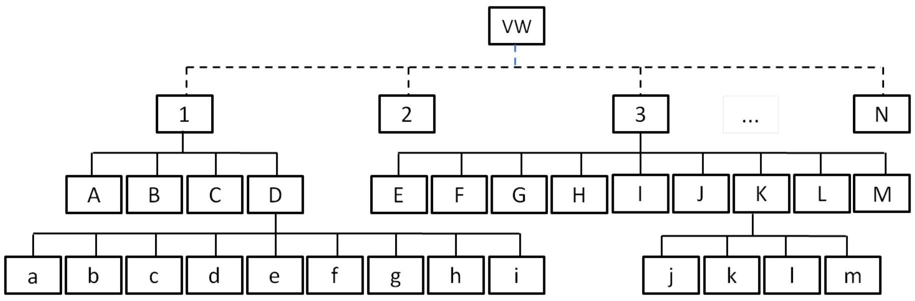


Figure 4.2: Hierarchy of a dynamic hierarchical model based on JoHNUM partitioning algorithm [61].

regions represented by circles. It gives an idea of the regions that must be considered for a time advance of a given federate. A circle highlights a federation with respect to a federate with a star such as a federation controlled by federate B having A and C as its neighbouring federates. Each named region provides its current LBTS value, Lookahead value, the latest LBTS values of adjacent regions, and the status of its Local Queue.

This mechanism considers the time information (current LBTS + Lookahead value) of all the required federates to calculate a new LBTS value of a federate. It exploits a similar approach to the basic HLA TM for discrete event simulation systems. Figure 4.3 explains the time advance with an emphasis on executing safe events for a federate B with an event A_4 from federate A with a timestamp four. The current LBTS value of federate B is three and to process this event, it calculates a new LBTS value based on the LBTS values of adjacent federates. The LBTS is the smallest among a set of LBTS values of adjacent federates (maintained as NRecord) and the timestamp of the smallest event in the LocalQueue. The new LBTS value in this case is four, which allows B to execute the event with timestamp four.

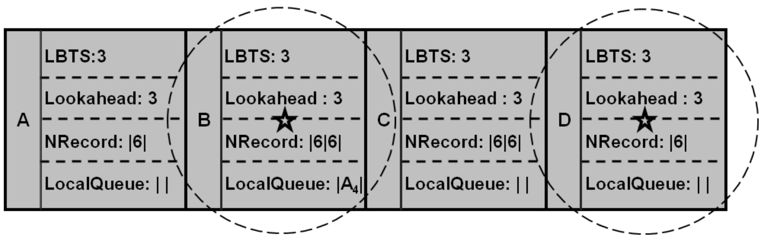


Figure 4.3: Illustration of the proposed synchronisation approach with a constrained communication model.

4.2.2 Federate, Federation, and their time relation

The terms **federate** and **federation** are re-defined for a decentralised control and a constrained communication model against the potential dynamic hierarchical models. In this work, a ***federate*** is a server executing a region that could be either a basic region or a bigger contiguous region. A ***federation*** is defined with respect to a federate and is a collection of federates that share boundaries with it.

A federate can be involved in more than one federation.
Four different concepts are introduced that need to be carefully considered for a mechanism using a fully decentralised synchronisation approach, due to their potential impact on system performance. They are the basic time advance mechanism, independent federations with no common federate, federations with a common federate, and temporarily blocked federations.
The time advance mechanism introduced in section 4.2.1 is the basic driving force of our TM approach that uses a constrained P2P environment.
Our mechanism restricts the P2P infrastructure and there might be independent federations with no common federates. Federations with no common federate, therefore, have no direct effect on the time advance of each other.
Since a federate can be a part of more than one federation, there might be federations with common federates. However, this might, or might not, block the federations sharing them.
The fourth concept is about federations that could temporary block the event processing of each other. This situation occurs in those cases when an adjacent federate for some reason is not up-to-date with other federates. However, being a synchronous algorithm, our method has the potential to recover quickly from this situation.
These cases are illustrated and explained in more detail later in section 4.2.4 with the help of Figures 4.4(a)-(d) for flat models and Figures 4.5(a)-(c) and Figure 4.6 for hierarchical models. The next section discusses our TM algorithm, which processes the events when they are safe and provides its LBTS to other federates when it increases, even if no events are processed.

4.2.3 Time Management Algorithm

In general the federation associated with a federate must include all those regions that might generate events which could directly affect the federate. Hence, each federate participates in a number of different federations, and the functionality of the RTI is distributed among federates. The concepts of LBTS and Lookahead values are used to maintain the local causality constraint. Each federate in the proposed mechanism processes its events when they are safe in consultation with the adjacent federates. It is the central part of the mechanism and follows a straightforward approach that is presented in Algorithm 2. Each federate also provides its federate LBTS value to other federates in its federation and guarantees never to generate events earlier than the federate LBTS. Hence, the local LBTS is calculated as the minimum of the neighbouring federate LBTSs and the earliest queued event (if any). The Lookahead is added to the local LBTS and is then sent to its neighbours, if and only if, the LBTS increases. This definition has a recursive nature and, especially at system startup, a number of updates to the LBTS may occur before the local LBTS reaches the point where queued events can be processed. A push strategy is used to send federate LBTS values to adjacent federates with the aim of reducing potential overhead in communication and minimising temporary blockage. A federate ensures that timestamped messages destined for a neighbouring federate are delivered before sending its LBTS information thus guaranteeing that messages will never arrive in a federate's past. Being a conservative algorithm, it always considers a positive Lookahead value. It achieves traditional guarantees and significantly reduces intermediate processing elements (hops) and dependencies in hierarchical models by directly communicating with neighbouring regions.

Each federate executing Algorithm 2 allows a safe range of event processing based on an LBTS value. It maintains a LocalQueue, LBTS and Lookahead values, and an array NRecord that stores the latest LBTS values received from the adjacent federates. A value in NRecord changes dynamically when a new LBTS value is received. The main loop is executed while the simulation is running. To guarantee that the events are processed in their temporal order, a new LBTS value is calculated at the start of each iteration. Later on, an event with the earliest timestamp is processed if it is safe (when its timestamp value is less than or equal to the LBTS), and might generate more internal and/or external events in response. It schedules new events and repeats this process for the new earliest event. A simple condition is used which never allows processing of an event with timestamp greater than the current LBTS value. To simplify the consideration of transient messages for an LBTS computation, a federate is forced to send any destined messages before sending its LBTS value. The LBTS computation in the proposed method is straightforward in terms of transient messages that might require traversal of

Algorithm 2 Decentralised Synchronisation Mechanism

```
Require: LocalQueue, LBTS, Lookahead, NRecord
    //Initialisations
    //In general, the set of adjacent federates might change dynamically based on split and merge operations [61]
 1: int n = Number of adjacent federates
 2: int NRecord[n] // maintains the latest LBTS values for adjacent federates
 3: for (i = 0; i < n; i++) do
 4:     NRecord[i] = 0 // changes dynamically with the LBTS value sent by adjacent federate i
 5: end for
 6: int LBTS = -1 //In order to force update the LBTS value
 7: int NewLBTS = 0
 8: Insert initial event(s) to LocalQueue //used for synchronisation with other federates
    //Main loop of program for safe processing
 9: while (System is running) do
10:     NewLBTS = $Min_{i=0}^{n-1}$(NRecord[i]) //determines minimum of LBTS values of adjacent federates
11:     if (LocalQueue has Events) then
12:         NewLBTS= $Min$(NewLBTS, Timestamp of earliest LocalQueue event)
13:     end if
14:     if (NewLBTS > LBTS) then
15:         LBTS = NewLBTS
16:         Send (LBTS + Lookahead) value to the adjacent federates
17:     end if
        //Check for an event that is safe to process
18:     if (LocalQueue has Events and Timestamp of earliest LocalQueue event $\leq$ LBTS) then
19:         Process Event //Remove the event and may generate new internal and external events
20:         Schedule internal and external events if any //External events are sent to adjacent federates via
            messages
21:     else
22:         Go to Sleep
23:     end if
24: end while
```

different components in traditional hierarchical systems. Time Management calculations are simple and communication is localised. At times, there might be no activity in some federates. The federate must maintain its LBTS value for its neighbours but it could sleep until an event or updated neighbouring LBTS value is received. When an LBTS update arrives, the federate updates the corresponding entry in NRecord and wakes up the process if it is sleeping. Similarly, when a new event arrives, it is added to the LocalQueue and wakes up the process if necessary. The primary aim of this current work is to ensure a consistent world with an emphasis on reducing communication delays. Other parameters, such as QoS, will hopefully be investigated in future.

4.2.4 Illustrations

In this section, we illustrate our decentralised synchronisation method with the help of a number of simple cases for the four concepts described in section 4.2.2

with the help of Figure 4.4 for flat models and using examples in Figure 4.5, and Figure 4.6 for hierarchical models. These concepts are: basic time advance, federations with no common federate, federations with a common federate (non-blocking situation), and federations with temporary blocked situations. These illustrations are based on random examples and do not provide the real state of any system. Each region in these figures is divided into two sections with a dashed line that represents the current LBTS value above, and LocalQueue below. A federate with a star means that a concept is explained with respect to it, and a circle highlights a federation with respect to a federate with a star. A static Lookahead value of 1 is assumed for these illustrations.

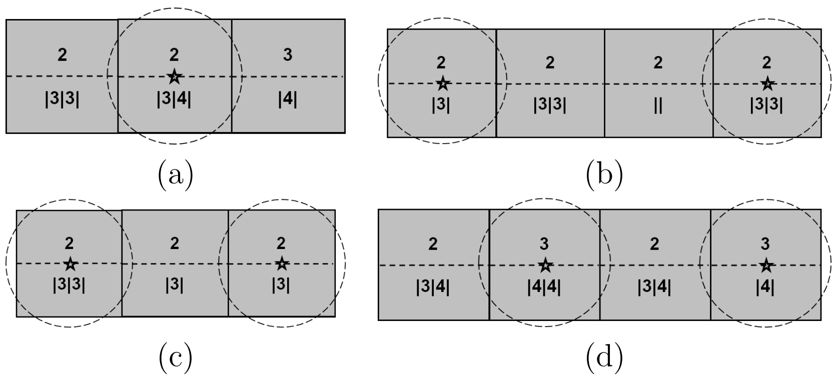


Figure 4.4: The illustration of different concepts used with our decentralised synchronisation approach. (a) The basic time advance. (b) Independent federations without a common federate. (c) Federations with a common federate (a non-blocking situation). (d) Federations with temporary blocking states.

Flat Models

The four concepts for flat models are illustrated with the help of 1×3 and 1×4 regional grids that are presented in Figure 4.4.

Figure 4.4(a) explains how the basic time advance executes safe events for a federate marked with a star. Its current LBTS value is two and wants to process an event with timestamp three. A new LBTS value is computed that is smallest among a set of LBTS values received from the adjacent federates and the timestamp of the smallest event in its LocalQueue. The new LBTS value in this case

is three, which allows the federate to execute the event with timestamp three.
This method does not impose global synchronisation and, therefore, the federations which are far apart from each other take independent decisions for their time advance, as shown in Figure 4.4(b). The current LBTS value is two for both the federates each marked with a star in these cases. The new LBTS values are three for both the federates, which allows them to process their events with timestamp three.
Federations having a common federate might be able to carry on with their processing without blockage. Figure 4.4(c) presents a case where two federates re-calculate their LBTS values to three and safely process their events with timestamp three. In certain situations, a time advance might be temporarily blocked because of adjacent federates with smaller LBTS values, as shown for the marked federates in Figure 4.4 (d). Since each federate is continuously trying to advance its time, these states are resolved quickly. In this case, other federates process their smallest timestamped events, thus allowing the blocked federates to process their events once updated LBTS values have been sent.

Hierarchical Models

The concepts for the hierarchical models are explained with the help of a system with two repeated splits (a 2×2 split, followed by a 2×2 and a 3×3 further split for two regions). However, each case considers a different system state for illustration purposes. It assumes that each federate (server) is managing a basic unit region. Although the models are hierarchical, the proposed method considers them as a flat model and, therefore, handles them the same way as flat models.
Figure 4.5(a) shows that the federate marked with star checks against the condition that time-to-advance must not exceed an LBTS value and, therefore, a new LBTS value is computed with the help of adjacent federates. It gives a new safe bound of three and allows the federate to execute the event with timestamp three and update its CurrentTime. Figure 4.5(b) shows that two federations far apart from each other take independent decisions for their time advance. Similarly, federations having a common federate might be able to advance their logical times without blocking each other, as shown in Figure 4.5(c). Figure 4.6 shows two situations

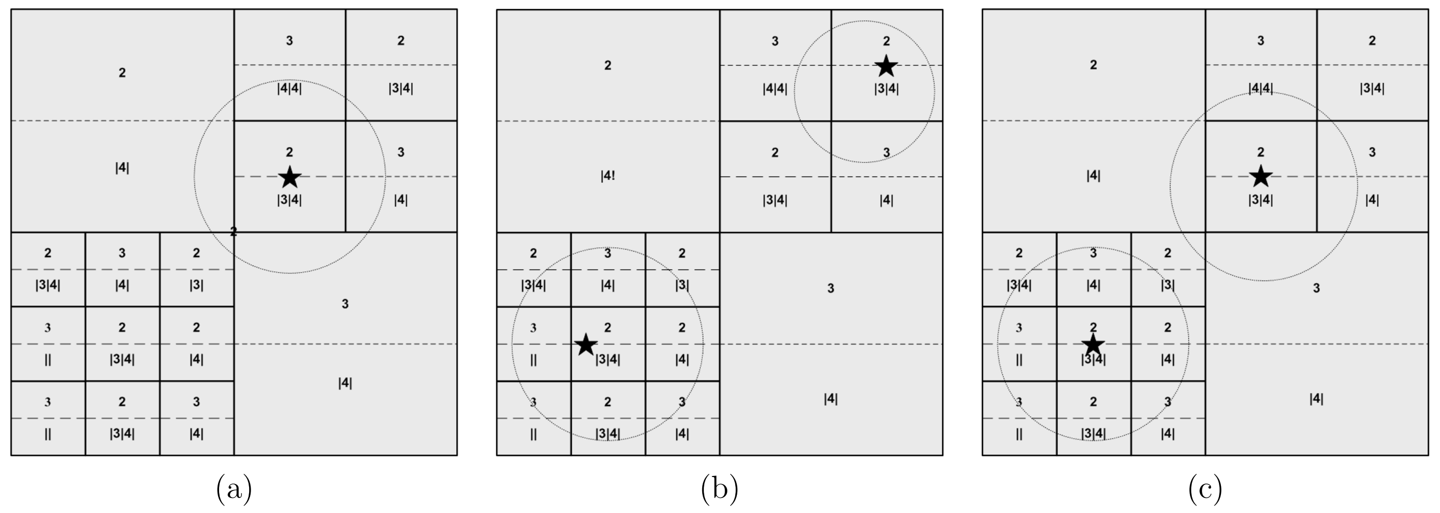


Figure 4.5: Illustration of the decentralised synchronisation approach in hierarchical models for: (a) A basic time advance; (b) Independent time computation of two federations apart from each other; (c) Two federations sharing a common federate.

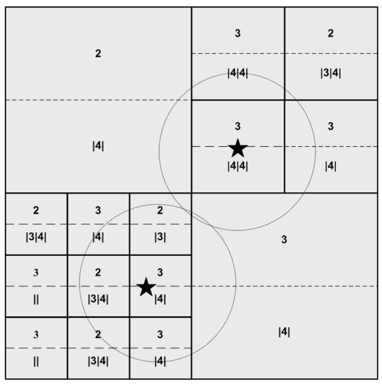


Figure 4.6: Illustrating temporary blocking states for the decentralised synchronisation method.

where the time advance is temporarily blocked because of adjacent federates with smaller values. They are quite common in decentralised environments and since each federate is continuously advancing its time, these states are expected to resolve quickly.

4.3 Evaluations and Comparisons

4.3.1 Simulation Setup

To demonstrate the effectiveness of the proposed mechanism, we have simulated the temporal order of the events for a simple scenario presented in Figure 4.7. It also shows the flow of events for a simple application that could violate local causality where agents in regions raise flags in response to other events and an observer should only see certain combinations. The scenario is that Region A raises a flag. After delay, region B copies A and, later, region C copies B. Region D observes both B and C and should never see a flag raised in C before B. However, if messages are delayed and time management is not enforced correctly, there is a potential threat to correct synchronisation.
Region A is only adjacent to B and has no direct impact on the activities of C and D. The purpose of its inclusion is to show that the proposed mechanism need not consider events from arbitrary regions. The aim of this simulation is to determine whether the proposed scheme maintains the local causality constraint. The pro-

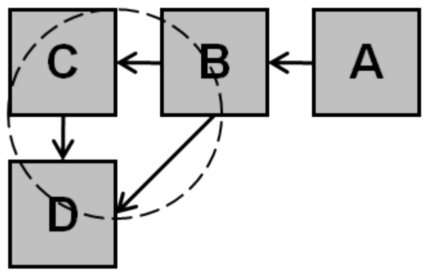


Figure 4.7: The simulated world and events flow model.

posed method considers the entire set of adjacent regions (hence, D considers both B and C) and, therefore, achieves a consistent state. However, the simulation may fail to reach a consistent state if a potential region is ignored.

4.3.2 Simulation Results

To evaluate whether the system works, both synchronised and non-synchronised scenarios are simulated in this section.

The Synchronised Scenario

A simulation run of the proposed method is presented in Figure 4.8. It assumes an initial value of -1 for the LBTS values, and a constant Lookahead value of 3 for each region. The initial values of NRecord (holding the LBTS values of adjacent regions) are all zero. The local queue of region A has an event with timestamp value of one (represented as I_1) which triggers the simulation while other queues are initially empty. The events are marked as $X_{timestamp}$, where X is the name of a region and the value of a timestamp is the sum of LBTS and Lookahead of a federate. An event generated in this simulation aims to tell the adjacent regions that it has raised its flag. The LBTS updates are sent to neighbouring regions via messages that might have different random delays. A newly generated event is placed in a corresponding Event Generated queue and sent to the adjacent regions. When these events are received by the corresponding regions, they are placed in local Queues and processed in their temporal order. A processed event at a given time is shown in bold and enclosed in square brackets. An LBTS value is calculated based on the values in NRecord that is updated each time a new LBTS for a region is calculated. Since messages between a pair of regions are delivered in sequence, the correct temporal order is maintained at any given time. A region updates the corresponding status as a flag set event is processed by a region. The system shows most of the simulation steps except for some time updates, as shown in Figure 4.8 and Figure 4.9.

	Region A					Region B					Region C					Region D (Observer)			
Step	Flag	LBTS	NRecord B	Local Queue	Event (s) Generated	Flag	LBTS	NRecord A,C,D	Local Queue	Event Generated	Flag	LBTS	NRecord B,D	Local Queue	Event (s) Generated	Flag	LBTS	NRecord B,C	Local Queue
-	0	-1	0	I_1	Empty	0	-1	0,0,0		Empty	0	-1	0,0		Empty	0	-1	0,0	
0	0	0	0	I_1	Empty	0	0	0,0,0	Empty	Empty	0	0	0,0	Empty	Empty	0	0	0,0	Empty
1	**1**	1	3	**[I_1]**	A_4	0	0	0,3,3	Empty	Empty	0	0	3,3	Empty	Empty	0	0	3,3	Empty
2	**1**	3	3	Empty	Empty	0	0	3,3,3	Empty	Empty	0	3	3,3	Empty	Empty	0	3	3,3	Empty
3	**1**	3	3	Empty	Empty	0	3	3,3,6	Empty	Empty	0	3	3,6	Empty	Empty	0	3	3,3	Empty
4	**1**	3	3	Empty	Empty	0	3	3,6,6	A_4	Empty	0	3	3,6	Empty	Empty	0	3	3,6	Empty
5	**1**	3	3	Empty	Empty	**1**	4	4,6,6	**[A_4]**	B_7	0	3	3,6	Empty	Empty	0	3	3,6	Empty
6-11	**1**	9	9	Empty	Empty	**1**	6	9,6,9	Empty	Empty	0	6	7,6	Empty	Empty	0	6	7,6	Empty
12	**1**	9	9	Empty	Empty	**1**	6	9,6,9	Empty	Empty	**1**	7	7,9	**[B_7]**	C_{10}	0	6	7,6	Empty
13	**1**	9	9	Empty	Empty	**1**	9	10,9,9	Empty	Empty	**1**	7	7,9	Empty	Empty	0	7	7,9	C_{10}
14	**1**	9	9	Empty	Empty	**1**	9	10,10,9	Empty	Empty	**1**	9	9,9	Empty	Empty	0	7	7,10	C_{10}
15-17	**1**	12	12	Empty	Empty	**1**	10	12,12,10	Empty	Empty	**1**	10	12,10	Empty	Empty	0	7	7,10	C_{10}
18	**1**	12	12	Empty	Empty	**1**	10	12,12,10	Empty	Empty	**1**	10	12,10	Empty	Empty	**B**	7	9,10	**[B_7] ,** C_{10}
19	**1**	12	12	Empty	Empty	**1**	10	12,12,10	Empty	Empty	**1**	10	12,10	Empty	Empty	**C**	10	12,10	**[C_{10}]**

Figure 4.8: Illustration of decentralised synchronisation method for a synchronised scenario.

At the start of the simulation (see Figure 4.8), a number of LBTS updates messages are processed before the initial event I_1 at region A is processed. It generates event A_4 for region B with timestamp four which is received at step four. It updates its LBTS value and sends a message to region B to update its NRecord value. However, region B has to wait until time update messages are processed, thus allowing event A_4 to process at step five. Region B generates event B_7 for region C and D. After each region generate an update message, it sends an LBTS update message which is processed by corresponding regions in temporal order. The event B_7 arrives at region C at step twelve, but it is received at region D at step eighteen, even later than the response event C_{10} generated by region C for D (which arrives at step thirteen). However, results show that the proposed mechanism does not allow D to process event C_{10} until after it receives and processes B_7. This demonstrates that the events are processed in their temporal order.

The Non-Synchronised Scenarios

The same specifications are used to simulate a non-synchronised approach which shows that it violates the local causality constraint. Two different scenarios are possible: in the first scenario, D considers only its own time information; but in the second scenario, D considers C (but not B), in addition to its own time information. The simulation result for the first scenario is presented in Figure 4.9. It is clear that at step thirteen, event C_{10} is processed before the arrival of event B_7; hence, a causality violation has occurred. The second scenario also violates the temporal order giving similar behaviour and is, therefore, not included in this work.

	Region A					Region B					Region C					Region D (Observer)			
			NRecord	Local	Event (s)			NRecord	Local	Event (s)			NRecord	Local	Event (s)			NRecord	Local
Step	Flag	LBTS	B	Queue	Generated	Flag	LBTS	A,C,D	Queue	Generated	Flag	LBTS	B,D	Queue	Generated	Flag	LBTS	B,C	Queue
-	0	-1		I_1	Empty	0	-1		Empty	Empty	0	-1		Empty	Empty	0	-1		Empty
0	1	1		**[I_1]**	A_4	0	-1		Empty	Empty	0	-1		Empty	Empty	0	-1		Empty
1-3	**1**	1		Empty	Empty	0	-1		Empty	Empty	0	-1		Empty	Empty	0	-1		Empty
4	**1**	1		Empty	Empty	1	4		**[A_4]**	B_7	0	-1		Empty	Empty	0	-1		Empty
5-11	**1**	1		Empty	Empty	1	4		Empty	Empty	0	-1		Empty	Empty	0	-1		Empty
12	**1**	1		Empty	Empty	1	4		Empty	Empty	**1**	7		**[B_7]**	C_{10}	0	-1		Empty
13	**1**	1		Empty	Empty	1	4		Empty	Empty	**1**	7		Empty	Empty	**C**	10		**[C_{10}]**
14-17	**1**	1		Empty	Empty	1	4		Empty	Empty	**1**	7		Empty	Empty	**C**	10		Empty
18	**1**	1		Empty	Empty	1	4		Empty	Empty	**1**	7		Empty	Empty	**B**	7		**[B_7]**

Figure 4.9: Illustration of a simulation run for the non-synchronised approach.

A Non-Restricted P2P scenario

Simulating a P2P environment with no restrictions on the number of federates considers the entire set of regional LBTS values for the LBTS computation of a region which introduces longer delay in the computation of a new (effectively global) LBTS value. In this approach, region A is also considered for LBTS computations by C and D, even though it has no direct impact and significantly increases exchange of messages over the network. A region has to wait for A's time advance to proceed. The P2P approach achieves the same results as the proposed method, but it has bottlenecks for very large simulation environments. It also increases computation as well as communication overhead (the basic management issue in P2P systems). Further investigations on this issue are our future work.

Summary

In summary, all regions affecting a federate must be part of its federation, but no additional regions need be included. Based on the simulation results, it can be seen that the proposed method maintains the local causality constraint. It considers a limited number of regions and is, therefore, more scaleable and efficient compared with traditional centralised approaches. It potentially reduces communication overhead compared with P2P environments. In our examples with small numbers of regions, local calculations are based on a large proportion of the total regions but, in a large scale simulation environment, only a very small proportion of regions would be involved in each calculation, making the proposed method flexible and scaleable.

4.3.3 An Abstract Model for Comparison

Our synchronisation mechanism considers a complex hierarchical model (see Figure 4.2) at a single level, as shown in Figure 4.1 based on our previous work [61]. A number of parameters might be used to compare it with traditional hierarchical approaches with centralised and distributed control. These parameters include dependencies, number of hops, complexity, delay, and scalability. It is worth mentioning that some of these parameters are dependent on each other. The dependencies among the components of hierarchical models and the intermediate processing points are the basic reasons for an increase in complexity, longer delays, and poor scalability. The traditional hierarchical approaches with distributed control are comparatively scalable and easy to implement, but they need to keep their interfaces as simple as possible. However, when used with conservative approaches, the dependencies among different components at multiple levels (as discussed by Cramp et al. [41]) introduce longer delays and are therefore not very scalable, though better than centralised approaches. A message in a hierarchical structure has to pass through a number of hops, thus not only increasing complexity but also delays. Our decentralised mechanism avoids going through intermediate points by adopting a direct communication between interacting federates. Similarly, the federates cannot proceed further due to global dependencies among the components at different levels in existing mechanisms, thus degrading the overall interactive experience. This method uses a restricted P2P model thus solving the management and communication issues. It allows different federates to process and advance their time without waiting for others having no impact on them. It has no central point failure issues compared with centralised approaches.

Serial Number	Levels in Hierarchy	Algorithm	Number of hops	Complexity	Delay	Blocking Levels	Scalability
1	2	Hierarchical	3	4×X	4×Y	Fully	Poor
		Fully Decentralised	0	X	Y	Partially	Good
2	3	Hierarchical	5	6×X	6×Y	Fully	Poor
		Fully Decentralised	0	X	Y	Partially	Good
3	4	Hierarchical	7	8×X	8×Y	Fully	Poor
		Fully Decentralised	0	X	Y	Partially	Good

Table 4.1: A abstract comparison of hierarchical methods with our decentralised synchronisation mechanism.

In this work, an abstract model has been adopted to compare the decentralised approach with existing hierarchical models while detail analysis is our future work. Table 4.1 presents the details of comparison for three different random hierarchical models of depth two, three, and four. It is important to note that this comparison is based on a world with a global consistent state. The traditional approaches use the maximum number of three, five, and seven hops (intermediate points) correspondingly compared with no hops involved in our method. Our method involves only a single step to communicate with neighbours. Based on the number of hops, we believe the complexity and delay are increased significantly. If this approach has complexity X and Delay Y, then traditional approaches have complexity = (X × number of links traversed) and delay = (Y × number of links traversed) in each case. Similarly, this mechanism is partially blocked compared with current approaches that might be blocked completely for certain decisions, such as a time advance. We believe that the proposed approach is potentially more scalable than the traditional schemes and is capable of resolving the blocking state efficiently.

4.4 Global Consistency in Virtual Worlds

Our current decentralised synchronisation approach employs locality and basically targets the most general category of virtual world applications that have only local effects. It provides global consistency for these applications but restricts the application domain of the virtual worlds. If we allow activities spanning arbitrary federates, there is no guarantee of global consistency. In this section, we address the issue of maintaining globally consistent virtual worlds, and how to accommodate the complete set of possible applications while avoiding using a global strategy. It briefly explores the current mechanisms using local consistency to achieve global consistent states of their corresponding applications. Our current approach is illustrated and its limitations are identified with the help of examples for the extended set of applications. We then categorise the applications of virtual worlds and suggest a possible dynamic adaptive strategy to obtain a global consistent state as an extension to our current mechanism.

4.4.1 The Literature

In this section, we present the existing methods for a diverse set of applications and explore the techniques they use to achieve global consistency. These applications include Reasoning [52], Constraint Networks [218], Learning [245], and Diagnosis [123]. The Literature shows a large set of applications that have their unique requirements and therefore a number of techniques are presented to reduce the computation, communication, and implementation overheads. The most widely techniques exploit the concept of locality that is used to determine the level of consistency for obtaining a global coherent model.

Dechter [52] presented a relationship among the size of variable domains, the set of constraints, and the level of local consistency required to obtain global consistency for reasoning tasks involving consistent databases called constraint networks. According to this author, all realistic models of human reasoning use the concept of locality such as conceptual memories where the activity spreads among neighbouring entities. Locality is used in reasoning to enforce local consistency that simplify the task to get a global consistent model of data. The conditions that help to achieve global consistency are based on the topological properties of the network representing the interactions among data items. A number of theorems were developed to show the achievement and correctness of globally consistent constraint networks. Furthermore, a number of different relevant examples were presented to illustrate their work. Waltz [225] presented a scene labelling scheme that often obtains global consistent objects by executing only neighbouring edges and vertices in each step. Beek et al. [218] further expanded the constraint networks by using two contemporary properties (called tightness and looseness) on restrictiveness of constraints in a network. According to them, using constraint tightness and level of local consistency guarantee that a solution can be found in a backtrack free manner. Constraint looseness is used to determine the level of local consistency of a network to achieve a global consistency. They explained and evaluated their work also with another type of consistency called relational consistency.

Learning from labelled and un-labelled data is another general problem that employs local consistency. The labelling process is propagated through the neighbouring points until a global state is achieved. This method is often called semi-supervised learning which requires a sufficiently efficient classifying function for a structure or model that is collectively revealed by both labelled and un-labelled data. The Literature shows a number of semi-supervised learning that depend and differ by the effectiveness of the classification function. Zhou et al. [245] presented an algorithm to smoothly label the un-labelled points. According to them, the key to semi-supervised learning is the prior assumption of consistency, which could be either local or global in scope. The former tells that the nearby points are likely to have the same label, while the latter means that points in same structure such as a cluster typically refers to the same label. They adopted the first approach, where each point iteratively forwards its local information neighbours until a global state is achieved. They evaluated their approach using a number of classification problems and demonstrated the effectiveness of un-labelled data in learning process.

Diagnosis is another problem that is computation intensive and local consistency is used to reduce the computation overhead. John et al. [123] presented a mechanism that uses local diagnosis for discrete event system modelled using automata. However, in local consistency, the views need to be consistent with each other.

According to them, local consistency does not ensure global consistency and the methods to avoid global computations do not scale well. However, they argue that the complexity of the algorithms drop when tree structures are used that help in achieving global consistency. They therefore presented a junction tree structure to ensure global consistency where the connections between components form a tree which is similar to the techniques exploited by the current hierarchical approaches.

Based on the above discussion, we believe that if only local causality is maintained and no arbitrary events are allowed such as in constraint networks, the system state is always consistent with an appropriate level of local consistency. Most of the activities in virtual worlds are affected by local events, but there exist a number of activities that might be initiated from an arbitrary location and local consistency is unable to maintain global consistency in a virtual world. However, we believe that a global strategy is not required for virtual worlds that is computationally intensive and therefore other adaptive methods are required to cope with the events generated from arbitrary locations for obtaining global consistency.
We believe that an extended adaptive technique has the potential to obtain a coherent state of the virtual world by adding the additional federate(s) involved in an activity between federates far apart from each other in the adjacency list that maintains the neighbouring federates. In the next section, we first explore the level of consistency and the limitations of our approach followed by a possible solution to overcome these issues.

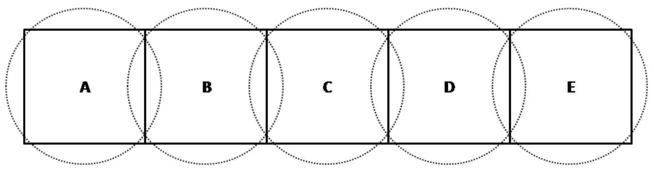


Figure 4.10: Illustrating the neighbouring regions in a 1-dimensional grid.

To illustrate the time advance and the causality management of our system, we use Figures 4.10-4.14. The following description is applied to these figures presented in a two dimensional space showing the time information of different federates at different simulation steps called time steps. Labels A, B, C, D and E on the *Federates* axis are representing the federates in a 1-dimensional grid of regions that are shown in Figure 4.10. The *Time Steps* axis represents valuable simulation steps but they are not covering each and every point in simulation. Each row labelled with a name (of a federate) represents the current time during the simulation at different steps during the simulation. Each circle provides the current time during the corresponding simulation step and is obtained by determining the minimum of the LBTS values of the neighbouring federates. The LBTS value is the current time plus the Lookahead value which is assumed to be a static value of 3 for these illustrations. An arrow is representing the messages sent to the adjacent federates carrying the LBTS values and are sent only when the current LBTS value is increasing. The solid arrows show that the messages are received in time while the dashed arrows represent delays in messages delivery. The dashed arrows are then forwarded with solid arrows to the time steps where they are received and processed.

4.4.2 Consistent Virtual Worlds: Examining our Current Method

Our current consistency approach advances the local time of each federate only with respect to the neighbouring federates and achieves a consistent state of the whole world. It has no inconsistencies because it does not allow any arbitrary events. Theoretically, the system also progresses in a timely manner as shown in Figure 4.11 if no delays are introduced in delivering the messages. It not only obtains a consistent view of the system with respect to the neighbouring federates but also to the non-neighbouring federates. A message generated at any federate for any other federate in the system is received in future and the whole system is globally consistent.

In practice, however, this system like most of the network systems also suffers from network delays. Due to maintaining the temporal order with respect to only the neighbouring federates, it is unable to stop progressing the time advance of non-neighbouring federates immediately when a federate is unable to advance its

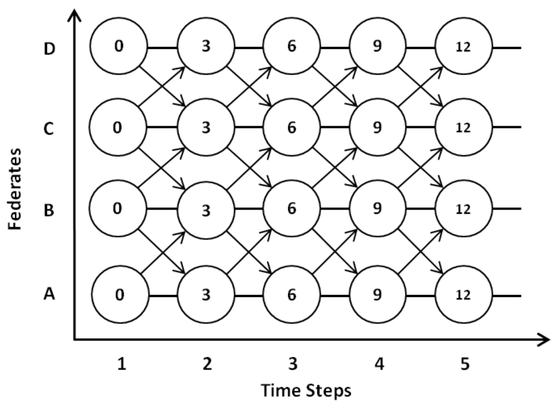


Figure 4.11: Time advance (theoretical) using our decentralised time advance mechanism with no delays.

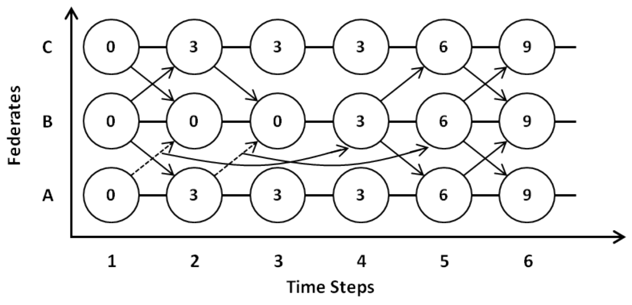


Figure 4.12: Time advance using our decentralised time advance mechanism with delays for a world of 3 federates.

time due to certain delays. However, the system propagates this blockage towards non-neighbouring federates through neighbouring federates iteratively. The federates at long distance from each other update their time information and process their local events independently but suffer from the propagation of delays. However, it is demonstrated that the system does not lead to a global lockup at once and it has the potential to resolve these situations quickly. These situations are explained with the help of Figures 4.12-4.14 for the time advance mechanism using environments having three, four and five federates correspondingly. The basic aim of different scenarios was to see how the blockage propagates and up to which level the local consistency allows federates far apart from each other to proceed independently in time. Figure 4.12 is unable to show the time advance (for both the federates named A, and C) more than once due to the fact that they are directly adjacent to federate B which is unable to update its current time. It is clear that both A and C update their time to three at time step two and then wait until step five, where they update their current time to six after resolving the blockage for federate B at step four. Federate D in Figure 4.13 and federate E in Figure 4.14 advances their corresponding time twice and thrice correspondingly before they enter into blocking state with respect to federate B in both cases. It is interesting to note that federate D and E are at distance two and three levels from federate B that is unable to update its time. A federate at further distance has more independence in advancing its time which equals the distance between the federates. However, the system has potential to resolve these situations quickly, but requires the system to propagate the updated information iteratively as shown in Figures 4.12-4.14.

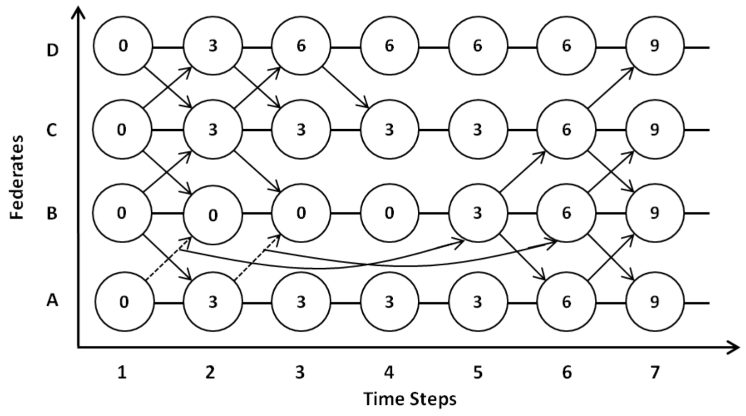


Figure 4.13: Time advance using our decentralised time advance mechanism with delays for a world of 4 federates.

Since, in our current method, each federate synchronises itself with neighbouring federates, it maintains global consistency for the activities having only a local effect. However, Figure 4.15 provides a few examples showing that it violates the causality for applications allowing arbitrary events. Figure 4.15 is an extended form of Figure 4.14 and use an additional type of arrows (coloured blue) to represent different events. It presents three cases to illustrate the potential violation: federate B sending events to federate D during time steps three to five; federate A

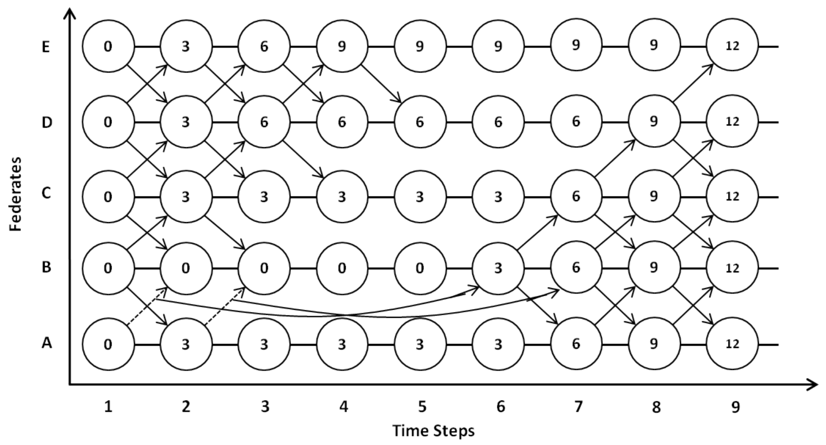


Figure 4.14: Time advance using our decentralised time advance mechanism with delays for a world of 5 federates.

sending events to federate E during time steps four to six; and federate C sending events to federate E during time steps four to six.

Example 1: The following steps illustrates the first case:

- An event generated at federate B at time steps three, four or five is assigned a timestamp three (current time which is zero plus the lookahead value 3) and is sent at the same time step.
- The recipient federate D receives the event in a successive time step which means that an event is received at time step four if it was generated at time step three.
- The current time of federate D is six in all the three cases and it is clear that these events arrive in past showing the violation of the causality constraint.

Example 2: The following steps explains the second case:

- Three events are generated and sent by federate A at time steps four, five and six for federate E with a same timestamp value of six.
- Federate E receives these events at time steps five, six and seven correspondingly.

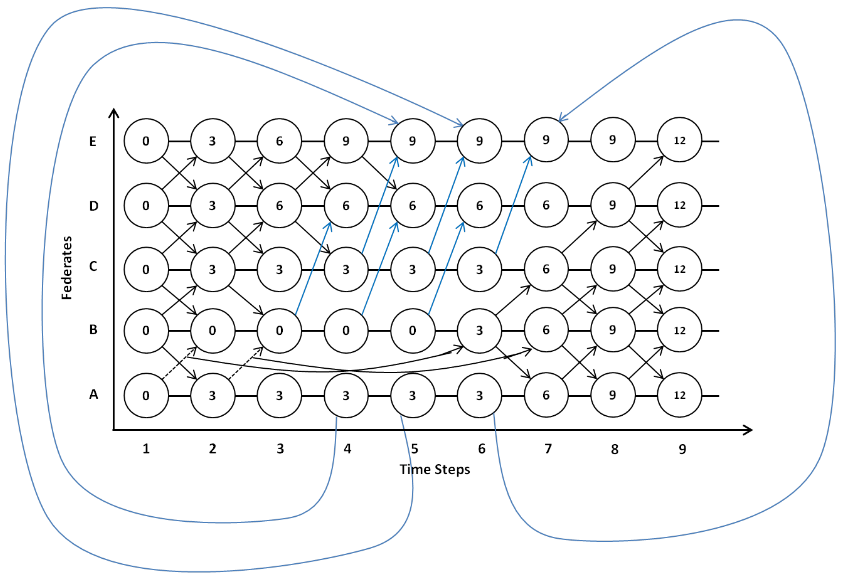


Figure 4.15: Illustrating the violation of causality for activities spanning arbitrary locations.

- The current time at federate E is nine during these time steps and it shows that the events are violating the causality constraint.

Case 3 is similar to Case 2 and is therefore not further explained. Based on these facts, we suggest that an extension to the current method is required for accommodating the additional applications (such as virtual phone calls) that has causal effects beyond the adjacent federates. However, we believe that a global strategy is not a viable solution especially for a very scalable system due to the potential computation overhead. In the next section, we suggest a dynamic adaptive method to overcome these issues.

4.4.3 Possible Extension to our Consistency Method

The overall activities in a virtual world can be classified in to three different categories: activities based on locality that have an effect on activities in the neighbouring regions only; activities that pass through the neighbouring regions towards non-neighbouring regions such as aeroplane or an avatar flying through a space; and activities that have an effect on activities in an arbitrary federate such as a virtual phone call or an avatar teleporting from one region to an arbitrary region.

The first category covers the majority of the applications, and are managed well by our current consistency mechanism giving a global consistent state of the worlds. The activities in second category are also managed by our approach, as the objects have only causal effects towards neighbouring regions. The concept of presence of moving objects in the neighbouring regions is exploited to synchronise the object movements with the local and adjacent federates thus always giving a global consistent state.
To include the activities in third category, we need to extend our current approach. Since, local approach is unable to maintain global consistency, and global strategy is very expensive in terms of computation, communication, and implementation overheads, we propose a dynamic adaptive strategy to accommodate these applications. We can easily extend the current consistency approach by allowing each federate to include the other federate involved in an activity in this category to its list of neighbouring federates. Similarly, these federates are removed from the adjacency lists when the activity is over such as a successful termination of a telephone call or a teleport. The component of our method dealing with the neighbouring federates is highly dynamic that reacts to both split and merge operations. However, due to the potential differences in local times, there might be a delay until the slower federate in time reaches a synchronised state with a federate faster in time. We believe that this extension has the potential to achieve the required level of consistency that ensures global consistency for virtual worlds.

In future work, we intend to perform a number of different experiments to evaluate the correctness of the existing and extended infrastructures.

4.5 Conclusions and Future Directions

This chapter presented a simple, but flexible, decentralised synchronisation infrastructure and illustrated it with the help of both simple flat and hierarchical scenarios. It is scalable and allows a federation to take independent decisions with distributed control among federates for direct consultation with the interacting

federates. It depends on the realistic assumption that events can only affect a finite and known set of adjacent regions. Simulation results supported our claim that it achieves the correct temporal ordering for randomly generated events. Furthermore, an abstract model is used to compare it with traditional approaches. It is clear that the number of hops, and thus complexity and delay in existing mechanisms, rise significantly with an increase in the levels of a hierarchy. However, these parameters have no impact on the proposed approach.
Further simulations are required to verify that the proposed system works with and have potential to quickly resolve the temporarily blocking states and delays compared with the traditional hierarchical infrastructures. Further simulation and implementation of this work is our future work.

Chapter 5

OpenSimulator: State-of-the-Art and Proposed Extension

This chapter examines the features and current architecture of an open source VW development framework called OpenSimulator (OS) [177] with reference to Second Life (SL) architectures. The key architectural limitations in OS architecture are highlighted and an extended architecture is proposed to cope with these limitations. The basic purpose of this extension is to make the OS scalable by introducing dynamic and fair distribution of load based on spatial partitioning of a virtual space. It also incorporates a consistency model based on a constrained communication model. Modules incorporating these functionalities are explained in terms of our previous work [61, 63]. It illustrates the traditional spatial partitioning methods for adding a new Simulator (Sim), followed by an introduction to our contemporary approach to overcome the limitations of traditional methods. This chapter is based on our previously published work [67] on this topic.

5.1 Background

Second Life (SL) is the state of the art in VWs and has gained much attention from end users. Due to its tremendous popularity and growth rate, the basic architecture started experiencing instability issues right from its inception due to central storage of data. To avoid these issues, the Architecture Working Group

(AWG) [196] at Linden Lab was given the task of designing a new architecture and protocols to open up the SL Grid (SLG) to allow others to run parts of the grid. It scales well in terms of content by adding more Sims, and it significantly reduces messaging overhead [189, 195]. However, it suffers from both over-provisioning and under-provisioning of resources because it statically assigns parts of the world to different resources and lacks dynamic capabilities to manage resources. The major issues with SL include increased latency and client crashes. Inventory loss is an even more disturbing fact that happens without any warning, where an inventory [118] is the collection of all the items a user owns or have access to. Similarly, it does not incorporate the conservative TM approach.

SL was the first option to work with for the prototype development of the current work; however, only the source code of client software is available for further development. Since the work in this thesis deals with technical aspects of underlying infrastructures, OS being a complete and independent open source implementation of the AWG work is used to develop a prototype. It has similar functionality and cope with the limitations of SL. Like SL, OS has no capabilities to scale dynamically, though it allows an arbitrary number of regions to be hosted by a single simulator process. However, its modular structure can easily be exploited to incorporate these features. It allows others to host parts of the world and integrate with a global grid. Similarly, it also allows organisations and individuals to host their own private grids, and it is supported on both Windows and Linux platforms.

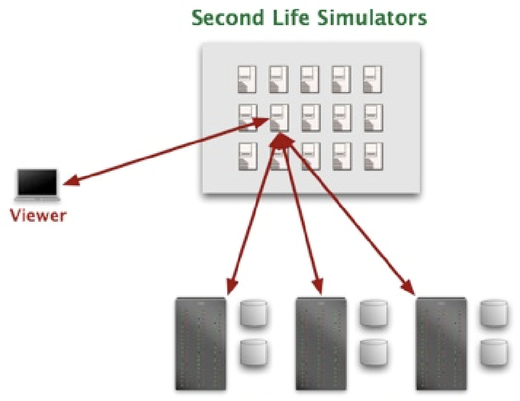
In this section, therefore, SL architectures and their components are explored to illustrate the way a viewer interacts with either a SL or OS world, before exploring and extending the OS architecture.

5.1.1 Second Life (SL)

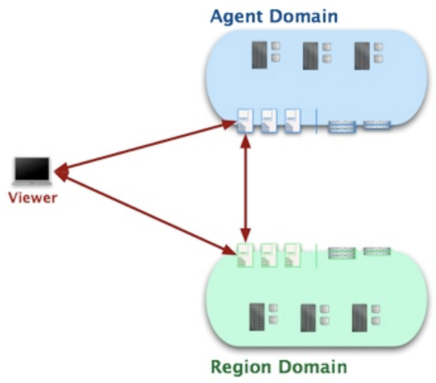
SL is an environment that allows individuals and organisations to develop their content in a 3D format over a purchased piece of land. SL is a MUVE which is not strictly a game because it lacks pre-defined rules. It is a popular virtual space for meeting friends, doing business, and sharing knowledge [201]. Currently, many applications are deployed in SL, broadly belonging to Education, Arts, Science, Work Solutions, Religion, Embassies, Competitive Environments, and Relationships [137]. Every activity is regulated by an evolving framework laid down by Linden Lab [95]. IBM has been working together with Linden Lab to achieve the goals of 3D web by extending the current web to incorporate a 3D experience. The whole space is divided into named 256m x 256m (65,536 m^2) areas called regions. Each region is hosted by a single Sim process on a dedicated core of a multi core server; however, each Sim can possibly host multiple regions (up to a maximum of 4). Each server runs scripts as well as providing communication between avatars and objects in a region [137]. A script [194] adds behaviour to an object and an avatar [10] is a 3D character representing a user in a VW. Each region can handle up to 100 avatars and 15,000 primitives (prims) [195]. A prim is a single part object, such as a box or cylinder, which is used to create multipart objects in VWs [182]. These regions are combined into estates based on a particular common set of rules, such as banned users and sun position. An estate is a group of private regions belonging to one resident. Each region on a grid must be part of the SL mainland (the Linden-designed continents) or privately owned estates on servers operated by Linden Lab [195]. There are many companies that sell/rent private estate land in SL which range from individual residents renting a single parcel to major companies with dozens of privately owned Sims [187]. A parcel is a divided part of a region which could be as small as $16m^2$ (4m×4m) and as big as the entire region [195]. SLG is the underlying infrastructure hosting SL. It refers to an integrated system that provides a networked collection of servers arranged in the form of a rectangular mesh. Some of these servers run Sims representing the land while others manage different independent, but integrated, services including presence, inventory management, and asset store. SLG enables users to create content and communicate, collaborate and engage in communal services [121, 195]. It restricts the connections of each server to up to four neighbouring servers. Linden Lab runs several grids for internal and external testing [195].

5.1.2 The Current Architecture

The current architecture of SLG is shown in Figure 5.1(a) [199]. It shows that a viewer interacts with a Sim that is hosted on a server, being part of the grid simulating a square region. Each Sim interacts with a centralised database and acts as a proxy that tracks the movements of avatars [199].



(a)



(b)

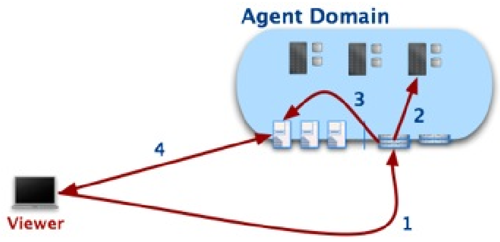
Figure 5.1: The SLG architectures with an interactive client [199]: (a) the existing architecture; (b) the extended architecture.

The huge popularity of SL resulted in frequent system crashes due to the use of a central database. Similarly, the current architecture is not capable of sustaining itself against the initially envisioned statistics [197, 198]. Furthermore, to make the VWs as ubiquitous as email and the web, the AWG at Linden Lab was given the task of extending the current architecture of SLG [196]. The basic aim is to develop protocols that will open up the SLG to others to run parts of the grid, including an open grid protocol for interoperability of these virtual spaces [167].

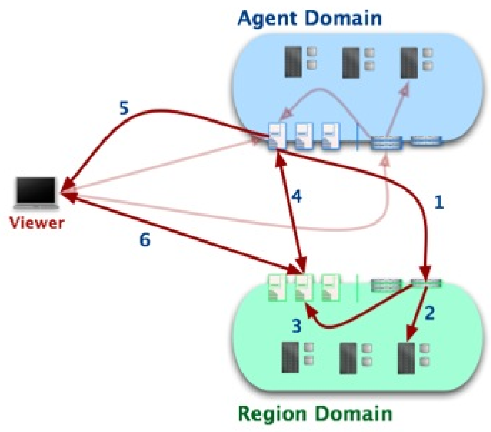
5.1.3 The Extended Architecture

AWG proposed an extended architecture comprised of two domains: an Agent Domain (AD) and a Region Domain (RD). The AD handles agents and the RD manages regions. An agent is the internal representation of a viewer [53]. These domains work together to realise system functionality [199]. A successful login and operation requires the viewer to communicate with both domains, as shown in Figure 5.1(b). The login process is illustrated in detail with the help of Figure 5.2(a) and Figure 5.2(b).

The AD manages user login and stores profile and inventory data. It comprises three components called services, hosts and stores which are reached by arrows named 1, 3, and 2 correspondingly, as shown in Figure 5.2(a). Agent services are instantiated when required and handle stateless information such as profiles [54]. This information can be cached and publicly accessed through web services [54]. Agent hosts handle logged-in agents and their sessions. The information it manages includes avatar location, and the status of an agent and its friends [54]. Agent stores maintain the actual data in databases such as inventory and profile data [54]. These components can be used redundantly as required.



(a)



(b)

Figure 5.2: Process of a viewer login to: (a) an Agent Domain (AD) [54]; (b) a Region Domain (RD) [163].

The RD deals with regions and follows the same architecture as an AD [163]. It is also comprised of services, hosts, and stores that are also reached by arrows named 1, 3, and 2 correspondingly, as shown in Figure 5.2(b). Region services handle stateless public information about regions, such as avatar positions, and object and parcel information. Region hosts are the servers (Sims) managing parts of the virtual space. They provide in-world avatars and objects interaction while taking data from the region stores. A region is unavailable when its host is down [163]. Region stores retain the actual regional data about objects, regions and parcels [163].

The login process takes the following steps to login a client to an AD (see Figure 5.2(a)) [54]:

1. the viewer sends login information to the agent service;
2. the agent service queries agent store for validation;
3. the agent service asks an agent host to initiate a session and finally;
4. the agent host establishes a connection with the viewer.

To complete the login process and obtain region data, the following steps login a client to an RD (see Figure 5.2(b)) [163]:

1. the agent host contacts the region service;
2. the region service queries region store to obtain region information;
3. the region service contacts region host and asks it to establish a session for a new agent;
4. the agent host directly contacts region host to obtain the avatar;
5. the agent host returns region host to the viewer for onward direct communication;
6. the viewer now talks to the region host.

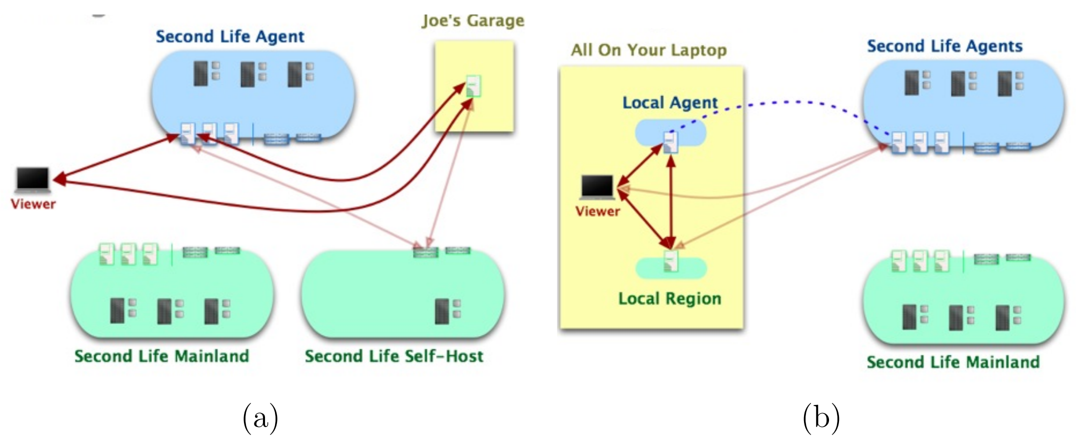


Figure 5.3: Additional scenarios based on the extended grid architecture for [200]: (a) home content as part of world content; (b) off line content.

The extended architecture distributes the activities. However, it keeps a number of central utilities that are globally required such as identity, topology, currency, and search [202]. The global identity enables a user to be unique across all ADs, but it requires different levels of verification and authentication. The topology identifies connected regions, a kind of DNS for the metaverse. Since, the basic mission was to open up the SLG to others, the extended architecture thus provides a number of additional implementations. Figure 5.3(a) provides such a scenario showing an individual home region (own Sim) integrated with the SL regions [200]. Similarly, it allows users to run a completely disconnected region but it requires the existence of both agent and region domains on a local system (see Figure 5.3(b) for this scenario [200]). Figure 5.4 shows how the SL infrastructure looks finally with the extended architecture [199].

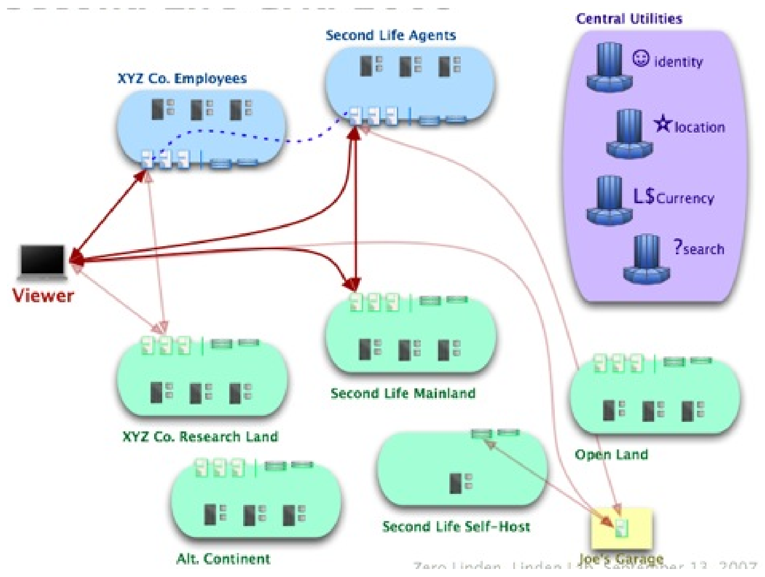


Figure 5.4: Complete extended look of the SLG infrastructure, if implemented as planned [199].

5.2 OS and its Current Architecture

OS is an open source multi-user 3D application development framework that simulates virtual spaces similar to SL. It supports the messaging protocol of SL, but

it does not support its game specific features. It is written in C# which uses the .NET framework on Windows, and Mono framework on Linux machines. It is currently pursuing innovative directions to become an extensible infrastructure for the 3D Web, and it supports an arbitrary number of regions per Sim (server) [177]. Five major services are required to provide interaction between a region and viewers based on the original design of Linden Lab network. These services are called User, Grid, Asset, Inventory, and Messaging (UGAIM) services and each has a vital role in the OS framework. Each region must be known to only one instance of each service [176].
UserServer is responsible for user authentication to the grid. It assigns a Universal Unique IDentifier (UUID) as a session identifier to a client that is used globally over the grid [176].
GridServer is responsible for authenticating regions to the grid. It gives a UUID to a region. Since all the regions belongs to a global 2D grid, each region is assigned a particular X and Y position [176].
AssetServer describes the items (with static nature) including sounds, textures, images, notecards, and scripts that are used by the users and organisations to develop their content. It is a database based on the principle of 'write few and read more'. The current implementation restricts modifications to assets, and they are immutable [8, 176].
InventoryServer is a database server that keeps track of the placements of assets by linking the UUIDs of users to their InventoryRoot folders. The InventoryRoot folder maintains a list of UUIDs for folders, and type and descriptive names for the assets. It also manages the associated permissions of the assets [176].
MessagingServer is used for in-world communication among people and to keep track of who can listen to conversations. It manages long distance messaging and keeps unread messages until they are read [176].

A region with a scene is the most important component of the OS framework that must be part of an estate similar to SL. A scene [192] is the representation of the content of a region. It runs physics and scripts, and keeps track of objects in a scene, and the observers connected to a scene. It provides scene updates to the observers.

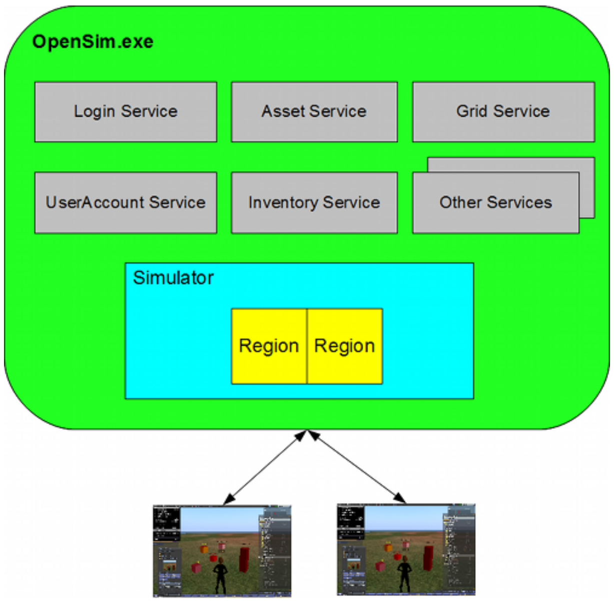


Figure 5.5: The OS architecture for standalone mode [172].

Primarily, OS has two modes of operations: Standalone mode and Grid mode. It allows a Sim to run an arbitrary number of regions in both standalone and grid modes. The only difference is where they get their UGAIM services from. In standalone mode, a region provides its own UGAIM interfaces and runs them in a single process. However, in grid mode each service is run as a separate process that could possibly be on a different machine. The UGAIM servers are all configured as centralised grid services [176].

In standalone mode, both region Sim and all data services run as a single process called OpenSim.exe. The abstract architecture of OS for standalone mode is given in Figure 5.5. In this case, a region server (Sim) is hosting two regions but it has the capability to run an arbitrary number of regions on a single machine. Clients are connected to the same process in standalone mode [172].

In grid mode, the UGAIM services are separated from the region server process and implemented as a separate process called Robust.exe. The grid mode architecture is presented in Figure 5.6. The data services can all run as a single Robust.exe instance but they can be split and run on entirely separate machines for improved performance. The OS instance (OpenSim.exe) is now only a region

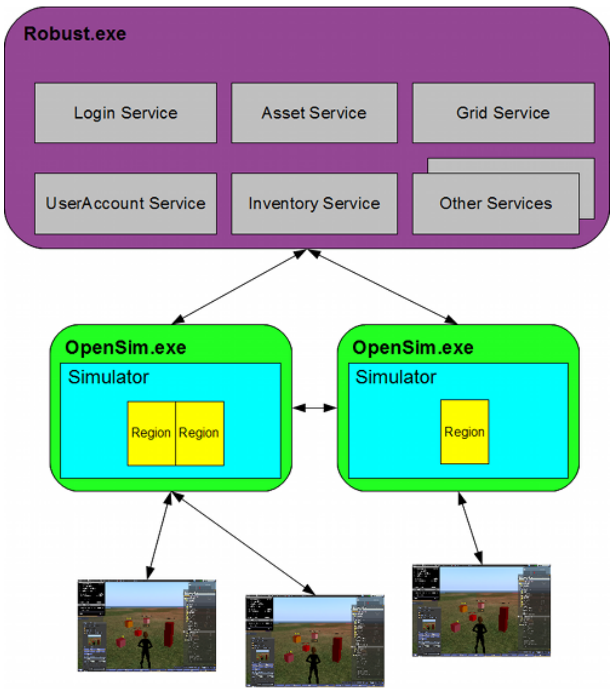


Figure 5.6: The OS architecture for grid mode [172].

server that can host an arbitrary number of regions and communicates with separate data services. It allows several instances of region Sims to be run on different machines [172]. The regions simulated by different instances are known to each other because they are controlled by a centralised grid service.

The user login in grid mode requires access to the user service that authenticates a client and then directs it to connect to a region on a simulator based on its previous location. The login service uses the IP address of a region Sim in standalone mode, but it uses the IP address of the host running UGAIM services (Robust.exe) in grid mode. In case no previous location is found, it is sent to a default region. When a user connects to a region, its avatar is added to the scene (described as a root agent) and the neighbouring regions are informed about the user [172]. In response, each neighbouring region adds a presence for the user (described as a child agent) to its scene. This allows the avatars to move smoothly to the neighbouring regions. When an avatar crosses a boundary, the state of the avatar in the current and previous regions is swapped, and a presence is added to the new neighbouring region while a region which is no longer a neighbouring region deletes its presence.

OS allows different database engines, such as SQLite, MySQL and MSSQL, with varying degrees of functionality and different storage orientations. SQLite comes bundled with OS (by default) and requires no extra configuration. However, it is not for production use or running the OS in grid mode. MySQL is fully supported and could be used with both centralised and localised scenarios. MSSQL is partially supported. In grid mode, OS uses databases at two levels to manage data associated with the grid and region Sim processes. A centralised database is normally used to manage UGAIM services but they can be managed through separate independent servers. Each region Sim uses a separate database that manages data for all regions of this instance.

OS architecture is very flexible and scales well in terms of content by just configuring new Sim instances. However, it has no capability to scale well dynamically in terms of concurrent users without re-configuring the Sims. Similarly, it has no capability to directly manage applications with conservative nature. Based on these issues, we feel that an extension to the existing architecture is required to add new simulators when load is increased, and reduce them to a minimum level when the load is decreased. Recently, OS has incorporated a number of features that can be used to improve scalability and load distribution. We believe that the flexible and modular structure can be easily exploited to incorporate conservative applications to VWs.

5.3 Interesting Features of OS

5.3.1 RemoteAdmin (RAd) Functionality

The RAd functionality uses a library for remote procedure calls (RPCs) builds on XML [242] and HTTP [100] (called XML-RPC [133]) to generate a request to be processed on a remote computer. It provides a number of methods to implement the most common OS console commands, such as ‘create region’ and ‘save or load the OAR files'. It has a number of flavours and works well with a number of different platforms including C#, .NET, Python, PHP, and Perl. A number of examples showing the use of RemoteAdmin methods and their requirements are available at [186].

5.3.2 OpenSim Archive (OAR) Functionality

The OAR functionality is capable of storing the entire asset data of a region. It can be used to load data on a completely different system using a different asset database. It fully restores the terrain, region parcel data, the textures of objects, and their inventories. OS provides two console commands to save and load OAR files, which are also exposed through the RAd functionality for remote processing. The default load option replaces the existing objects with the content of an OAR file. However, if the merge option is used, then the OAR content is merged with the existing objects in a region. The basic use case of the OAR functionality is to share the content of entire region with others. However, its performance is not very good, especially when used with very large archives [173].

5.3.3 Megaregions

The region is the most important component of the OS framework, and a Sim can run an arbitrary number of regions. The Standard region size (a square space of 256m×256m) is small, and although a Sim can run multiple regions, it requires complex border crossings between the regions. Furthermore, there were demands for larger regions from OS developers and users. To resolve these issues, the concept of megaregions was recently introduced to OS. It converts a number of regions into a larger megaregion allowing the avatars to cross regional borders seamlessly and transparently within the same physical server. However, this concept is not mature and has a number of limitations. Currently, it uses a configuration file with special arrangements to set up a megaregion and dynamic construction is not possible. The conversion of already existing regions into a megaregion transfers all the contents into a single root region. However, regional data could be stored and later on restored (possibly using the OAR functionality) to overcome some of these limitations [106].

5.4 Related Projects

This section briefly introduces the previous and current work undertaken using the OS framework for scalability and load distribution issues.

5.4.1 Load Balancer Project

Load balancing and scalability issues in OS were addressed using a project called Load Balancer which is no longer maintained and is not part of the OS framework [141]. Load balancing was achieved by re-assigning regions from an overloaded server to a less overloaded server dynamically, without re-starting the Sims. To achieve this goal, it was serialising a region and creating a clone of the region on a target server using the same stream. It then destroyed the original region after telling client viewers about the address of the cloned region. It used the concept of sharding (replication of regions) to scale the number of interacting users. Each shard was responsible for updating a fraction of avatars and send state updates to other sub-regions. Though it manages to send state updates, we believe that it physically lacks the concept of meeting people face to face, which is the basic theme of VWs.

5.4.2 ScienceSim

ScienceSim is a virtual environment that can be used as a tool for collaboration, visualisation and experimentation. Intel is currently developing the hardware and simulation infrastructure behind it. ScienceSim uses the modular components of the OS framework and tries to leverage many Internet standards and technologies to provide an integrated set of technologies for building applications. The most important contribution of Intel to ScienceSim is the enhancement of the OS code for performance and scalability. Intel uses the basic OS framework but improves performance by fine-tuning different areas of its code, and they have contributed a number of patches to OS. The number of objects in a scene is increased by a factor of ten, and the memory footprints are decreased significantly. Message processing and inner loop structures are made more efficient, and a number of core data structures and operations are replaced with more efficient approaches. A VW world architecture based on the concept of distributed scene graph compared with the traditional simulator centric architecture is presented for scalability with an additional layer for communication. According to Liu et al. [140], load balancing is the adjustment of scene partitions to the servers, and the assignment of players to appropriate servers. Due to Intel work on performance, OS can now support more objects and participants than the existing VWs [20, 107]. However, their approach is to target a completely different aspect (scene graph) of the system compared with the current work, which is based on the concept of spatial partitioning.

5.5 A Proposed Extension to the OS Architecture

OS has two modes of operation as well as two different architectures that differ in the ways UGAIM services are implemented and accessed. We propose extensions to the abstract architectures of OS based on our work that are presented in Figure 5.7 and Figure 5.8, for standalone and grid modes respectively. Two additional components named Load Distribution (Fair) and Time Management are introduced in both architectures. Load Distribution is marked fair representing the fact that ARA algorithm distributes the load as balance as possible while maintaining the continuity constraint. It does not propose any changes to the existing components and basic structure but, instead, uses the modular components to achieve the goals. This work can be applied to both an environment with a single parent Sim, or to a Sim as part of a grid. However, it is assumed that each starts with a bigger space (shown as a megaregion that has four regions) based on an arbitrary number of basic OS regions, regardless of architecture. Both architectures implement the same modules to dynamically increase the capacity of a Sim and achieve a consistent state of the whole space using a restricted communication model. The only difference in both standalone and grid architectures is the way a client accesses the UGAIM services, as shown in Figure 5.7 and Figure 5.8.

The Load Distribution (Fair) component continuously monitors the load and uses the following functions to achieve scalability and implement communication among

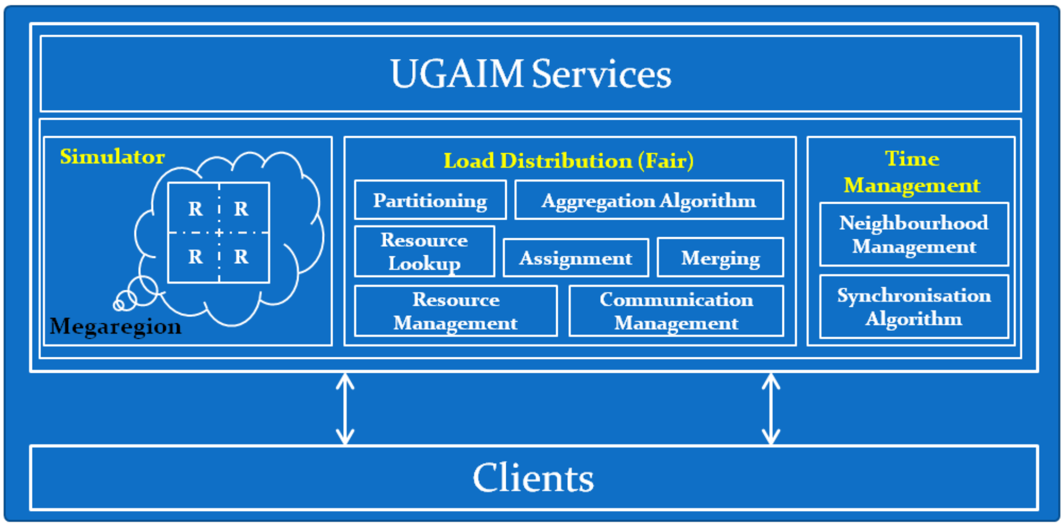


Figure 5.7: The proposed extended architecture for standalone mode of OpenSimulator (OS).

the servers.
The Partitioning algorithm splits a megaregion into a number of sub-regions, based on the proposed strategies presented in section 3.1 of chapter 3, against player distribution.
The Aggregation Algorithm (ARA) determines two contiguous larger spaces taking input from the Partitioning algorithm in the form of a 2D grid. It uses the aggregation strategies presented in section 3.2 of chapter 3 to obtain valid spaces.
The Resource Lookup module maintains a pool of resources and selects a resource for sharing the load with the overloaded server.
The Assignment function delegates an aggregate (a continuous larger space) to a newly added server. It takes the output of the aggregate algorithm as an input and transfers the sub-regions in turn for improved performance.
The Resource Management function manages the RMT and helps the communication management module to determine the location of regions and avatars in a set of servers sharing a simulator's load.
Communication Management manages communication activities between different servers jointly simulating a megaregion. It holds the messages in case of a split operation and forwards them all together to the avatars when they are

resumed at the destination server.

The Merging process implements the reverse process to splitting which can be initiated by either a parent or a child server. It considers the physical boundaries and merges with a server only if it merges with the valid adjacent regions to the region it serves.

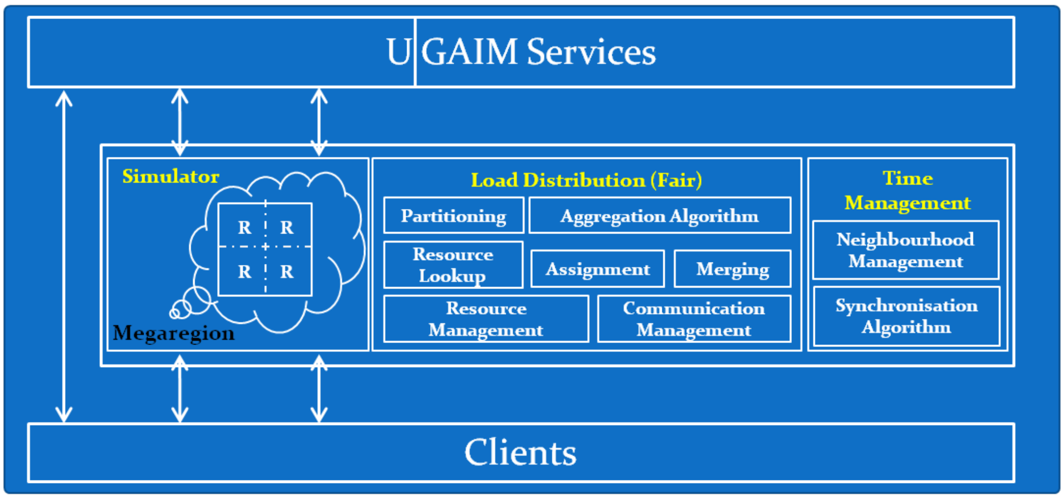


Figure 5.8: The proposed extended architecture for grid mode of Open-Simulator (OS).

The Time Management component uses a P2P approach together with a constrained communication model to implement consistency. It is comprised of two processes: Neighbourhood Management and Synchronisation Algorithm.

The Neighbourhood Management manages those regions that share physical boundaries with the region(s) that a given server is simulating. This is a dynamic activity and, based on an increase and decrease in load, the number of regions might change over time. Its basic goal is to determine the regions whose time information must be considered by our synchronisation algorithm (see chapter 4) to maintain a global consistent state of the space.

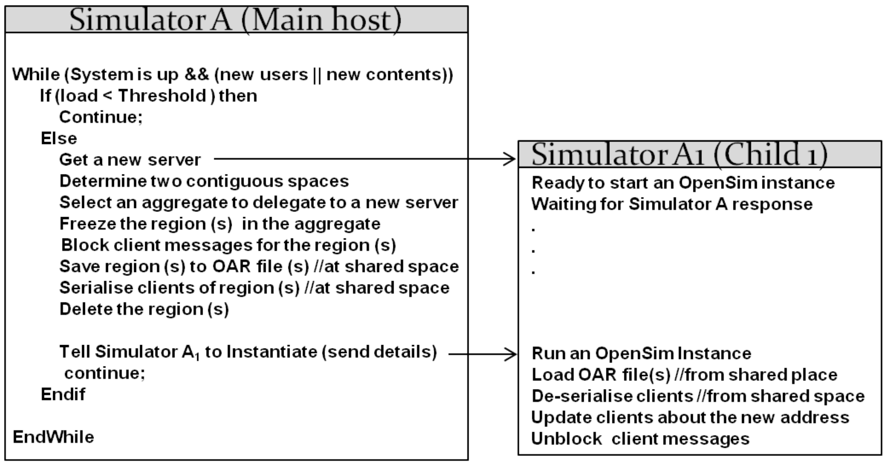
The Synchronisation Algorithm is continuously running at every server simulating part of the VW and updating its simulation clock by using the time information of neighbouring servers. It is also responsible for sending its time information to help others to update their clocks, thus potentially maintaining a consistent space. Further detail on consistency is presented in chapter 4.

5.6 Illustration and Discussion

This section discusses the basic traditional steps used to add a new instance of OS and share the load with it. Since the basic aim of this work is to achieve scalable worlds with an emphasis on reducing resource under-utilisation, we start with a single server simulating a bigger space, possibly based on an arbitrary number of regions. This space offers a reasonable amount of content and is normally not over-populated with players. However, with the passage of time, users join in the world and ultimately, at a certain point, it exceeds the SplitCapacity. In this case, the system initiates a split and shares the load with another server.

5.6.1 Traditional Steps in Spatial Partitioning

A generic and possible set of steps required to instantiate an additional instance of OS and run part of the system on a child node is presented in Figure 5.9.



```
While (System is up && (new users || new contents))
    If (load < Threshold ) then
        Continue;
    Else
        Get a new server
        Determine two contiguous spaces
        Select an aggregate to delegate to a new server
        Freeze the region (s) in the aggregate
        Block client messages for the region (s)
        Save region (s) to OAR file (s) //at shared space
        Serialise clients of region (s) //at shared space
        Delete the region (s)

        Tell Simulator A1 to Instantiate (send details)
         continue;
    Endif

EndWhile
```

```
Ready to start an OpenSim instance
Waiting for Simulator A response
.
.
.

Run an OpenSim Instance
Load OAR file(s) //from shared place
De-serialise clients //from shared space
Update clients about the new address
Unblock client messages
```

Figure 5.9: Steps in traditional spatial partitioning methods to achieve a new Simulator instance and distribute the load with it.

Initially, SimulatorA is the only server simulating the whole space and it continuously checks the load against the number of users and content. At a certain point, when the overall system load exceeds a threshold (SplitCapacity), SimulatorA initiates the process to get a new server (Simulator A_1 for example) to share the load with. It determines two contiguous aggregated spaces after splitting the original space, and selects one of the aggregates to delegate to the new Sim. SimulatorA freezes the region(s) in the selected aggregate and holds (blocks) the messages for the players in frozen region(s). These messages are maintained and forwarded to the server simulating the concerned players when a transfer is over. System uses the OAR functionality to store the contents of region(s) in a shared location. Similarly, it serialises the clients for the region(s) to a shared location. When the complete information is stored, the process removes these regions from SimulatorA and tells SimulatorA_1 to run an OS instance with the same specifications of region(s). The reason for using the same specification is that the users should get the same neighbouring regions. Regions are maintained by grid services that use a unique X, Y location for each region. In response, SimulatorA_1 instantiates an instance of OS and loads the contents from OAR file(s). Clients are deserialised and they are told to use the new Sim (giving the address of SimulatorA_1 and the port it is listening to). The system forwards the blocked messages to the regions which, in turn, hand them over to the clients. The distribution process is completed and the space is now managed by two different Sims. Each Sim now independently monitors its load and could share the load again with a new Sim if it crosses the SplitCapacity.

5.6.2 Our Contemporary Approach to Spatial Partitioning

The main problem with transferring all the regions at once from a Sim and running them on a new Sim is that it takes significant time and, therefore, the user has a negative experience of the system. An alternative method would instead transfer one region at a time, thus considerably reducing the time a given user suffers from a transfer. However, starting an OS instance also takes a considerable amount of time, though it might start with one region. Therefore, a more efficient method could start an OS instance with a dummy region in parallel with other activities, such as creating the OAR file(s) and client serialisation. Regions with actual streams are then remotely generated, which takes less time and thus reduces the total time a user suffers from this process.

We believe that freezing the clients for few minutes in traditional spatial partitioning techniques gives a very negative impression of the overall system. Therefore, we used the concept of a transit region that gets two fold benefits. It could be used instead of a dummy region to start a Sim in advance, so there is no need to wait until a region is removed. It also gives users a fair choice to move to some other part of the grid or keep themselves busy with some simple activities until the original setup is resumed.

5.7 Conclusions

In this chapter, the current and extended architectures for SL were examined as a reference. It proposed an extension that works with both OS architectures to incorporate the features for dynamic scalability as well as consistency. The current architecture of OS is explained, and its interesting features and related projects are explored in detail. The modules incorporating these features are described and the traditional abstract set of steps are used to illustrate how a system scales in case of excessive load. It concludes with an introduction to the contemporary approach adopted for spatial partitioning in this work.

Chapter 6

Scalable Virtual Worlds: Investigating Opensimulator

This chapter presents the investigation of the operational capabilities and robustness of OpenSimulator (OS). It outlines the operational view of the OS architecture and introduces a number of worlds' content available as OAR files that are used to populate our worlds. It also presents a summary of initial tests for both the capabilities and components that we wanted to use for the implementation of this work. It details an abstract load model based on both static and dynamic content as well as interactive clients that helps in determining when to initiate a split or a merge operation. It also presents the evaluation of different database options and configurations. A scalability model is introduced and a simple grid model is used to implement it by using the RAd functionality for tranferring a region from one Simulator (Sim) to another Sim on both Windows and Linux platforms. It also presents improved strategies to reduce the time taken by different activities during a transfer.

6.1 Introduction

The OS framework is the state-of-the-art in open source VW development frameworks. It is continuously under development and many components are not mature, although the basic functionality works well. Therefore, a thorough study was conducted to check the capabilities and especially the components needed to develop our scalability work. This chapter is devoted to a number of investigative studies mostly based on load and time analysis of different activities during a transfer.

6.1.1 OS Operational View

This section briefly details how the management functions work in both standalone and grid modes of OS.
OS provides different commands to perform different regional and grid management activities. The standalone mode provides all regional and managerial commands to run through a regional console window. However, the grid mode separates them into grid and regional commands where the former are applied from a Robust console and the latter from a region console. These commands use basic routines of the OS framework. RAd functionality is an additional OS component that allows developers to run regional commands on a remote machine which are accessed through Remote Procedure Calls (RPCs). It is used later for a number of activities when transferring a region.

6.1.2 Platforms

This work is investigated and implemented on two different platforms for various reasons including the availability of a few Windows-based systems, limited capabilities of Windows systems, and in order to demonstrate scaling bigger worlds.

Windows

A private network of four Windows systems was used for the experiments. It includes a Pentium dual core system with a combined processor speed of 3.2GHz and 2GB RAM. The other three Pentium IV systems each has a processor speed of 3.2GHz and 1GB RAM. However, they were used in different configurations for different activities.

Linux

A much larger Linux environment (the UEA Cluster) was also used to scale our work. It allowed us to use a large number of nodes (dual quad core 2.66GHz, with 8GB RAM) for running the services, as well as a large number of Bots.

6.1.3 World Content and Interactive Players

Two different types of content are used to populate regions of a world for most of our experiments. These are: dynamic content and example world content. The experiments based on dynamic content use an increasing number of prims that have attached scripts. This workload is called Prims and Active Scripts (PandASs), and it is obtained from ScienceSim project [193]. For our examples, a prim is a square that has an attached active script which rotates it and changes its colour every few seconds. The other examples of dynamic content, which are normally called non-player characters (NPCs), include moving ducks and monsters. NPCs use server side scripts, and are different than Bots in the sense that Bots run scripts from client software. The experiments based on worlds content use regional content obtained from their developers which have different themes and are mostly based on static content and simple scripts. These worlds are called OpenVCE, FairieCastle, EducationSim, Maya Pyramid and CSI World, in addition to the simple world developed for this study (named Our world), and all are briefly introduced in Table 6.1. They are available as OAR files and each is used to populate a region, but they could be duplicated if required. The following is a brief introduction to these worlds:
The **OpenVCE (Open Virtual Collaboration Environment)** is a community project that provides free support facilities for collaborative communities and integrates a community web portal to a VW. It can be retrieved from [168].
The **FairieCastle** is a castle on an island that has a hidden cave and a waterfall behind it. It has scripts that grows trees and mushrooms and can be downloaded from [59].
The **CSI Virtual World** is a VW for solving crimes by testing virtual blood samples and inspecting collected evidence. The archive of this world can be downloaded at [45].

The **Maya Pyramid** is a virtual representation of the Temple of the Inscriptions, a Mayan pyramid in Palenque, Mexico together with large houses and small thatched huts called ‘palapas’. It is available for free download at [148].
The **EducationSim** is a simple space with a classroom, a conference room, a small house and orientation land [174]. The archive can be downloaded from [56].
Our world is a simple world with a small number of prims and is used for simple tests conducted for a variety of purposes.

Virtual World	File Size	Prims/Scripts	Assets/Objects
OpenVCE	21MB	2148/152	385/96
FairieCastle	19.3MB	2680/116	290/194
CSI Virtual World	8.02MB	1256/184	151/307
Maya Pyramid	6MB	1227/2	56/34
EducationSim	2MB	1439/8	22/206
Our world	-	8/-	3/8

Table 6.1: The description of example worlds content.

In order to access and modify the content in a VW, a user needs client software. Since it is impractical to request a large number of online players, we used the concept of Bots to test the behaviour of interactive clients. We used the TestClient (TClient) application (TestClient.exe) of the OpenMetaverse[171] library to login Bots, and we used their basic commands to develop scripts for different activities. OpenMetaverse is a project that comprises a collection of open source building blocks for developing VW platforms [171]. Each TClient has the capability to login a large number of Bots. However, using a single instance for many Bots introduces longer delays as the activities are performed in a sequence. Hence, we used multiple TClients that login ten Bots at the maximum.

6.2 Initial Tests

This section provides a description of number of fundamental tests that were conducted to determine the capabilities and flexibility of the OS framework. The standalone mode has limited capabilities than grid mode. However, we started with standalone mode and then moved to grid mode which is used for the implementation of our work. It was observed that RAd functionality also allowed certain operations between Sims running in the standalone mode. These test targeted the OS design, console and remote administration, and teleporting. Teleporting is the process of instantly changing the position of an avatar. The basic aim was to determine if the RAd and OAR functionalities had the potential to implement the split and merge operations on both current and remote Sim while maintaining system design.

The following valuable tests were conducted with a small amount of content and a single player, and this gave us confidence in building further on this framework. All services and OS instances (for both standalone and grid) used in these experiments were running on a single system. These tests were performed on both Windows and Linux platforms.
First of all, both standalone and grid modes of OS were configured. The SQLite and MySQL options for the standalone mode and MySQL for grid mode were tested.
The basic **Create Region (CR)** functionality was used to create a region on the current Sim. Then, the RAd functionality was used to create a region on three possible environments: the same Sim, on another standalone Sim, and on a Sim as part of a grid.
The basic and RAd **Save OAR (SOAR)** functionalities were tested for storing the complete content of a region into a shared location. Furthermore, the basic and RAd **Load OAR (LOAR)** functionalities were investigated for loading the content stored at a shared location in an OAR file to a region. We also tested loading the content to a remote computer.
The basic and RAd functionalities for both **Delete Region (DR)** and **Remove Region (RR)** commands were inspected firstly on current Sim, and secondly on a remote Sim.
The **boundary crossing effects** for an avatar were observed while it was walking to or flying between regions on a single Sim for both standalone and grid modes. We further examined boundary crossings between regions on different Sims over a grid infrastructure.

The basic teleport mechanism was tested to teleport between adjacent regions on both standalone and grid environments. We also tested teleports between non-neighbouring regions. Moreover, we tested multiple random teleports to different regions on a grid. The teleport mechanism of scene class was used explicitly to transfer a player from one region to another region and then transfer it back to its home region.
To investigate the effects of delegating a region to a new Sim on grid, we tested an avatar walking, flying, and teleporting between the transferred region and its neighbouring regions.

Based on the successful results of these experiments, we found that the OS is flexible enough, and its components can easily be utilised to extend its current architecture. A scalability framework based on our work is presented and implemented in chapter 7. However, the most important issue here is to determine the content and player capacity while keeping performance in mind. Similarly, determining a point when a system needs to initiate a split and, conversely, when a system requires to start a merge operation are critical factors in developing scalable worlds. For these reasons, we conducted a number of different sets of experiments to test the OS behaviour against both static and dynamic content as well as interactive clients. These experiments, their analysis and an abstract load model based on these experiments are presented in the next section.

6.3 A Generic Load Model

According to Gupta et al. [92], SL and WoW both have a capacity well below 100 interactive users over a high speed server. However, current VWs normally employ a number of restrictions on their content players' activities. To determine the system behaviour against static and dynamic content as well as interactive users, we performed different sets of experiments. The main purpose was to develop a generic model that predicts the system capacity and determines a point in the system when it needs to initiate a split. Similarly, it could be used to approximate a value for initiating a merge operation. Initially, the experiments were conducted with a single region and were then extended to see system behaviour for multiple regions on both Windows and Linux platforms. The results are presented as graphs based on average values of a large number of observations (collected for approximately three minutes or more) for each experiment and on the Standard Deviation (STDEV) that shows the variations in these observations. The measures are based on observations taken during steady states of the OS world, and each experiment was started with empty region(s) that were then populated using OAR commands. The workload described in section 6.1.3 is used to populate the regions and, to determine the impact of load introduced by the interactive clients, we used two scripts named ScriptA and ScriptB. The former repeatedly executes a sequence of four operations: forward 5, back 5, go home and sleep 10. It tells a Bot to go forward for five seconds, then go backward for five seconds, followed by a teleport to a home location and then sleep for ten seconds. The player's home location is assigned before running the script. The latter is a modified form of ScriptA that removes the sleep command to further stress the system.

6.3.1 Experiments on the Windows Environment

The experiments in this section use the Windows environment described in section 6.1.2. The dual core system is used to run both an OS instance and MySQL database that holds regional data. The rest of the systems are used to login Bots to the system. We initially started with three parameters (SimFPS, PhysicsFPS, and CPU% usage) to study a system behaviour, but later on we concentrated on SimFPS. This is because PhysicsFPS was following exactly the same pattern as SimFPS, and CPU was never found to be a bottleneck in our experiments. However, it was used up to its full capacity in a single set of experiments based on in-world scripts. The sets of experiments are presented in the following order: Static Content, Dynamic Content using In-world scripts, Logged-in Bots with no Activities, Logged-in Bots running ScriptA, Logged-in Bots running ScriptB, and Logged-in Bots in 2 Regions running ScriptB. The general outcomes are then discussed before starting experiments on the Linux environment.

Static Content

In this set of experiments, we examine the system behaviour for an increasing number of static content in a single region of a Sim. The experimental data used in these experiments are:

- Empty Region, no content
- 2000 prims
- 4000 prims
- 6000 prims
- 8000 prims
- 10000 prims

Figure 6.1 presents the Mean and STDEV for SimFPS, PhysicsFPS and CPU%. It reveals that SimFPS is between fifty and sixty for all the experiments, with the highest observed value of fifty-seven. The PhysicsFPS has a similar outcome but it remains in a range of forty to fifty, with forty-seven being the maximum observed. Figure 6.1 further shows that, in a steady state, the CPU utilisation is almost zero since no activities are happening in the world once the content is loaded into both the scene and database.

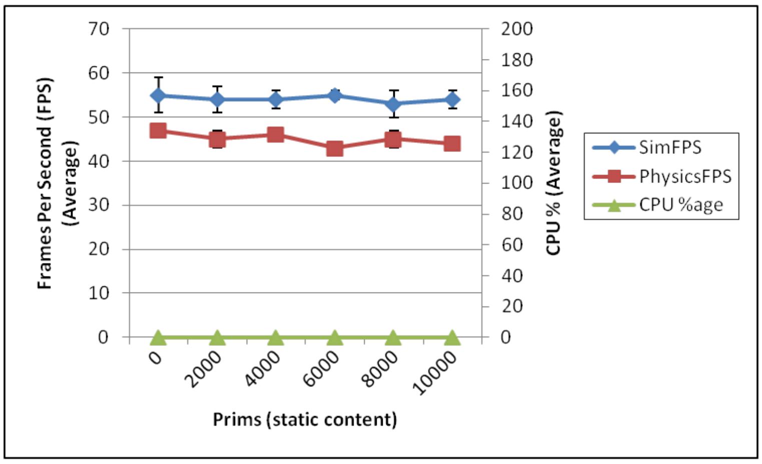


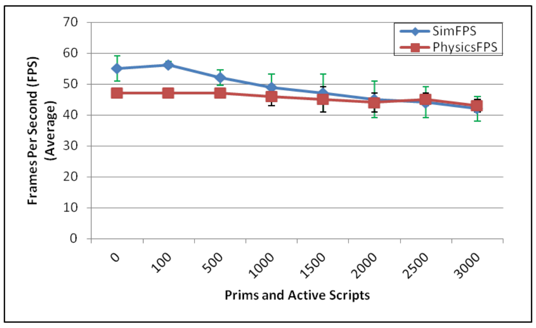
Figure 6.1: Mean and STDEV for an increasing number of static content for SimFPS, PhysicsFPS and CPU%.

Dynamic Content using In-world Scripts

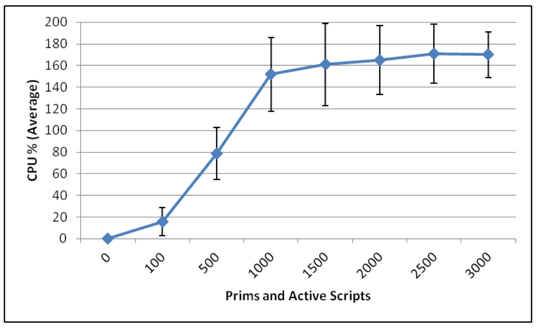
To explore the system behaviour for the content with dynamic behavior using in-world scripts, we performed a set of experiments with an increased number of Prims and Active Scripts (PandASs). The number of PandASs in each experiment are given as:

- Empty Region
- 100 PandASs
- 500 PandASs
- 1000 PandASs
- 1500 PandASs
- 2000 PandASs
- 2500 PandASs
- 3000 PandASs

Figure 6.2 shows the Mean and STDEV for SimFPS, PhysicsFPS, and CPU%. It can be seen in Figure 6.2(a) that SimFPS gradually decreases with an increase in number of dynamic content. However, PhysicsFPS has no noticeable decrease. Since the dynamic content is continuously rotating each cube and changing its colour, the CPU% (shown in Figure 6.2(b)) parameter shows a significant increase in CPU utilisation as the capacity increases. It can be seen that CPU utilisation is up to 200% in two cases.



(a)



(b)

Figure 6.2: Mean and STDEV for an increasing number of dynamic content (PandASs) for (a) SimFPS and PhysicsFPS, and (b) CPU%.

Logged-in Bots with no Activities

In this set of experiments, we wanted to check the system performance for interactive Bots that are logged-in but doing nothing. Starting with no players, we add ten players each time to a successive experiment. The Mean and STDEV for SimFPS, PhysicsFPS, and CPU% for this set of experiments are presented in Figure 6.3. It reveals that logged-in Bots that perform no activities have no impact on either SimFPS and PhysicsFPS, and they behave like static content. However, the system in this case consumes CPU time because it sends regular updates to the clients, as shown in Figure 6.3. Since updates are sent periodically at diverse time intervals for different sets of Bots, the CPU utilisation shows big variations.

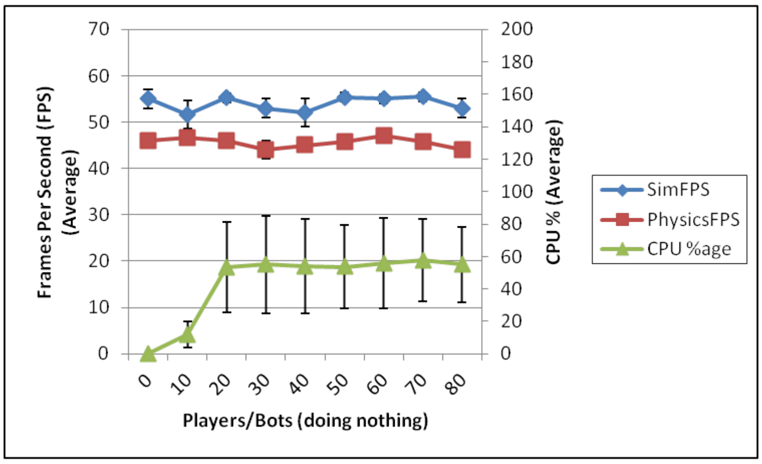
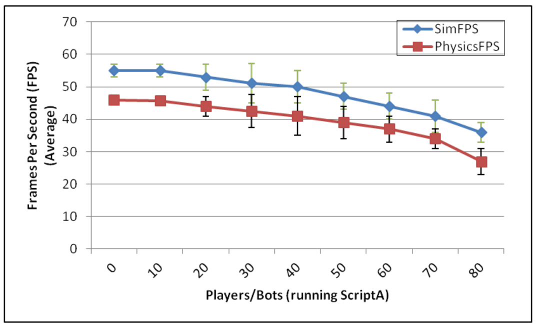


Figure 6.3: Mean and STDEV for an increasing number of Players/Bots logged-in but doing nothing for SimFPS, PhysicsFPS, and CPU%.

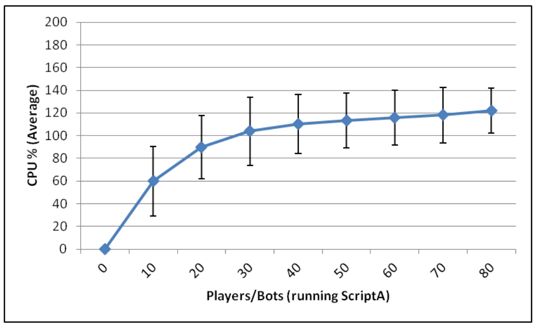
Logged-in Bots running ScriptA

In this set of experiments, the experimental descriptions are the same as before but Bots are asked to repeatedly follow the activities provided in ScriptA.
The experimental results are presented in Figure 6.4. Figure 6.4(a) reveals that SimFPS is gradually decreased as the number of interacting players are increased. It further shows that the PhysicsFPS is decreased with a similar rate as SimFPS. We believe that the interactive clients have a great impact on frame rate, but

the system in this case gives better performance because the script commands are applied to Bots in sequence and different sets of Bots are sent to sleep mode for ten seconds, thus executing an even number of players at a given time. It also gives long variation for frame rates due to these reasons. The CPU utilisation is increased with the addition of more Bots in each successive experiment, as shown in Figure 6.4(b). On average, the system processes the same number of Bots and, therefore, CPU utilisation is almost the same in each case. To examine how the system behaves when all the Bots are continuously involved in different activities, we used a modified version of ScriptA for the next set of experiments. The basic aim was to determine the point (system capacity) when the system performance starts degrading.



(a)



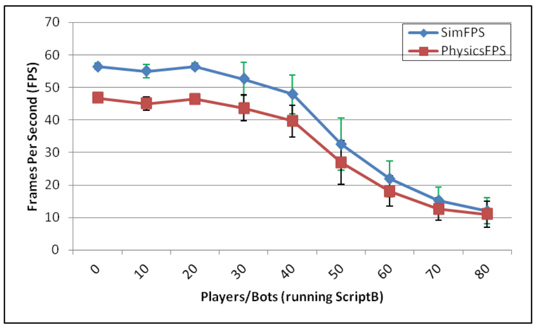
(b)

Figure 6.4: Mean and STDEV for an increasing number of players/Bots running ScriptA for (a) SimFPS and PhysicsFPS. (b) CPU% Usage.

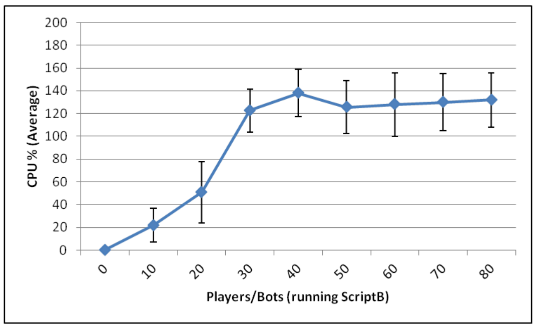
Logged-in Bots running ScriptB

In this set of experiments, we used ScriptB and asked an increasing number of Bots in each experiments to follow the activities of the script. The number of players in each experiment are, again, the same as before.

The results are explored in Figure 6.5. It can be seen in Figure 6.5(a) that both SimFPS and PhysicsFPS maintain an acceptable frame rate for up to forty players. However, the frame rate drops quickly when more players are added and that reduces the frame rate significantly, thus degrading the user interactive experience.



(a)



(b)

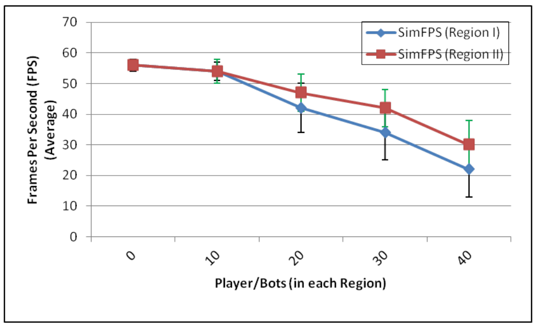
Figure 6.5: Mean and STDEV for an increasing number of players/Bots running ScriptB for (a) SimFPS and PhysicsFPS. (b) CPU%.

The CPU utilisation is normal in this case as shown in Figure 6.5(b).

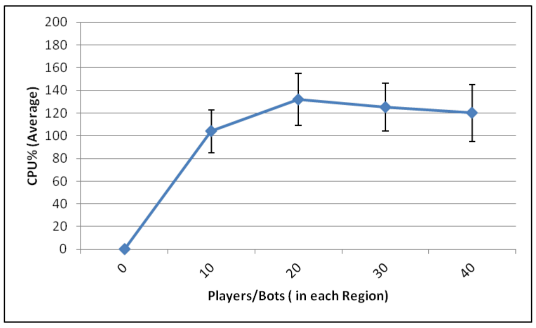
Based on the experiments presented so far using a single region, we identified that interactive clients and their activities have most impact on system performance (described in terms of SimFPS and PhysicsFPS). The system performance degrades greatly with an increase in the number of players and frequency of their activities. It was further observed that an increasing number of dynamic content also has a gradual but slight impact on these parameters. However, CPU utilisation is greatly increased for them. These parameters are vital for the development of a load model, but first we need to investigate how a system with multiple regions behaves against these parameters. This is because an OS instance can host an arbitrary number of regions. We used SimFPS and CPU utilisation for this set of experiments because the PhysicsFPS always follows the same pattern as SimFPS. To examine how a system with multiple regions behaves against presented load, we discuss another set of experiments in the next section with a Sim with two regions.

Logged-in Bots in 2 Regions running ScriptB

This set of experiments is different to the rest of the experiments as the players are now equally distributed among two regions and they are increased by ten in each region in successive experiments. They are also repeatedly executing the activities listed in ScriptB.



(a)



(b)

Figure 6.6: Mean and STDEV of increasing number of players/bots equally distributed among 2 regions running ScriptB for (a) SimFPS (Region I) and SimFPS (Region II). (b) CPU%.

Figure 6.6 provides the SimFPS parameters of both Region-I and Region-II as well as CPU utilisation. It is clear that a system for up to forty players (twenty in each region) in a Sim maintains the rate of Frames Per Second (FPS) above 30FPS, as shown in Figure 6.6(a). SL maintains a minimum FPS for an acceptable performance [92]. However, it can be seen that its performance is degraded when more players are added to the Sim. The SimFPS for Region I drops below 30FPS when the system has about sixty players. Since the actual world could have different content and interactive players, we therefore believe that when the FPS for any of the region falls below a certain limit, the system should either stop taking further connections or distribute the load with additional Sims. This is because it degrades the overall performance of a system. To avoid a negative experience, a distribution needs to be initiated at much a higher value than 30FPS as, after a certain range, the rate goes down very quickly. The worlds with more than two regions yield similar outcomes and are, therefore, not included in this thesis. The CPU load is again not beyond the capabilities that is presented in Figure 6.6(b).

Discussion

We performed different sets of experiments and observed that interactive clients have the most impact on system performance. Dynamic content also showed some impact. We used SimFPS, PhysicsFPS, and CPU utilisation to determine a system's behaviour against different workloads. However, the PhysicsFPS followed exactly the same pattern as SimFPS. SimFPS is the main measure which is ultimately used for the development of our load model. The CPU utilisation is never observed to be a bottleneck for all experiments. When a TClient was used to log in a large number of Bots, we observed network issues such as delays and drop of connections. We overcome these issues by restricting up to ten Bots per TClient. Since not all sets of experiments showed an impact, it is necessary to repeat limited sets of experiments to see how additional hardware support the workload presented in each experiment. Given that CPU is never used to its full extent for a dual core node, it is assumed that it might not be a potential bottleneck over a dual quad core node, and therefore it is not calculated for the experiments over Linux environment. In the next section, we put emphasis on dynamic content and interactive players. The main goal is to see if more resources can assist in achieving an improved performance.

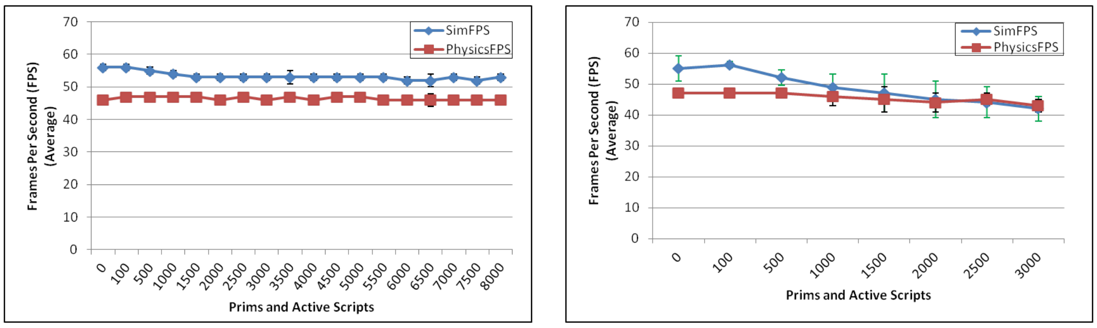
6.3.2 Experiments on the Linux Environment

This section use a more sophisticated and high speed computation facility using the Linux platform described in section 6.1.2. Grid services (a Robust instance) and their corresponding database are running on a dedicated node. The region server is running on a different node while a number of other nodes are used to log in Bots to the world. The sets of experiments that are repeated in this section include Dynamic Content with In-world Scripts, Logged-in Bots running ScriptA, Logged-in Bots running ScriptB, and Logged-in Bots in 2 Regions running ScriptB. The graphical results of these experiments on the Windows environment are reproduced for comparison purposes in this section. Only SimFPS and PhysicsFPS parameters are considered in this section.

Dynamic Content using In-world Scripts

Since we observed that there was an impact of dynamic content (PandASs) on SimFPS and PhysicsFPS in the Windows environment, we repeated this set of experiments with an increasing number of PandASs by adding 500 PandASs in each successive experiment. We performed an extended set of experiments with up to 8000 PandASs due to the fact that the Linux node has four times more cores compared with the Windows node used which has two cores.

Figure 6.7(a) shows the results for both SimFPS and PhysicsFPS. It demonstrates that the system behaviour for a very large number of PandASs is stable and there is no decrease in FPS for either of the parameters compared with the Windows node (see Figure 6.7(b)). These results revealed that different cores of the node used for running these experiments are handling part of the in-world activities, thus keeping the rate of FPS stable. However, a Sim is normally assigned to a single core in a grid environment that could possibly host multiple regions. Therefore, the content is usually restricted up to a certain level for a region. In the next set of experiments, we wanted to explore the behaviour of a Linux node and the impact of different cores of a node when used against an increasing number of Bots.



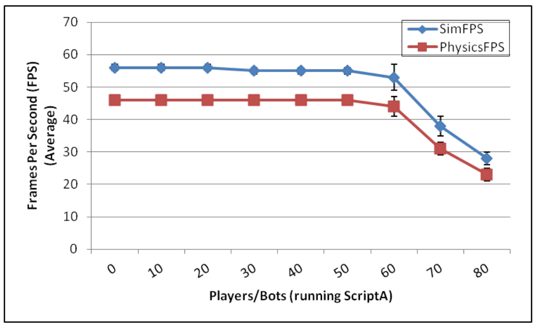
(a) (b)

Figure 6.7: Mean and STDEV for an increasing number of dynamic content (PandASs) for (a) SimFPS and PhysicsFPS on a Linux node, and (b) SimFPS and PhysicsFPS on a Windows node (from Figure 6.2(a)).

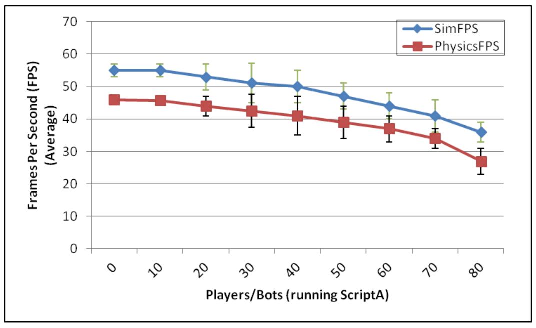
Logged-in Bots running ScriptA

Exactly the same set of experiments of the Windows environment was repeated on the Linux environment, and the increasing number of players follow the commands given in ScriptA.

Figure 6.8 (a) shows the system behaviour using both SimFPS and PhysicsFPS and it can be seen that it provides more stable results with few variations compared with the results of the same set of experiments on the Windows environment (shown in Figure 6.8(b)). It handled up to fifty Bots without any degradation of SimFPS but it started degrading the rate of FPS at about sixty players and dropped considerably when more Bots were added. There is a quick decline between sixty and eighty Bots where, for eighty players, the rate of FPS drops lower than 30FPS. This demonstrates that using multiple cores failed to help increase the number of Bots. By comparing the results in both Figure 6.8(a) and Figure 6.8(b) for Linux and Windows environments, we can see that both environments have almost the same trends for handling interactive Bots. Therefore, the use of multiple cores is unable to achieve better performance than the simple systems. Since we used more nodes to log in Bots and logged in less Bots per TClient, the Linux environment has no connection drop issues. To further stress the system and see how the system responds to heavy load, the same set of experiments (called Logged-in Bots running ScriptB) was repeated on the Linux environment in the next section.



(a)

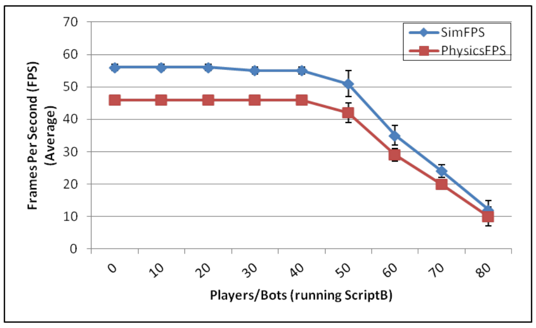


(b)

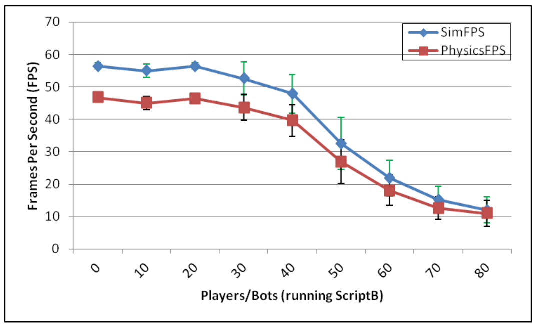
Figure 6.8: Mean and STDEV for an increasing number of players/Bots running ScriptA for (a). SimFPS and PhysicsFPS on a Linux node, and (b) SimFPS and PhysicsFPS on a Windows node (from Figure 6.4(a)).

Logged-in Bots running ScriptB

Figure 6.9(a) shows very similar results for this set of experiments on the Linux environment compared with its results on the Windows environment presented in Figure 6.9(b) for total number of players it can handle. The Linux environment showed a slightly improved resistance where it managed to handle up to sixty players before its FPS dropped below 30FPS. The Windows environment crosses the limit a few times while handling fifty players but the average was almost 30FPSs.



(a)



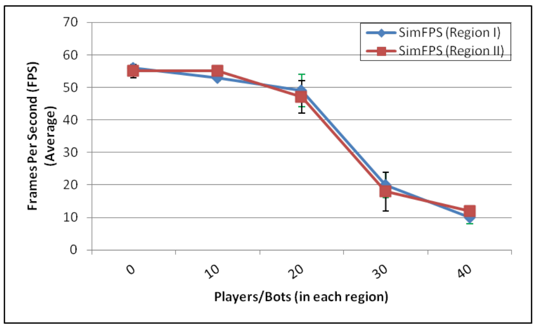
(b)

Figure 6.9: Mean and STDEV for an increasing number of players/Bots running ScriptB for (a). SimFPS and PhysicsFPS on a Linux node, and (b) SimFPS and PhysicsFPS on a Windows node (from Figure 6.5(a)).

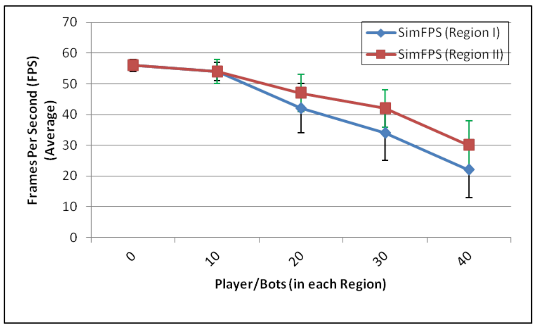
Logged-in Bots in 2 Regions running ScriptB

To determine how a Linux node behaves when there are two regions in a Sim, we repeated this set of experiments with regions being populated equally with ten additional players in each region in each successive experiment. The players are again executing ScriptB.

The results of the experiments on the Linux environment are presented in Figure 6.10(a) and compared with the corresponding set of experiments on the Windows environment, shown in Figure 6.10(b). It shows that it handles up to forty Bots with a rate of FPS above 30FPS but the rate of FPS with additional Bots dropped more quickly than the Windows environment. The Windows environment showed improved behaviour where it performed much better for Bots between forty and sixty. The Linux environment was consistently under 30FPS for sixty and more Bots. The Windows environment had variations where it dropped a few times, below 30FPS but it maintained an average of just above 30FPS for sixty players.



(a)



(b)

Figure 6.10: Mean and STDEV for an increasing number of players/Bots (in each region) for a 2 Region world running ScriptB for (a) SimFPS on a Linux node, and (b) SimFPS on a Windows node (from Figure 6.6(a)).

Discussion

We performed selected sets of experiments on the Linux environment to determine system behaviour with additional hardware resources and compared their results with the corresponding sets of results on Windows environment. It was determined that, with the exception of dynamic content, the rest of the experiments for both environments gave a similar response against the same workload added to these environments. The most interesting finding of the experiments on the Linux node is that the provision of additional hardware failed to perform better compared with the Windows node. We faced a number of issues with the OS framework when used on Linux platform. It crashed frequently and showed very unstable behaviour compared with the Windows environment. The potential reason might be its development on .Net framework for Windows environment. However, it works well with Linux and mono framework but the best working combination of OS and mono frameworks needs to be found as they are continually under development.

6.3.3 Comparison

Based on a large set of experiments on both Windows and Linux environments, we came to the conclusion that the provision of additional hardware cannot improve

system performance. Liu et al. [140] have described this as a system bottleneck that restricts OS worlds to scale, and is due to the simulation centric architecture used by the OS framework. Most of the VWs using this architecture put restrictions on both activities and content. However, our concern is to develop a model for the current architecture to overcome issues with both static and dynamic systems and to improve their performance. For a scalable system, the most critical decisions are when to initiate a split and a merge operation. To initiate a split, we need to take action before it degrades to a point which is not acceptable. Based on these experimental results, we aim to develop a generic load model to help the system to determine suitable decisions while maintaining good performance.
The Linux node has the capacity to hold more static and dynamic content. However, for performance reasons they need to be restricted for the following reasons: normally a Sim is usually assigned to a single core of a node and has restricted capabilities; content storage is a sequential process that takes a long time; static content is stored once to the database but dynamic content needs to be stored periodically for persistence; and region removal with much content requires a considerable amount of time. Based on the results performed in both environments, we can conclude that a system with a reasonable resources can handle a world with reasonable amount of content and up to sixty Bots performing a variety of different actions. However, more than sixty players will degrade system performance and a Sim should never allow it. We suggest initiating a split when the FPS for a region is decreased and goes below 40FPS. For most of these experiments, this was the point where the system was supporting approximately forty players. The following section defines a number of concepts used in our final experiments based on the outcome of these experiments.

6.3.4 SplitCapacity, SimCapacity and MergeCapacity

For the final experiments presented and discussed in chapter 7, we take both split and merge decisions based on the number of interactive players. Since both Windows and Linux environments showed a similar behaviour, we use the same values for the concepts defined as SplitCapcity, SimCapacity and MergeCapacity on both platforms. **SplitCapacity** is taken to be an approximate value when SimFPS was observed dropping below than 40FPS in one of the regions to avoid a negative experience and improve performance. In both cases, there were around forty players involved in different simultaneous activities. However, a Sim can hold up to sixty players (named as **SimCapacity**) with varying degrees of interactivity in addition to a reasonable amount of static and dynamic content. It is an estimated value based on the aforementioned experiments for which a Sim managed to maintain 30FPS for SimFPS on average. We allow up to sixty players per Sim if no further distribution is possible. When a system reaches this capacity, it declines to accept more connections. **MergeCapacity**, on the other hand, is used as a minimum limit to initiate a merge operation. To avoid frequent splits after merge operations, we assume a MergeCapacity to be half of the SplitCapacity that is twenty players for our experimental work.

6.3.5 The Load Model

According to Gupta et al. [92], SL maintains a frame rate of at least 30FPS for a satisfied level of user experience. We have used this as an argument for the development of our abstract load model. The basic purpose is to help developers and practitioners using OS architecture to take the most vital decision about initiating a load distribution process. Split is normally initiated at an earlier stage to avoid worst situations and bad user experience (considered at least 40FPS for our work). Since a Sim can host multiple regions, we initiate a split when the rate of FPS for any region goes below 40FPS. We use only the SimFPS parameter to define our load model because both SimFPS and PhysicsFPS degrade at the same rate. However, if the requirements have a different effect on PhysicsFPS, it could be included in the model.

Our abstract load model that initiates a split is presented in Algorithm 3. Each Sim maintains a separate Queue for each scene it manages to hold the observations based on SimFPS. Initially, each slot is assigned a maximum value of fifty-seven for SimFPS being set by the OS framework. It maintains a number of most recent observations recorded during a time interval and are used to take a split decision. For our approximated values used in this work, we took an observation every twenty seconds and used the ten most recent observations to determine an average

Algorithm 3 A Generic Load Model

```
Require: List<scenes>
Ensure: Alert the system to initiate partitioning
1: for $i \leftarrow 1$ to Count(List<scenes>) do
2:     Get the latest SimFPS value for the scene;
3:     Update the corresponding queue for the scene; //deletes the oldest and adds a latest observation
4:     Get Mean of the corresponding queue; //determines average of observations for this scene
5:     if (Mean < 40) then
6:         Alert the system to initiate partitioning;
7:         Break;
8:     end if
9: end for
```

value. However, different values could be selected depending on the requirements. It is therefore left undefined and need to be decided by the developer. The Sim initiates this model periodically which determine one of the two possible outcome. It takes the list of scenes as input and scans the entire set of scenes as follows. It first determines the latest value of SimFPS for a scene and then updates the Queue for the corresponding scene. Based on an average value, it alerts the Sim to initiate a split if the Mean value for a scene goes below 40FPS. This model provides a point in the system at which to initiate a split and a variation of this model can be used to determine SimCapacity of a Sim. It needs to maintain on average, a frame rate of 30FPS instead of 40FPS. However, the merging can be initiated against a numeric value that is normally selected against SplitCapacity of a Sim.

6.4 Introducing the Scalability Model

This section introduces a simple scalability model and identifies activities during both a split and merge operations between different Sims. Based on the identified activities, we then determine the OS components that could be used to implement this scalability model. It outlines time statistics that are used in section 6.6 for an informal time analysis of experiments based on content and interactive players.

Introduction

To scale an OS world comprised of multiple regions hosted by a single Sim, the traditional methods transfer some of the regions to another Sim. These scalable

systems always use a grid model which integrates the whole world by managing a centralised map of regions. Each region on the grid has a unique place and, to give a coherent space, the regions are arranged adjacent to each other. Keeping these issues in mind, spatial partitioning methods always transfer regions by serialising their content as well as players. Once the content is transferred and the region is up, the players are de-serialised again to resume their normal activities. We believe that the players in these methods have a bad experience, and to avoid freezing the players and to improve their experience, we adopt a contemporary approach in our work that transfers all the players temporarily to an intermediate region during a region transfer. This region is called a transit region and is only used during split and merge operations. In our case, players can move freely to other regions or keep themselves busy with simple activities available at the transit regions. Once their original region has been relocated, we teleport them back to their original position and they resume their normal activities.
In this section, we introduce our approach for transferring a region from one Sim to another Sim with the help of two Sims and a regional grid of two regions with two additional transit regions for the Sims, as shown in Table 6.2. This simple model is then converted into a more generalised framework in chapter 7. The parent Sim (Sim-I) has a regional grid of two regions, named A and B, and a transit region called T1. We have a single child Sim that is hosting no actual region but rather a transit region called T2 and is available to share the load. Both the Sims are connected to a grid instance. Each region presents content and players visit these places based on interest.

Parameter	Sim-I (Parent)	Sim-II (Child)
HTTP Listening Port	9000	9005
Regions(Region:X,Y Coordinates: Port)	A:1000,1000:9000 B:1001,1000:9001	Nil Nil
Transit Region(Region:X,Y Coordinates:Port)	T1:1005,1005:9105	T2:1006,1006:9106

Table 6.2: The description of 2 Sims and a world with 2 regions

6.4.1 Steps in Scalability

Each Sim that has at least a real region continuously monitors the load and it can be in any of the three states, which are: a normal state, an overloaded state, and an under-loaded state. Each of these states requires it to take appropriate action as described next. If there are no players in a region to transfer, it skips the two steps that are used to transfer players.

Normal State

In this state, the system is running a normal load and no split or merge is required.

Overloaded State: Split

If a system (Parent Sim, Sim-I in this case) becomes overloaded, it delegates one of the regions (say A) to the other Sim. The following actions in sequence are required to transfer a region:

- Save the content of region A into a shared space
- Move the players in region A to region T1
- Obtain the region A specifications
- Remove region A from Sim-I
- Create region A on Sim-II with the same specifications
- Load the content to region A on Sim-II
- Move the players from region T1 (on Sim-I) to region A (now on Sim-II)

This ends the splitting process and normal activities for the players are resumed. It doubles system capacity and players are given a better alternative than freezing.

Underloaded State: Merge

In case the capacity of child Sim (Sim-II) goes below a minimum threshold, it initiates a merging process. The merging process uses the following steps in the given order:

- Save the content of region A into a shared space

- Move the players to region T2
- Obtain the region A specifications
- Remove region A from Sim-II,
- Create region A on Sim-I using original specifications
- Load the content to region A on Sim-I
- Move the players from region T2 (on Sim-II) to region A (now on Sim-I)

This terminates the merging process and the players' activities are resumed. Sim-II is again available to share the load. It is important to note that activities across different Sims require central grid services.

6.4.2 Required Components

To implement the activities described in our simple scalability model, we identified the following components and capabilities of OS and RAd functionality:
The OAR functionality provides an advanced serialisation method that can be used to store regional data before deleting a region. It provides methods to store data to an OAR file and load data from an OAR file, as described earlier in section 5.3.2 of chapter 5.
RAd functionality implements a wide range of server commands to be executed on a remote server. The following methods can be used to implement the activities in our scalability model.
The **Save OAR (SOAR) method (admin_save_oar)** issues a command to a remote server to save a regional content in an OAR file. It requires the name of the region and a file name. The **Load OAR (LOAR) method (admin_load_oar)** sends a command to a remote server to load the content of an OAR file to a given region. It also requires a region name and the name of the OAR file. The **Create Region (CR) method (admin_create_region)** is used to create a region on a remote computer. It takes the following mandatory parameters: region_name, listen_ip, listen_port, external_address, region_x, region_y, and estate_name. The **Close Region (CsR) method (admin_close_region)** remotely allows to remove a region from a Sim and takes region_name as an input. Data from the database is not deleted in this method. The **Delete Region (DR) method**

(admin_delete_region) remotely allows a region to be deleted from a Sim and it deletes its data from the database. It also takes region_name as a parameter.
The remaining activity to suppose is transferring players from one region to another region. The scene object has a teleport method that can be used explicitly for this purpose. The grid service facilitates determining a remote region based on region specifications.

6.4.3 Statistical Parameters

In this section, we describe the time parameters that we track to determine the timings actually taken by different activities during a transfer. Based on the activities identified in our scalability model, we collect the following statistics: Store Content Time (SC Time), Load Content Time (LC Time), Remove Region Time (RR Time) (representing both Delete Region (DR) and Close Region (CsR)), Create Region Time (CR Time), Transfer to Transit Time (T2T Time), Transfer to Region Time (T2R Time), Content transferred, Number of players transferred, and Average player transfer Time.
The time to store the content of a region into an OAR file (using SOAR method) is called **Store Content Time (SC Time)**.
The **Load Content Time (LC Time)** represents time to load content from an OAR file (using LOAR method) to a region.
The time to remove a region using either Delete Region (DR) or Close Region (CsR) methods from a Sim is called **Remove Region Time (RR Time)**. A parameter, called case, then identifies a method during an experiment.
The time to create a new region on a remote Sim is called **Create Region Time (CR Time)**. It is usually based on the specifications of original region that is relocated during a transfer.
The total time to explicitly teleport the players in a region to a transit region when relocating a region is called **Transfer to Transit Time (T2T Time)**.
The **Transfer to Region Time (T2R Time)** represents the total time to transfer players back to a region from a transit region.
Content transferred parameter represents regional content that is transferred when relocating a region.

Number of players transferred maintains the number of players that are transferred due to a split.
Average player transfer Time is the time a player takes on average to transfer from one region to another.

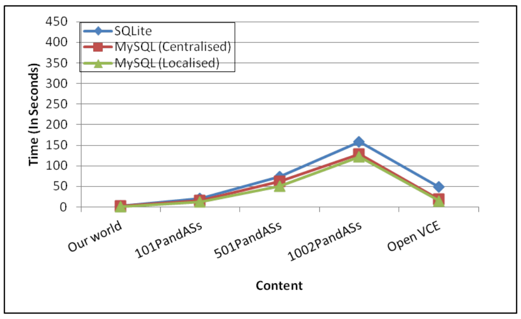
Activity	Case	SQLite (Sec)	MySQL (Centralised) (Sec)	MySQL (Localised) (Sec)
Create Region	Any	4Sec	3Sec	2Sec
Teleport to Location	Any	6Sec	5Sec	5Sec
Load Content (LC)	Our world	3Sec	1Sec	1Sec
	101PandASs	3Sec	2Sec	2Sec
	501PandASs	7Sec	3Sec	2Sec
	1002PandASs	11Sec	6Sec	6Sec
	OpenCVE	114Sec	94Sec	90Sec
Store Content (SC)	Our world	1Sec	1Sec	1Sec
	101PandASs	1Sec	1Sec	1Sec
	501PandASs	2Sec	1Sec	1Sec
	1002PandASs	2Sec	2Sec	2Sec
	OpenCVE	8Sec	6Sec	6Sec
Delete Region (DR)	Our world	2Sec	2Sec	1Sec
	101PandASs	20Sec	16Sec	12Sec
	501PandASs	74Sec	62Sec	51Sec
	1002PandASs	158Sec	128Sec	122Sec
	OpenCVE	49Sec	19Sec	15Sec
Remove Region (RR)	Our world	2Sec	2Sec	2Sec
	101PandASs	16Sec	14Sec	12Sec
	501PandASs	90Sec	81Sec	74Sec
	1002PandASs	421Sec	201Sec	196Sec
	OpenCVE	5Sec	3Sec	2Sec

Table 6.3: A comparison of time taken by different activities for three database options (SQLite (localised), MySQL (centralised), and MySQL (localised)) hosting a Sim data on Windows environment.

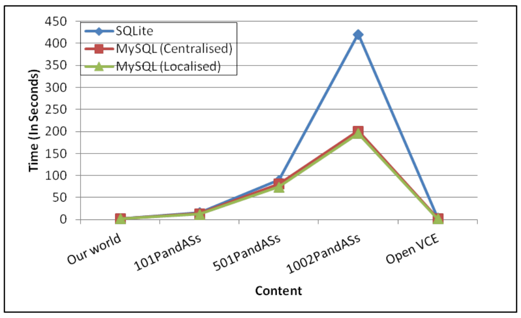
6.5 Investigating Database Options

This section investigates and compares three different databases configurations that can be used with a grid infrastructure to store the data of regional servers. The basic aim is to determine the best option that could be feasibly used for large scale VWs. Grid services (implemented as a Robust.exe instance) always

use a MySQL database. The following three database options are used to store regional data of different Sims: local SQLite data file for each Sim, using a centralised MySQL database for all Sims, and a local MySQL instance for each Sim. To evaluate and see the behaviour of these configurations, we used a number of different regional commands to obtain time statistics for their corresponding activities. They are applied from the console of a regional server and used to compare the time information for different database options. Table 6.3 provides the details of activities identified by our split model and their time information.
Create Region (CR) and teleport operations have no direct link with databases but the MySQL options still give a better response. The teleport operation takes various amounts of time as it might need to download the regional data to a viewer. Load Content (LC) is basically adding the content to a scene and then later, as a background process, it stores the data into a database. SQLite gives a much slower response in loading the content of an OAR file to a scene compared with MySQL options. SQLite is a lightweight system and is unable to maintain the data of large scale VWs. The Store Content (SC) operation takes almost the same amount of time for all the options because it has no concern with databases. A system bottleneck was found while deleting a region from a Sim. If a DR is called during a periodic backup, the system first completes the backup process and then deletes the region. SQLite was taking a long time while quitting the Sim or removing/deleting a region with large content. In some cases, when persistence was required, MySQL also took a considerable amount of time but in steady states, and for static content it performs better than SQLite. The main reason that each database takes a considerable amount of time is the fact that during a delete operation, a Sim deletes objects from both scene and database in sequence. On the other hand, the RR operation takes a longer time when the content has a number of dynamic objects and their states need to be updated in the database. It also takes a long time for those regions whose data is not already stored in the database. In general, both MySQL options take less time for both deleting a region and removing a region from a Sim, as shown in Figure 6.11(a) and Figure 6.11(b). Furthermore, the MySQL (localised) option shows a little improvement over the centralised option for our simple setup. We believe that a centralised option becomes a system bottleneck as a VW scales.



(a)



(b)

Figure 6.11: The comparison of SQLite, MySQL (Centralised), and MySQL (Localised) as a prospective configuration and their impact on (a) Delete Region (DR), and (b) Remove Region (RR).

In short, SQLite is a good option for simple experimental work and to start working with, but for further experimentation and to run a grid infrastructure, it is recommended to use MySQL. A single centralised MySQL instance could be utilised for a grid environment hosting regional data for all Sims, but it could become a system bottleneck for large scale VWs. Hence, it is recommended to use a localised database for each Sim and, from now onwards, we utilise a separate MySQL instance for each Sim.

6.6 Informal Time Analysis Model

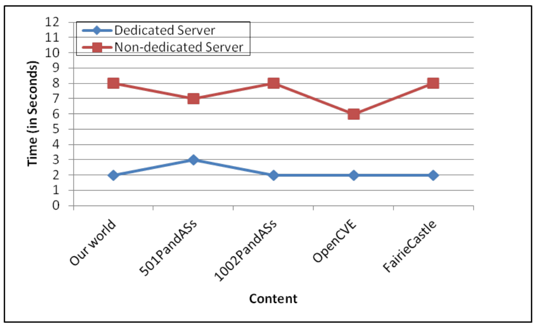
This section describes an informal way to transfer players and content from one Sim to another Sim (on a grid infrastructure) with the help of RAd methods and the teleport method of scene object of OS framework. We introduced three additional region console commands to initiate our methods for split and merge operations. Our methods combine the activities into a series of function calls to implement these operations. Two different split commands are added that differ in the way they remove a region from a Sim. One uses Delete Region (DR) and the other accesses the Close Region (CsR) method of RAd functionality for this purpose. The basic aim is to determine how much time is taken by each activity during a transfer. We use a simple grid of two Sims, described in Table 6.2, to perform a number of experiments varying both players and content. This is to examine the time statistics for the activities required for split and merge operations when they are used from a remote machine. Since both split and merge operations use the same set of activities, we have only reported the time information of selected activities from split operations for our investigations. We manually initiate these commands from the console and start each experiment with empty regions, and then populate them with either content or players depending on the experiment concerned. We use both Windows and Linux environments (described in section 6.1.2) to perform the same set of experiments, and then they are compared to observe their behaviours. Each environment is introduced before the experiments that cover three categories based on dynamic content, example worlds content, and interactive players. We use dedicated servers to host different Sims based on our investigation, which is presented in the following section (section 6.6.1). The statistics defined in section 6.4.3 are used for the informal time analysis in this chapter with the basic aim of obtaining time estimates and developing improved strategies to minimise their timings.

Experiment	Content	Non-dedicated Server		Dedicated Server	
		CR Time (Sec)	LC Time (Sec)	CR Time (Sec)	LC Time (Sec)
1	Our world	8Sec	7Sec	2Sec	2Sec
2	501PandASs	7Sec	8Sec	3Sec	3Sec
3	1002PandASs	9Sec	9Sec	2Sec	4Sec
4	OpenCVE	6Sec	109Sec	2Sec	101Sec
5	FairieCastle	8Sec	96Sec	2Sec	86Sec

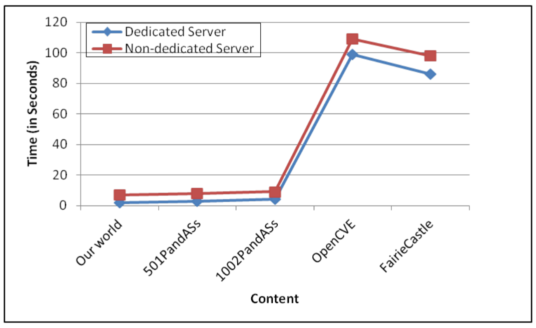
Table 6.4: The comparison of time information for both Create Region (CR) and Load Content (LC) between a dedicated and a non-dedicated Sim.

6.6.1 Dedicated and Non-dedicated Sim Servers

For a better performance, the VW Sims are normally assigned to dedicated cores on Grid infrastructures. This is because running other activities shares the computation and communication facilities that potentially degrade system performance. In this section, we provide a brief analysis of activities and their comparison between two Sims where the first is running on a dedicated system but the second is running more applications in addition to the Sim. The dedicated server was used to run the parent Sim (Sim-I) and its database instance. A dual core system was used to run an instance of Robust and the child Sim (Sim-II), in addition to two database instances for both grid and a regional server. It was also running a number of other applications. Table 6.4 provides a summary of time information for two activities using RAd methods for Create Region and Load Content on both dedicated and non-dedicated servers. It can be observed that the non-dedicated server takes much longer to perform these activities. Figure 6.12(a) shows a comparison of time taken by creating a region between a dedicated and a non-dedicated server. It can be seen that a Sim on a dedicated server creates a region much faster than the one on a non-dedicated server. It takes two to three seconds compared with the six to nine seconds of a non-dedicated server. A comparison between both Sims for Load Content from an OAR file is presented in Figure 6.12(b). It shows that a dedicated server loads the content in less time than a non-dedicated server.



(a)



(b)

Figure 6.12: The comparison of time information between dedicated and non-dedicated servers for (a) Create Region, and (b) Load Content.

In short, dedicated servers for a Sim provide much better and quicker responses to perform different activities, and also improve user interactive experience. From now onwards, we use dedicated servers for running region Sims and grid services.

6.6.2 Time Analysis on Windows Platform

In this section, we use our Windows environment, as described in section 6.1.2, for three different sets of experiments. The dual core system is used to run the grid instance and two Pentium systems are used to run two Sims (Sim-I is a parent and Sim-II is a child Sim). The fourth system is used to log in an increasing number of Bots in the third set of experiments.

Dynamic Content

Table 6.5 describes the set of experiments that transfer an increasing number of dynamic content (PandASs) and shows the time information for different activities required to transfer a region without players. RR is the only activity that takes long time for both DR and CsR. DR takes less time than CsR but it is still a considerable amount of time. In both cases, the time to remove a region increases with rise in number of content, as shown in Table 6.5. The rest of the activities (SC, CR, and LC) take an acceptable amount of time. This is because these workloads have no complex scenarios and each object is of a single prim.

Case	Exp. No.	From Sim	To Sim	Region Name	Prims and Active Scripts	SC Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)
Delete-Region	1	Sim-I	Sim-II	A	8Prims only	1Sec	2Sec	2Sec	2Sec
	2	Sim-I	Sim-II	A	101 each	1Sec	12Sec	2Sec	2Sec
	3	Sim-I	Sim-II	A	501 each	2Sec	56Sec	3Sec	3Sec
	4	Sim-I	Sim-II	A	1002 each	2Sec	126Sec	2Sec	4Sec
	5	Sim-I	Sim-II	A	1503 each	3Sec	170Sec	3Sec	6Sec
Close-Region	1	Sim-I	Sim-II	A	8Prims only	1Sec	1Sec	2Sec	2Sec
	2	Sim-I	Sim-II	A	101 each	1Sec	12Sec	3Sec	2Sec
	3	Sim-I	Sim-II	A	501 each	1Sec	77Sec	2Sec	3Sec
	4	Sim-I	Sim-II	A	1002 each	2Sec	196Sec	2Sec	5Sec
	5	Sim-I	Sim-II	A	1503 each	3Sec	343Sec	2Sec	7Sec

Table 6.5: Summary of the time information for experiments transferring dynamic content (PandASs) on Windows environment.

Example Worlds

In this set of experiments, the region that is transferred to the child Sim is populated with one of the example worlds content described in section 6.1.3. Most of the content in these worlds are static except FairieCastle which has a number of dynamic scripts. Table 6.6 shows the results of the experiments which show that both SC and CR take time which falls in an acceptable range. The removal time for DR is longer than CsR, except for FairieCastle which takes time during the backup process before closing it. The OpenVCE content was not stored in the database when we called CsR and, therefore, it took longer to first store the content in the database before closing the region. However, when the content was stored, the same operation for OpenVCE world took just a few seconds. Since the example worlds represent proper environments with reasonable numbers of objects and scene complexity, they took a considerable amount of time to load content from an OAR file to the corresponding region.

Case	Exp. No.	From Sim	To Sim	Region Name	Regional Content	SC Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)
Delete-Region	1	Sim-I	Sim-II	A	OpenCVE	8Sec	13Sec	2Sec	101Sec
	2	Sim-I	Sim-II	A	FairieCastle	10Sec	19Sec	2Sec	86Sec
	3	Sim-I	Sim-II	A	Maya Pyramid	4Sec	5Sec	2Sec	18Sec
	4	Sim-I	Sim-II	A	CSI World	5Sec	25Sec	2Sec	38Sec
	5	Sim-I	Sim-II	A	EducationSim	2Sec	17Sec	2Sec	10Sec
Close-Region	1	Sim-I	Sim-II	A	OpenCVE	7Sec	131Sec	2Sec	96Sec
	2	Sim-I	Sim-II	A	FairieCastle	8Sec	25Sec	2Sec	91Sec
	3	Sim-I	Sim-II	A	Maya Pyramid	3Sec	2Sec	3Sec	18Sec
	4	Sim-I	Sim-II	A	CSI World	4Sec	3Sec	2Sec	39Sec
	5	Sim-I	Sim-II	A	EducationSim	1Sec	3Sec	3Sec	13Sec

Table 6.6: Summary of the experiments showing timing information of different activities when transferring a region populated with example worlds content.

Interactive Clients

In this set of experiments, we investigate the time taken to transfer an increasing number of Bots between regions. We consider an empty region with no content. It takes a few seconds to remove a region from a Sim as well as to create a region on another Sim. Table 6.7 shows the details of experiments that reveals interesting information. Each player takes on average nine seconds to transfer from one region to another. However, in the case of network overload, it could take much longer as a system normally retries a few times before initiating a timeout signal. One such situation can be seen in experiment 3 for T2R Time, which takes 142Sec instead of about 90Secs for a total of ten players. It was observed that three Bots were disconnected in this situation, and the system retries to contact them, was the reason to increasing the total time taken by the transfer. However, the disconnection issue was potentially due to the non-mature nature of the TClient application and happened rarely. It normally happened when we used an instance of TClient to log in a large number of Bots. Hence, a real environment would have less chance of connection drops for the actual players. Figure 6.13 depicts two sets of transfers (for both T2T Time and T2R Time) during a single transfer, and shows that the time might be increased due to communication issues. It can be observed that the time taken is simply the average time (9Sec) × number of players and is quite static in normal situations.

Exp. No.	From Sim	To Sim	Region Name	Players	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	T2R Time (Sec)
1	Sim-I	Sim-II	A	1	8Sec	1Sec	2Sec	8Sec
2	Sim-I	Sim-II	A	5	43Sec	2Sec	3Sec	48Sec
3	Sim-I	Sim-II	A	10	88Sec	1Sec	2Sec	142Sec
4	Sim-I	Sim-II	A	15	132Sec	2Sec	2Sec	138Sec
5	Sim-I	Sim-II	A	20	183Sec	2Sec	3Sec	180Sec

Table 6.7: Summary of the timing information for the experiments transferring an increasing number of players on Windows environment.

6.6.3 Time Analysis on Linux Platform

In this section, we repeat the same sets of experiments (presented in section 6.6.2) on the Linux environment described in section 6.1.2. The main purpose is to see how the system behaves when dual quad core nodes are used to host different

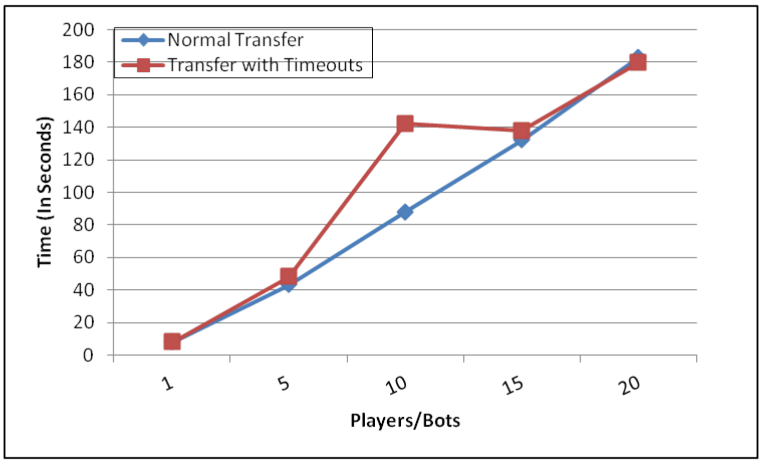


Figure 6.13: Time taken by teleport operation for transferring an increasing number of players on Windows environment.

Sims, and if there is any improvement in terms of time for different activities. One node is used to run the grid services and two are hosting a parent and a child Sim, called Sim-I and Sim-II respectively. To log in increasing number of Bots, we used additional nodes and a number of TClient instances.

Dynamic Content

Table 6.8 provides experimental results for transferring an increasing number of dynamic content (PandASs). It shows that SC, CR, and LC have similar outcomes as the Windows environment and they take only a few seconds. The time taken by both DR and CsR methods is decreased compared with corresponding experimental results on the Windows environment but they still take a considerable amount, of time which is up to a couple of minutes. The use of high speed nodes could only improve the performance to a certain level, due to the basic structure of both DR and CsR methods.

Example Worlds

The Linux environment for example worlds experiments failed to further decrease the time parameter for removing a region (both DR and CsR) compared with Windows environment and provided almost similar outcomes. This is due to the

Case	Exp. No.	From Sim	To Sim	Region Name	Prims and Active Scripts	SC Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)
Delete-Region	1	Sim-I	Sim-II	A	8Prims only	1Sec	2Sec	2Sec	1Sec
	2	Sim-I	Sim-II	A	101 each	1Sec	9Sec	1Sec	1Sec
	3	Sim-I	Sim-II	A	501 each	2Sec	41Sec	3Sec	3Sec
	4	Sim-I	Sim-II	A	1002 each	2Sec	98Sec	2Sec	3Sec
	5	Sim-I	Sim-II	A	1503 each	4Sec	128Sec	3Sec	5Sec
Close-Region	1	Sim-I	Sim-II	A	8Prims only	1Sec	1Sec	2Sec	1Sec
	2	Sim-I	Sim-II	A	101 each	2Sec	12Sec	2Sec	1Sec
	3	Sim-I	Sim-II	A	501 each	2Sec	57Sec	3Sec	3Sec
	4	Sim-I	Sim-II	A	1002 each	2Sec	148Sec	1Sec	3Sec
	5	Sim-I	Sim-II	A	1503 each	3Sec	227Sec	2Sec	4Sec

Table 6.8: Summary of timing information for experiments transferring dynamic content (PandASs) on Linux environment.

sequential approach for deleting scene objects from both scene and database. However, it greatly reduced the LC time, which is more than 60% less than the time taken by Windows nodes (see Table 6.6). The time taken by SC and CR parameters is almost the same as the Windows environment and is normally acceptable. The results of this set of experiments are provided in Table 6.9.

Case	Exp. No.	From Sim	To Sim	Region Name	Regional Content	SC Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)
Delete-Region	1	Sim-I	Sim-II	A	OpenCVE	6Sec	13Sec	2Sec	32Sec
	2	Sim-I	Sim-II	A	FairieCastle	7Sec	21Sec	1Sec	26Sec
	3	Sim-I	Sim-II	A	Maya Pyramid	4Sec	5Sec	1Sec	6Sec
	4	Sim-I	Sim-II	A	CSI World	5Sec	27Sec	2Sec	12Sec
	5	Sim-I	Sim-II	A	EducationSim	1Sec	18Sec	1Sec	4Sec
Close-Region	1	Sim-I	Sim-II	A	OpenCVE	7Sec	4Sec	1Sec	36Sec
	2	Sim-I	Sim-II	A	FairieCastle	8Sec	17Sec	1Sec	27Sec
	3	Sim-I	Sim-II	A	Maya Pyramid	3Sec	1Sec	2Sec	7Sec
	4	Sim-I	Sim-II	A	CSI World	6Sec	4Sec	1Sec	17Sec
	5	Sim-I	Sim-II	A	EducationSim	1Sec	1Sec	1Sec	5Sec

Table 6.9: Summary of timing information for experiments transferring example worlds content on Linux environment.

Interactive Clients

In this section, we repeat the experiments with an increasing number of players on our Linux environment. The main emphasis is to obtain the average time required to transfer a player from one region to another region. Table 6.10 presents the details of experiments and shows that each player on average takes about eight seconds. It gives a one second improvement (per player) on the Windows environment but we observed that there were hardly any disconnections for Bots. This is mostly due to the use of more TClient instances and using each to log in only a few Bots.

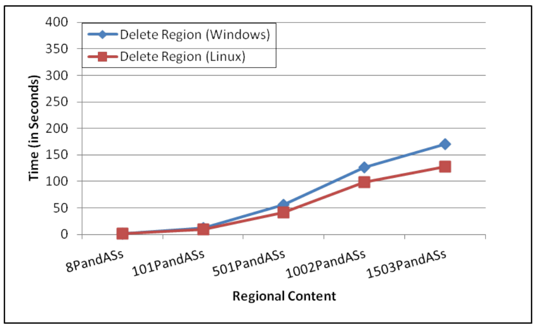
Exp. No.	From Sim	To Sim	Region Name	Players	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	T2R Time (Sec)
1	Sim-I	Sim-II	A	1	7Sec	1Sec	1Sec	8Sec
2	Sim-I	Sim-II	A	5	40Sec	3Sec	2Sec	38Sec
3	Sim-I	Sim-II	A	10	81Sec	2Sec	1Sec	78Sec
4	Sim-I	Sim-II	A	15	118Sec	2Sec	1Sec	121Sec
5	Sim-I	Sim-II	A	20	158Sec	2Sec	2Sec	162Sec

Table 6.10: Summary of timing information for experiments transferring increasing number of Bots on Linux environment.

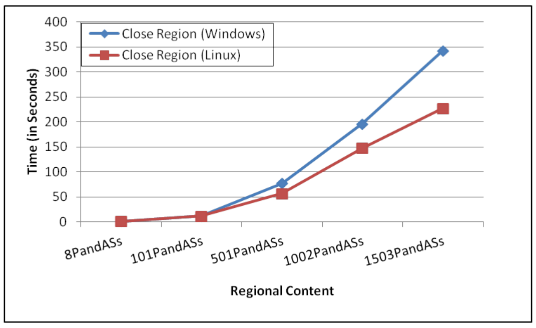
6.6.4 Comparison and Discussion

In this section, we presented a number of experiments on both Windows and Linux platforms for three different categories of load: an increasing amount of dynamic content (PandASs), the example worlds content (described in section 6.1.3), and an increasing number of interactive players. The same sets of experiments were used to compare the performance of both systems. This section only presents a comparison of those parameters where the Linux environment performs better than the Windows environment.

The experimental results for dynamic content on the Windows environment show that both DR and CsR methods take a considerable amount of time. The same set of experiments on the Linux environment for both operations obtains better results, although they still consume a large amount of time. The comparison of DR



(a)

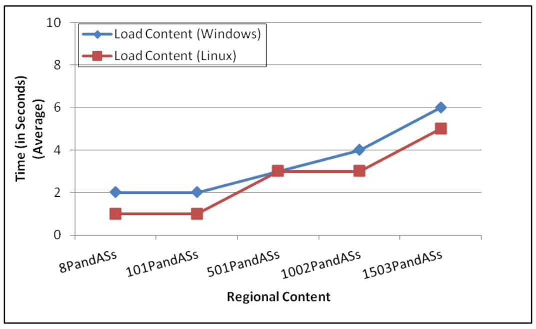


(b)

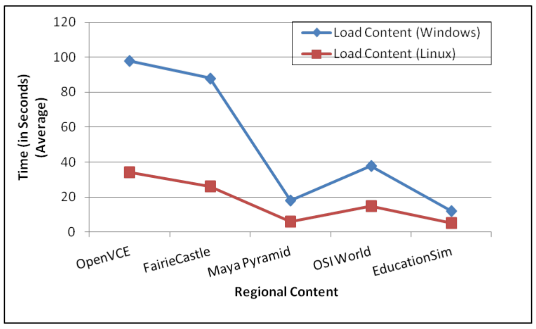
Figure 6.14: Comparison of timing information of dynamic content (PandASs) on both Windows and Linux environments for (a) Delete Region (DR), and (b) Close Region (CsR).

and CsR for both Windows and Linux environments for dynamic content is shown in Figure 6.14. However, both environments showed an almost similar behaviour against the static content with a slight improvement of the Linux environment on Windows environment. The basic reason is that DR follows a sequential approach to deleting the content from both a scene and database in turn. Similarly, the CsR method uses the persistence mechanism to store the updated content before closing it. It is observed that DR time increases with a rise in the amount of static content. However, static content requires no periodic backups and, therefore, CsR takes only a few seconds to remove a region. To reduce the overall transfer time for a region, the DR time needs to be minimised.

The transfer experiments on the Windows environment for example worlds content identified that the LC activity is also a time-consuming activity and needs a great deal of time to load and setup regional content. The Linux node greatly improved the time taken by LC for the same set of experiments. The comparison of both environments for the LC against dynamic content (PandASs) and example worlds content is presented in Figure 6.15. We believe that allowing content based on dynamic content, the LC time is much less than the time taken by basic DR and CsR methods. If a scene has complex objects, it needs an increased amount of time to load the content. We believe that OAR functionality hides the complex operations to set up regional content. On the other hand, it is easy to significantly minimise the time taken by DR and CsR by using OAR functionality that could



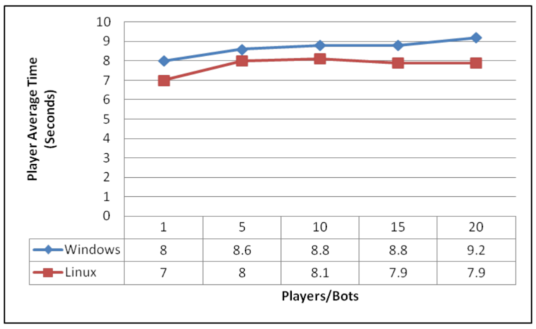
(a)



(b)

Figure 6.15: Comparison of timing information of Load Content (LC) on both Windows and Linux environments for (a) dynamic content, (b) example worlds content.

be used to back up regional data, thus potentially avoiding the basic persistence process of objects into a database.



	1	5	10	15	20
Windows	8	8.6	8.8	8.8	9.2
Linux	7	8	8.1	7.9	7.9

Figure 6.16: Comparison of timing information for increasing number of players transfer on both Windows and Linux environments.

We also performed a number of experiments for transferring players from one region to another to determine the average time taken by each player. A player transfer on the Windows environment takes on average approximately nine seconds, compared with approximately eight seconds on the Linux environment, as shown in Figure 6.16. The Linux system slightly improved the average player transfer time but we might need to investigate other ways to reduce it further in future. We believe that, though it takes a little while to transfer a player, it gives better experience than simply freezing them.

Based on this discussion, in the next section we present two strategies that significantly minimise the total time taken by a region transfer compared with original OS methods.

6.7 Improved Strategies

In this section, we introduce a number of potential ways to improve transfer time for a region by applying better strategies for different activities during a transfer. It presents two improved strategies targeting region removal from a Sim, and provides a comparison based on time parameters for dynamic content (PandASs) and example worlds content on both Windows and Linux environments.

6.7.1 Introduction

Two basic methods are available in both the OS framework and RAd functionality to remove a region from a Sim: Delete Region (DR) and Close Region (CsR)(is remove-region in OS framework). The former generally takes longer but removes all the data from both scene and database before closing down its client server. The latter closes both the scene and server but does not remove data from the database. DR for a large set of content (both static and dynamic) takes a long time to delete a region due to its sequential approach to deleting the content from both scene and database. On the other hand, CsR for dynamic content takes a great deal of time due to the backup process to store the most recent changes to the content.

Using a centralised database for managing data for all region Sims can potentially provide a more efficient way to load content when used together with CsR, thus avoiding storing the content to an OAR file. It takes advantage of grid services that manage all the components through unique identifiers. Even if the system allows loading the existing content to a region when it is created with the same specifications on a different Sim, it has a number of key limitations. Firstly, it is only beneficial when most of the content is static, thus avoiding the persistence process, and it closes a region promptly. However, we believe that VWs provide both static and dynamic content. Similarly, they allow users to create and modify their content that might require being stored before closing a region. Secondly, a

centralised system is also a potential bottleneck for large scale VWs.
On the other hand, LC also takes a considerable amount of time, though greatly reduced by the Linux environment. However, the OAR functionality we used provides an easy and handy way to transfer a region compared with other methods that might require complex explicit operations. If we wanted to apply data transfer between databases of different Sims, we need the basic persistence process to first store updated content in a region. Moreover, the process of transferring through database migration might be complex in the sense that it needs to transfer other relevant information from different tables and stores. The OAR functionality makes it much easier to restore the complete terrain, assets and prims to a region.
Based on these issues, and to help develop VWs with a reasonable amount of both static and dynamic content, we developed two strategies which perform much better than the basic DR and CsR methods, especially if a region has dynamic content. They skip the persistence process of OS framework and take advantage of OAR to relocate a region with the latest content. However, they use direct queries to database for cleaning up the data. These strategies take the advantage of OAR functionality to obtain a backup of regional data, thus skipping the persistence step in CsR with explicit queries to clean up the database.

6.7.2 Improved Strategies

We have written additional procedures while keeping the original basic methods for their general use. These strategies use direct database operations to perform data deletion and cleanup activities which are performed implicitly by DR method. These improved strategies are described below:
The **first Improved strategy** (Improved-I) deletes the objects of a regional scene and relevant entries from the database. It then calls the delete-region method of RAd functionality to remove the region, thus breaking up the sequential cycle of eliminating objects from both scene and database in sequence.
The **second Improved strategy** (Improved-II) removes a region with persistence operation being disabled and then explicitly deletes the relevant entries from the database directly. The object persistence is replaced by storing regional data in OAR files for transferring updated content.

6.7.3 Time Analysis and Comparison

In this section, we present a comparison of the basic DR and CsR methods together with our improved strategies. We have performed experiments with both dynamic content (PandASs) and example worlds on both Windows and Linux environments. We have already demonstrated that the Linux environment performs better than Windows for certain activities. For this comparison, we have reproduced only a subset of experiments that we performed using our informal time analysis model in section 6.6.

Table 6.11 presents the experimental results of an increasing number of dynamic content for both Windows and Linux environments. It is clear that the basic DR and CsR operations take much longer as the number of content increases. Although, the Linux environment is faster than the Windows environment, it still takes considerable amount of time. Improved-I greatly reduced the time required to delete the content after breaking the sequential cycle; however, the best results are achieved by Improved-II which minimised the removal time to just a few seconds. However, it could be seen that for a region with very little content, the basic DR takes less time than our improved strategies. Nevertheless, we believe that the regions normally have a reasonable amount of content (both static and dynamic) and, therefore, we recommend Improved-II in most of the cases.

Environment	Exp No.	Regional Content	Basic RAd Delete-Region	Basic RAd Close-Region	Improved-I	Improved-II
Windows	1	0PandASs	1Sec	1Sec	2Sec	2Sec
	2	100PandASs	12Sec	14Sec	3Sec	2Sec
	3	500PandASs	56Sec	77Sec	5Sec	3Sec
	4	1000PandASs	126Sec	196Sec	15Sec	4Sec
	5	1500PandASs	170Sec	343Sec	26Sec	4Sec
Linux	1	0PandASs	2Sec	1Sec	1Sec	1Sec
	2	100PandASs	9Sec	12Sec	2Sec	2Sec
	3	500PandASs	41Sec	57Sec	6Sec	2Sec
	4	1000PandASs	98Sec	148Sec	11Sec	3Sec
	5	1500PandASs	128Sec	227Sec	22Sec	3Sec

Table 6.11: Comparison of different Delete Region (DR) and Close Region (CsR) strategies for dynamic content on both Windows and Linux environments.

Experimental results for example worlds content are presented in Table 6.12. It is of note that both Windows and Linux systems perform almost the same for static content. It can be seen that it takes more time to remove FairieCastle due to dynamic entities included in its content. The rest of the worlds are removed in a few seconds. Both the improved strategies guarantee the removal of content from the database and achieve better results than the DR method. Figure 6.17 shows a comparison of four strategies on the Windows environment for both dynamic content and example worlds. These strategies are compared for both sets of content on the Linux system with the help of Figure 6.18.

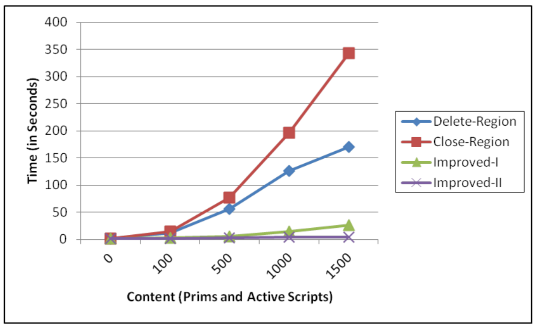
Environment	Exp. No.	Regional Content	BasicRAd Delete-Region	Basic RAd Close-Region	Improved-I	Improved-II
Windows	1	Our world	1Sec	1Sec	2Sec	2Sec
	2	OpenCVE	13Sec	5Sec	4Sec	3Sec
	3	FairieCastle	19Sec	25Sec	5Sec	3Sec
	4	Maya Pyramid	5Sec	2Sec	2Sec	2Sec
	5	CSI World	25Sec	3Sec	3Sec	3Sec
	6	EducationSim	17Sec	3Sec	2Sec	2Sec
Linux	1	Our world	1Sec	1Sec	2Sec	2Sec
	2	OpenCVE	13Sec	4Sec	2Sec	2Sec
	3	FairieCastle	21Sec	17Sec	4Sec	3Sec
	4	Maya Pyramid	5Sec	1Sec	2Sec	2Sec
	5	CSI World	27Sec	4Sec	5Sec	2Sec
	6	EducationSim	18Sec	2Sec	3Sec	2Sec

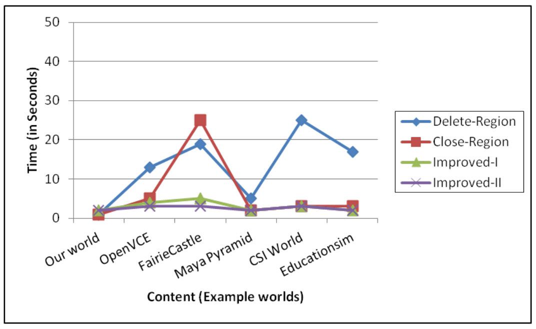
Table 6.12: Comparison of different Delete Region (DR) and Close Region (CsR) strategies for the example worlds on both Windows and Linux environments.

6.8 System Issues/bugs and fixtures

In this section, we report a number of bugs in the OS framework that we fixed to achieve the desired functionality.

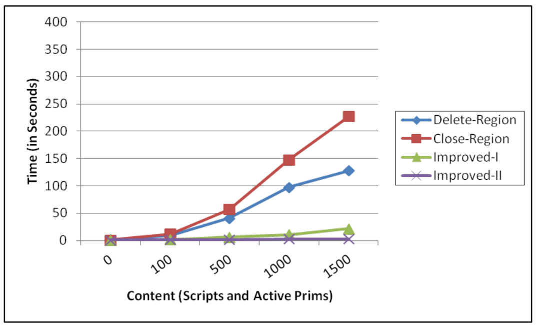
Since we initially used region console to investigate the capabilities of OS framework, we found that the basic RR method was deleting a region instead of its fundamental functionality of closing it. This issue was fixed by calling the correct procedures to achieve the desired functionality. For our implementation work, we

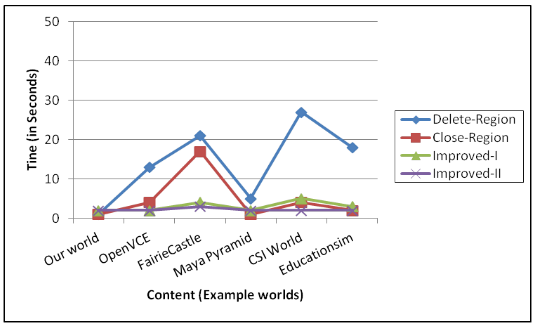




(a) (b)

Figure 6.17: Comparison of different methods to remove a region from a Sim on Windows environment for (a) dynamic content, (b) example worlds.





(a) (b)

Figure 6.18: Comparison of different methods to remove a region from a Sim on Linux environment for (a) dynamic content, (b) example worlds.

used the RAd functionality that provides the correct functionality of CsR (which is RR in OS).

While experimenting with our informal model we noticed that, when a region is deleted from a Sim, the database entry that links a region to an estate is not cleared. Therefore, it prevented us from re-creating a region with the same specification while relocating them. We fixed this by writing an explicit query to delete the corresponding entry from a regional database.

6.9 Conclusions and Future Work

We conducted a number of tests to determine the robustness of the OS capabilities and used a number of time statistics to investigate and compare different aspects of this work.

We presented a load model based on SimFPS parameter after a wide range of experiments for static and dynamic content, as well as interactive clients, on both Windows and Linux environments. We used a minimum of 30FPS and 40FPS for SimFPS to determine SimCapacity and SplitCapacity consecutively. A value of 30FPS is used to guarantee better user experience, while 40FPS for SimFPS avoids bad experience by initiating a split much earlier than reaching a point that might degrade performance. The system takes decisions based on average values which are calculated by considering a number of observations in the last few minutes. The number of players for both Windows and Linux remained the same, and this showed that due to the current software implementation, additional hardware was unable to scale the world.

We presented the scalability model and determined the components that are used to implement both split and merge operations. We started with an informal implementation framework and conducted a study of three sets of experiments based on dynamic content, example worlds content, and interactive players. OS allows different database options (SQLite and MySQL) and orientations (centralised and localised). We investigated both SQLite and MySQL databases for standalone mode and examined MySQL for grid mode using both centralised and localised orientations. To reduce communication overhead and avoid longer delays in scalable systems, a local MySQL server was determined to be an excellent choice. Furthermore, it was determined that a Sim requires to be managed by a dedicated server for improved performance.

The Windows platform revealed that both Delete Region (DR) and Close Region (CsR) are time-consuming activities to remove a region from a Sim and load content from an OAR file. Furthermore, a player transfer (using teleport method) on average takes approximately nine Seconds. The Linux environment was significantly faster for loading the content, which is approximately a 60% improvement over the Windows environment. The average player transfer time is improved by

a second, taking approximately eight seconds. DR and CsR are improved significantly but they still take a considerable amount of time. To reduce the time taken by DR and CsR methods on both environments, we presented two improved strategies. They used direct database access to delete regional data. In addition to this, the best strategy (Improved-II) uses CsR instead of DR together with by-passing the backup process to get a significant improvement over the basic methods. We conducted an experimental study for their timings compared with both DR and CsR operations.
We found a number of bugs and fixed them to ensure that the system worked according to our requirements by writing explicit routines, including extended DR and RR methods. Since VWs allow users to create and modify content, we believe that world content are mostly based on a reasonable amount of both static and dynamic content. We therefore suggest using the Improved-II strategy that performs well in all situations.

The following points can be further investigated:
Our load model is based on the SimFPS parameter due to a similar impact on PhysicsFPS. Furthermore, performance tests (targeting physics) could be conducted to identify the response of other parameters that might help to extend the current load model for different requirements.
Future work might investigate the reasons why there is no improvement in system capacity with additional resources.
We used the OAR functionality to transfer regional content which greatly reduced the time taken by traditional methods to remove a region. However, content load still takes a considerable amount of time, depending on scene complexity. Other methods could be explored to further reduce the content load time.

Chapter 7

Scalable Virtual Worlds: Implementation

This chapter examines an abstract framework for the implementation of JoHNUM infrastructure and ARA algorithm for developing scalable VWs using an extension to the OS framework. It expands with an increase in the number of players by using additional resources, and it shrinks with a decrease in the number of players by releasing under-used resources. The ARA algorithm is extended to achieve contiguous areas of both square and non-square shaped regional grids for assignment. It presents motivation for our implementation work and gives a detailed analysis of different sets of experiments for regional grids of four and nine regions. It uses modified versions of example worlds content to populate regions and a number of different metrics were used for the evaluation and comparison of this work.

7.1 Introduction

In this section, we present the limitations of existing VWs and the motivation for the framework developed in this chapter, based on the investigations presented in chapter 6. It provides a description of VWs used for the experiments conducted in this chapter. It describes Sims and provides descriptions of the regions used for regional grids. It also provides extended versions of example worlds and statistics that are used for evaluation and comparison purposes.

7.1.1 Background and Motivation

In general, VWs use spatial partitioning to share the load with additional servers [132]. However, it is an expensive operation that transfers both content and players and requires special attention to reduce the time taken by a distribution process. The modular design of OS framework can be exploited to develop strategies for achieving this goal. Its potential to host bigger spaces and transfer regions in turn greatly motivated us to use the concept of spatial partitioning. It has the potential to reduce content un-availability time, and the total time taken by a transfer. It also reduces the number of players that suffer from a transfer, compared with traditional methods. The basic goal is to cope with issues in both static and dynamic approaches. Other motivating factors for combining OS regions into a bigger space include the fact that, with a few exceptions, most SL regions are never visited or are visited by very few people [219]. Based on results of previous studies targeting spatial partitioning, we adopted a different approach. Instead of freezing the players, we transfer them to a transit region while relocating their current region. Even though it takes a few minutes, we believe that our approach is acceptable for the following reasons: it is a rare operation and merging uses a very relaxed strategy to re-integrate regions thus avoiding frequent splits; though players are unable to add or modify content during splits, they normally have a better experience; and since we are teleporting players explicitly, they will observe the disappearance and then re-appearance of the players around them. However, players are warned about "maintenance" work before starting a region transfer. In the worst cases, a region might have twenty players and each takes between eight and nine seconds to transfer.

Initially, each Sim is assigned a bigger continuous space made up of a number of regions in a flat orientation. When it reaches an excessive load, the load is shared with additional Sims to scale the world. On the other hand, the system revokes under-utilised resources. Resources are, therefore, used according to the requirements that potentially overcome most of the issues. Due to the unique characteristics of VWs, the traditional optimisation techniques such as sharding

cannot be applied. However, we have developed better strategies to reduce the total time taken by a transfer. Our current ARA algorithm results in fair distribution of load and, in the case of excessive load, we obtain a new Sim to share the load with. However, this work does not look into load balancing, but this is our future work to be carried out. We believe that our merging strategy (called Child Merge) has the potential to achieve load balancing, which is presented in section 7.5 and demonstrated in section 7.6. In this chapter, our basic aim is to develop an abstract framework and to show how it achieves the goals set for this work by implementing it.
According to Liu et al. [140], the main scalability barriers constrained by resources include CPU Utilisation, Network bandwidth, and Network Latency. The CPU load could increase dramatically as a VW scales up in any direction. With an increase in the number of concurrent users or scene complexity, the network can quickly become overwhelmed by combined traffic. Furthermore, a walk-through system requires visualisation updates much faster than a truly responsive system. According to Lake et al. [132], when the number of connected clients increases, the frame rate begins to decrease. Lag appears in physics and network processing. If further clients are added, the frame rate quickly drops to a point where the scene is not usable. To avoid these issues, and to achieve improved performance, we have taken certain decisions based on the load model developed and described in chapter 6. We suggest a reasonable amount of static and dynamic content and interactive players in a Sim for which the system maintains at least 30FPS for SimFPS. In case additional Sims are available and a Sim is managing more than a single unit region, it initiates a split when the SimFPS reaches 40FPS. These decisions overcome those situations that are normally managed by allowing a system to accept plausible results or slow down execution to reduce the performance gap that results in a degraded user performance.

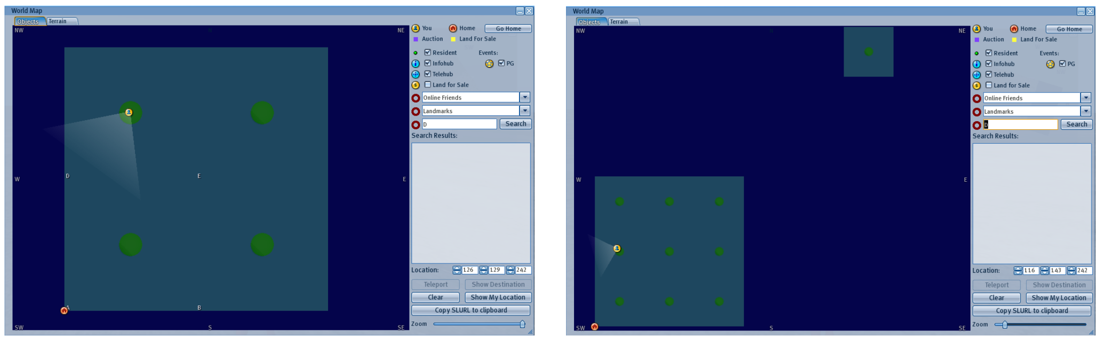
7.1.2 VW Environment and Setup

In this section, we provide a description of the regions that are used to constitute regional grids of different shapes and sizes with an upper bound of nine regions. However, it is important to mention that much bigger worlds can be used to begin

Simulator (Sim)	HTTP Listening Port	Regions (Region: XY Co-ordinate:Port)	Transit Region(Region: XY Co-ordinate:Port)
Sim-I (Parent)	9000	A:1000,1000:9000 B:1001,1000:9001 C:1002,1000:9002 D:1000,1001:9003 E:1001,1001:9004 F:1002,1001:9005 G:1000,1002:9006 H:1001,1002:9007 I:1002,1002:9008	T1:1005,1005:9105
Sim-II (Child)	9005	Nil	T2:1006,1006:9106
Sim-III (Child)	9010	Nil	T3:1007,1007:9107
Sim-IV (Child)	9015	Nil	T4:1008,1008:9108
Sim-V (Child)	9020	Nil	T5:1009,1009:9109
Sim-VI (Child)	9025	Nil	T6:1010,1010:9110
Sim-VII (Child)	9030	Nil	T7:1011,1011:9111
Sim-VIII (Child)	9035	Nil	T8:1012,1012:9112
Sim-IX (Child)	9040	Nil	T9:1013,1013:9113

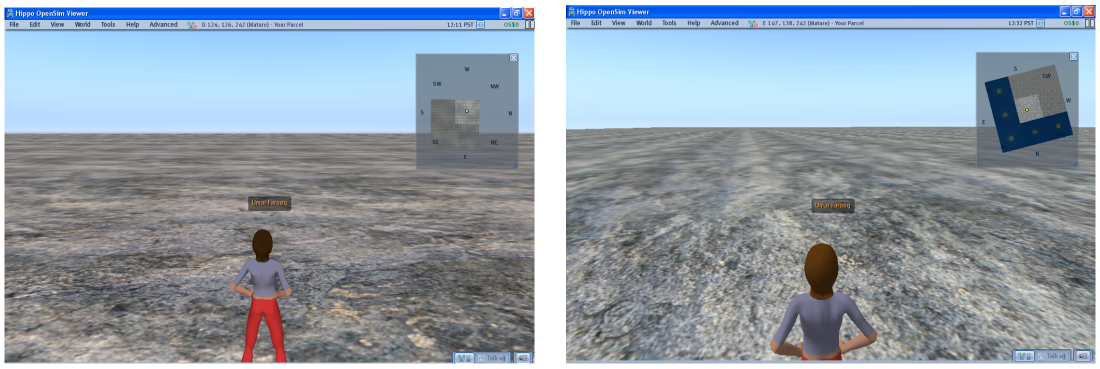
Table 7.1: Description of 9 Sims with their transit regions and 9 content regions.

with, based on system capabilities. Table 7.1 provides details of nine Sims, each with a local transit region. Sim-I is the parent Sim and it initially hosts all the regions in a VW used for an experiment. We have used square-shaped regional grids of four and nine regions in our current experimental work, and adjacent regions from Table 7.1 are used to obtain greater contiguous spaces, as shown in Figure 7.1. Figure 7.1(b) shows the configuration of our parent Sim (Sim-I) with its local transit region. Our **implementation model assumes** that a VW is pre-partitioned into the number of regions in a grid and does not actually follow our theoretical split model. It is based on the fact that dividing into more smaller regions achieves further improvements. Each region is taken as a unit region that is considered by ARA algorithm during the aggregation process. They cannot be further divided and, hence, all Sims are directly connected to the parent Sim in a RMT of a single additional level. Transit regions are only used during split and merge operations and provide no real world content. Figure 7.2 presents regional grids of four and nine regions and shows the visibility of regions to a player based on their current location. Each player knows all the neighbouring



(a) (b)

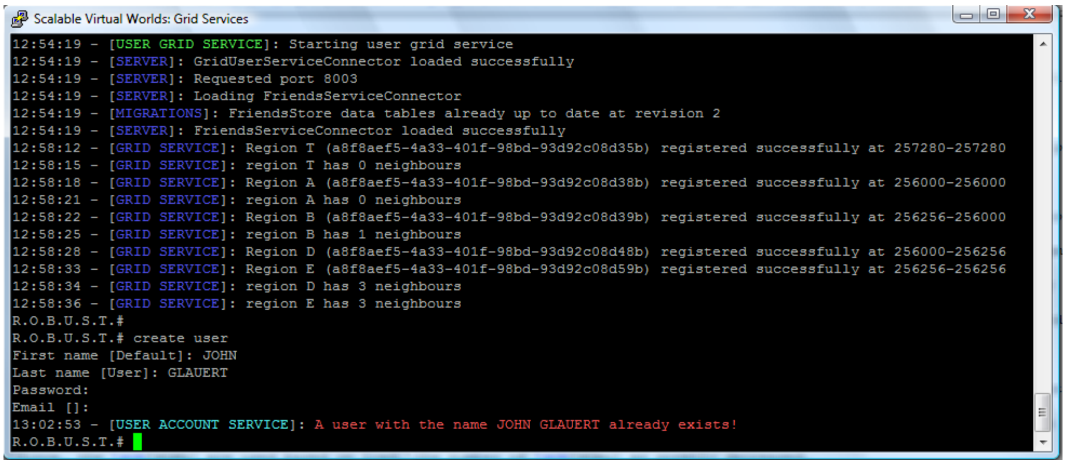
Figure 7.1: World map showing adjacent placement of regions to get contiguous spaces for (a) 4 regions grid, and (b) 9 regions grid with a transit region.



(a) (b)

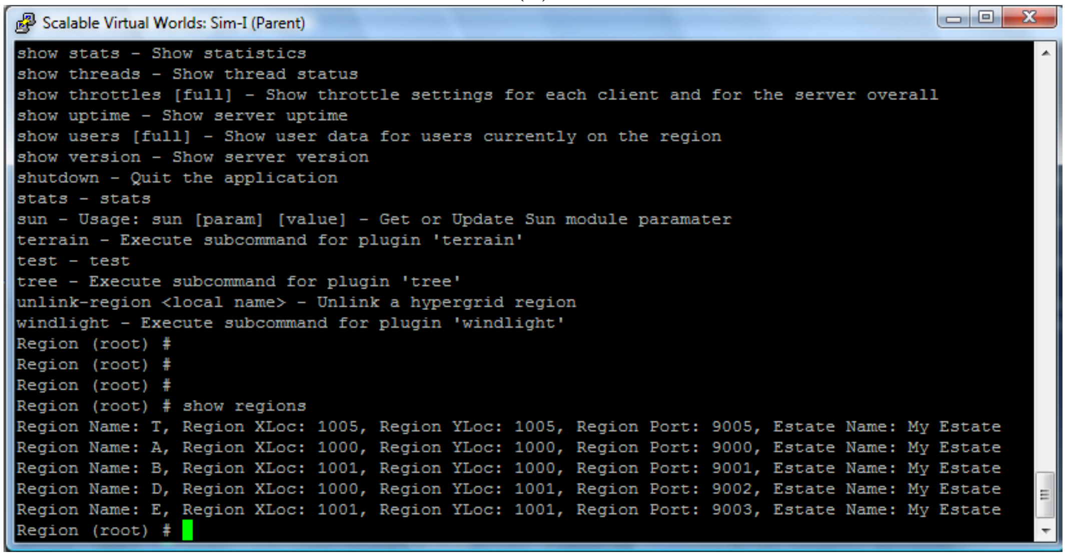
Figure 7.2: Description and visibility of regions to a player in a regional grid of (a) 4 regions, and (b) 9 regions.

regions and a presence for it is added to adjacent regions for smooth boundary crossings. For each experiment, child Sims are running with transit regions in advance and waiting for the load to share with the parent Sim. The **Content** used to populate regions is presented in section 7.1.4. A **Robust instance** is used to provide grid services and integrate the VW. In grid mode, each region on a Sim requires to be registered with the grid for its global presence. Figure 7.3(a) shows a Robust console with messages showing registered regions and user creation. Each region server allows different management functions to be performed from a region console, as shown in Figure 7.3(b) for Sim-I.



```
12:54:19 - [USER GRID SERVICE]: Starting user grid service
12:54:19 - [SERVER]: GridUserServiceConnector loaded successfully
12:54:19 - [SERVER]: Requested port 8003
12:54:19 - [SERVER]: Loading FriendsServiceConnector
12:54:19 - [MIGRATIONS]: FriendsStore data tables already up to date at revision 2
12:54:19 - [SERVER]: FriendsServiceConnector loaded successfully
12:58:12 - [GRID SERVICE]: Region T (a8f8aef5-4a33-401f-98bd-93d92c08d35b) registered successfully at 257280-257280
12:58:15 - [GRID SERVICE]: region T has 0 neighbours
12:58:18 - [GRID SERVICE]: Region A (a8f8aef5-4a33-401f-98bd-93d92c08d38b) registered successfully at 256000-256000
12:58:21 - [GRID SERVICE]: region A has 0 neighbours
12:58:22 - [GRID SERVICE]: Region B (a8f8aef5-4a33-401f-98bd-93d92c08d39b) registered successfully at 256256-256000
12:58:25 - [GRID SERVICE]: region B has 1 neighbours
12:58:28 - [GRID SERVICE]: Region D (a8f8aef5-4a33-401f-98bd-93d92c08d48b) registered successfully at 256000-256256
12:58:33 - [GRID SERVICE]: Region E (a8f8aef5-4a33-401f-98bd-93d92c08d59b) registered successfully at 256256-256256
12:58:34 - [GRID SERVICE]: region D has 3 neighbours
12:58:36 - [GRID SERVICE]: region E has 3 neighbours
R.O.B.U.S.T.#
R.O.B.U.S.T.# create user
First name [Default]: JOHN
Last name [User]: GLAUERT
Password:
Email []:
13:02:53 - [USER ACCOUNT SERVICE]: A user with the name JOHN GLAUERT already exists!
R.O.B.U.S.T.#
```

(a)



```
show stats - Show statistics
show threads - Show thread status
show throttles [full] - Show throttle settings for each client and for the server overall
show uptime - Show server uptime
show users [full] - Show user data for users currently on the region
show version - Show server version
shutdown - Quit the application
stats - stats
sun - Usage: sun [param] [value] - Get or Update Sun module paramater
terrain - Execute subcommand for plugin 'terrain'
test - test
tree - Execute subcommand for plugin 'tree'
unlink-region <local name> - Unlink a hypergrid region
windlight - Execute subcommand for plugin 'windlight'
Region (root) #
Region (root) #
Region (root) #
Region (root) # show regions
Region Name: T, Region XLoc: 1005, Region YLoc: 1005, Region Port: 9005, Estate Name: My Estate
Region Name: A, Region XLoc: 1000, Region YLoc: 1000, Region Port: 9000, Estate Name: My Estate
Region Name: B, Region XLoc: 1001, Region YLoc: 1000, Region Port: 9001, Estate Name: My Estate
Region Name: D, Region XLoc: 1000, Region YLoc: 1001, Region Port: 9002, Estate Name: My Estate
Region Name: E, Region XLoc: 1001, Region YLoc: 1001, Region Port: 9003, Estate Name: My Estate
Region (root) #
```

(b)

Figure 7.3: Console window for a (a) Robust instance (for grid management), and a (b) Region server (for managing regions).

7.1.3 Platforms

For the evaluation of our implementation work, we use both Windows and Linux environments, as described in section 6.1.2 of chapter 6, with the following configurations.

Windows

The Grid services (Robust instance), together with an instance of MySQL database for grid, are running on a dual core system. Two Pentium systems are used to run Sims (a parent (Sim-I) and a child (Sim-II)) and their regional database instances. The dual core system is used together with a Pentium system to populate the VW with interactive clients.

Linux

Since all Linux nodes have the same capabilities, we use one node to run grid services and an instance of MySQL database for its data use. We have used up to nine Sims, each running a local database instance and a region server. Regional grid description is given in Table 7.1, which provides the specifications of Sims. This allowed us to use many other nodes to log-in bots to the VW.

7.1.4 Content and Players

The content used to populate regions during our experiments are extended versions of the example worlds content described in section 6.1.3 of chapter 6, and are detailed in Table 7.2 for the corresponding regions. We added an additional 500 PandASs to each content and measured them on both Windows and Linux environments to determine timing information for different activities. The time taken by SC and LC operations did not increase. However, there was a significant increase in the time taken by both DR and CsR methods to remove a region from a Sim whose average values are provided in Table 7.2. Experiments described in this chapter use Improved-II strategy to remove a region, which greatly reduces the overall time taken by a transfer. Even though the Windows systems take much longer for the LC operation, we still prefer to use OAR functionality for content transfer for the following reasons: firstly, removing a region with a very little content (see Table 7.2) using basic operations also takes a considerable amount of time; secondly, it hides the complexity behind the process. Our decision to use OAR functionality is justified on Linux systems as we observe a substantial decrease in content load time.

Region Name	Content	Windows Platform		Linux Platform	
		DR Time	CsR Time	DR Time	CsR Time
A	OpenVCE + 500 PandASs	78Sec	91Sec	63Sec	73Sec
B	CSI World + 500 PandASs	93Sec	90Sec	78Sec	71Sec
C	Educasim + 500 PandASs	84Sec	92Sec	68Sec	75Sec
D	FairieCastle + 500 PandASs	88Sec	118Sec	64Sec	104Sec
E	Maya Pyramid + 500 PandASs	68Sec	87Sec	51Sec	68Sec
F	Our world + 500 PandASs	59Sec	81Sec	45Sec	60Sec
G	Our world + 500 PandASs	‘’	‘’	‘’	‘’
H	Our world + 500 PandASs	‘’	‘’	‘’	‘’
I	Our world + 500 PandASs	‘’	‘’	‘’	‘’

Table 7.2: Regional content for 9 regions used in our experiments

Instead of using actual players, we have used Bots to populate VWs. This uses TClient as described earlier in section 6.1.3 of chapter 6 and each instance is used to log in multiple Bots. To avoid issues, we logged-in a maximum of ten bots per TClient. For each log in, system requires a unique user account and, therefore, we created an account for each Bot. We used simple TClient commands to develop different scripts that are executed by the Bots in sequence, using the same TClient instance. To show an increase in the number of inter-sim crossings as a system scales, we use a script called ScriptT. Using a random strategy, each Bot makes a random move to a randomly selected region including its current region in the grid using teleports. They are then asked to return to their original regions. Based on Bots movements, we determine the total number of times players crossed Sim boundaries.

7.1.5 Statistical Metrics

We used different parameters to show the effectiveness of our dynamic framework, and these are categorised as Time Statistics and System Statistics. Most of the time statistics were defined in section 6.4.3 of chapter 6, and here we only define those parameters that were not previously defined.

Time statistics are based on the time each activity, as part of a transfer, takes. It includes SC Time, T2T Time, RR Time, CR Time, LC Time, T2R Time, Region Content Transfer Time (RCT Time), Region Un-availability Time (RUnAv Time), Region Transfer Time (RT Time), Total Time, and Average Player Time.
Region Content Transfer Time (RCT Time) is the time taken by content transfer (of a region) excluding players' transfer time.
Region Un-availability Time (RUnAv Time) is the time when a region is locked for a transfer until it is set up on a destination Sim. It includes the time taken by transferring players from the prospective region to the local transit region. The time that is required to transfer both content and players is called **Region Transfer Time (RT Time)**. It includes both transferring players to transit and then from transit to actual region after relocation.
Total Time is the cumulative time taken by all regions transferred, being part of an aggregate.

System statistics include Content transferred, Number of players transferred, Number of regions migrated, Number of resources used, Number of Concurrent users, Number of inter-sim crossings, Sim utilisation, Player disruption, and Transfers per player.
Number of regions migrated shows the number of regions in an aggregate being transferred.
Number of resources used is the total number of used resources.
Number of Concurrent users shows the total number of concurrent users in a grid served by a set of Sims.
Number of inter-sim crossings shows the total number of connections and disconnections between Sims for all users over the grid.

Sim utilisation represents the average capacity of each Sim.
Player disruption means the total number of connections/disconnections per player.
Transfers per player is the total number of times a player is transferred.

7.2 Abstract Scalability Framework

In this section, we provide an abstract framework by extending our scalability model presented in chapter 6. The basic aim is to present how our prototype implements both scaling and merging processes and how automatic decisions to initiate a split or a merge are taken by each Sim. This framework is implemented as a plug-in application in C# which works well on both Windows and Linux environments. It uses grid mode of OS where the implementation of the activities take the benefit of the integrated grid services. The basic steps in the main controlling module and both distribution and merging processes are presented with the help of Algorithms 4, 5, and 6. Each Sim is either a parent or a child handling part of the world. Modules in the framework are defined in terms of simple activities. We provide the abstract logic of main modules that use different methods of RAd functionality to implement them, details of which are available at [186] but are not discussed further. However, explicit procedures are briefly described.
The main procedure that is taking split and merge decisions based on the values of SplitCapacity and MergeCapacity is the heart of our framework. Each Sim continuously monitors the load and applies the logic presented in algorithm 4 until the simulation is running. We maintain a pool of servers each running a Sim (described in Table 7.1) that are available to share the load with overloaded Sims. If a system load exceeds SplitCapacity, it potentially initiates DistributeLoad module (algorithm 5) when a split is possible. It determines a list of scenes by calling the ARA algorithm (presented in section 3.2.2 of chapter 3), and then it calls DistributeLoad module to transfer regions to a newly selected Sim. On the other hand, if the system load is less than, or equal to, MergeCapcity and a merge is possible, it potentially initiates MergeLoad module (algorithm 6) to integrate its load with a Sim, including parent among the existing Sims. Merging is only ini-

tiated by child Sims in our current implementation. It selects a Sim for merging if, and only if, they maintain two constraints. Firstly, their combined load needs to be less than, or equal to, MergeCapacity and secondly, their regions must get a contiguous space. If both conditions are satisfied, the Sim initiates the MergeLoad function with the Sim-ID (represented as RemoteSimulator) of the Sim to merge its load with. MergeLoad then follows the steps outlined in algorithm 6 to transfer its load. Our current implementation applies all or none strategy for merging because we are interested in reducing the number of resources and not achieving a uniform load. However, a transit region is never transferred during a split or merge operation.

Algorithm 4 The Scalability Framework Components: Main module

```
1: while (VW is running) do
2:     if (system load >= SplitCapacity and split is possible) then
3:         List<Scene> RegionsToTransfer = DetermineAggregatedRegions(); //uses ARA algorithm
4:         DistributeLoad(RegionsToTransfer);
5:     else
6:         if (system load <= MergeCapacity and Merge is possible) then
7:             Uri RemoteSimulator = GetSimulator(system load);
8:             MergeLoad(RemoteSimulator);
9:         end if
10:    end if
11: end while
```

Algorithm 5 The Scalability Framework Components: DistributeLoad Module

```
Require: RegionsToTransfer //List of regions to transfer
1: Get a RemoteSimulator;
2: for (int i = 0; i < RegionsToTransfer.Count; i++) do
3:     Warn the players about maintenance to begin;
4:     Lock the region;
5:     Save content to an oar file at shared location;
6:     Get region specification;
7:     if (RegionsToTransfer[i] has players) then
8:         Transfer players to transit region;
9:     end if
10:    Remove region;
11:    Create region on RemoteSimulator; //provide specification
12:    Load content from oar file to the region on RemoteSimulator;
13:    if (transit region has players) then
14:        Transfer the players back to the region on RemoteSimulator;
15:    end if
16: end for
```

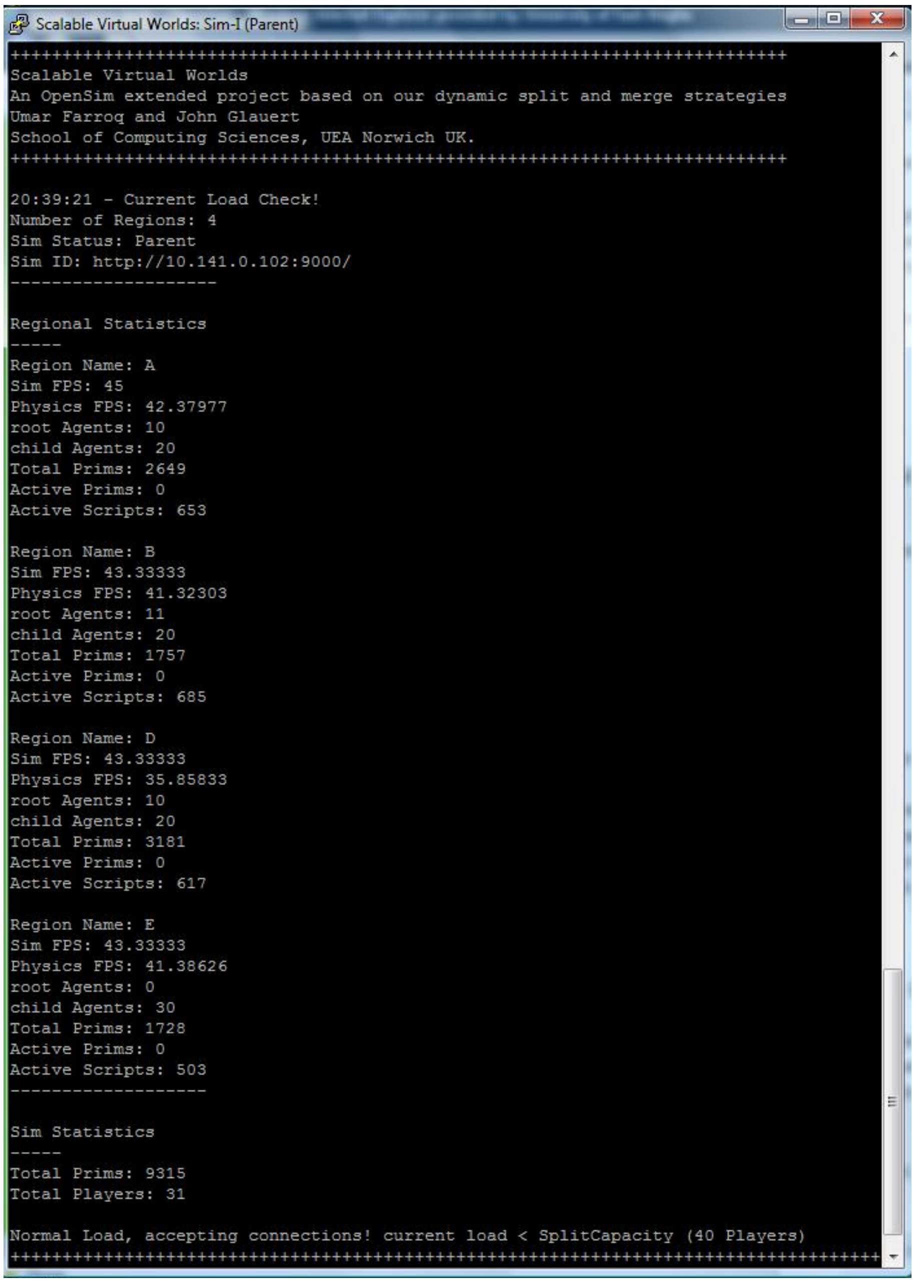
Both DistributeLoad (algorithm 5) and MergeLoad (algorithm 6) modules use the same set of actions in a sequence and transfer the regions in turn. Players in a

Algorithm 6 The Scalability Framework Components: Merging Module

```
Require: Uri RemoteSimulator //Simulator to merge the load with
1: while (Region other than transit exists) do
2:     Warn the players about maintenance to begin;
3:     Lock the region;
4:     Save content to an oar file at shared location;
5:     Get region specification;
6:     if (Region has players) then
7:         Transfer players to transit region;
8:     end if
9:     Remove region;
10:    Create region on RemoteSimulator; //provide specification
11:    Load content from oar file to the region on RemoteSimulator;
12:    if (transit region has players) then
13:        Transfer the players back to the region on RemoteSimulator;
14:    end if
15: end while
```

region are warned about the maintenance work and the scene objects are locked before transferring it. DistributeLoad module takes a list of regions to transfer while MergeLoad requires a Sim to merge the load with. The merge operation selects a Sim using the merging strategies implemented in our work, which could be a parent or another child Sim. When a merge is permitted, a Sim returns all the regions and releases itself. In case there are players in a region while it is being transferred, they are transferred to the local transit region. Since the transit region is only used explicitly by the system, it returns the players in transit after relocating the region. The rest of the activities for both modules are self-explanatory and are not further explained. Although a region transfer completes when the players are transferred back to the actual region, the region is available to other users when it is relocated and the content are loaded to the region.

We have implemented this framework as a plug-in to OS framework and tested it over both Windows and Linux environments. Figure 7.4 shows a status report of parent Sim running a VW of four regions. It provides both regional and Sim statistics. Its current load is normal and it is accepting client connections. The local transit region is not counted in against the load or other processes. It can be noted that SimFPS is above 40FPS for about thirty-one players and a reasonable amount of both static and dynamic content. Figure 7.5 presents a later stage in the system after a split where the VW is now managed by two Sims (Sim-I and Sim-II). Both the Sims are managing normal load and are accepting client



```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Scalable Virtual Worlds
An OpenSim extended project based on our dynamic split and merge strategies
Umar Farroq and John Glauert
School of Computing Sciences, UEA Norwich UK.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

20:39:21 - Current Load Check!
Number of Regions: 4
Sim Status: Parent
Sim ID: http://10.141.0.102:9000/
--------------------

Regional Statistics
-----
Region Name: A
Sim FPS: 45
Physics FPS: 42.37977
root Agents: 10
child Agents: 20
Total Prims: 2649
Active Prims: 0
Active Scripts: 653

Region Name: B
Sim FPS: 43.33333
Physics FPS: 41.32303
root Agents: 11
child Agents: 20
Total Prims: 1757
Active Prims: 0
Active Scripts: 685

Region Name: D
Sim FPS: 43.33333
Physics FPS: 35.85833
root Agents: 10
child Agents: 20
Total Prims: 3181
Active Prims: 0
Active Scripts: 617

Region Name: E
Sim FPS: 43.33333
Physics FPS: 41.38626
root Agents: 0
child Agents: 30
Total Prims: 1728
Active Prims: 0
Active Scripts: 503
-------------------

Sim Statistics
-----
Total Prims: 9315
Total Players: 31

Normal Load, accepting connections! current load < SplitCapacity (40 Players)
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

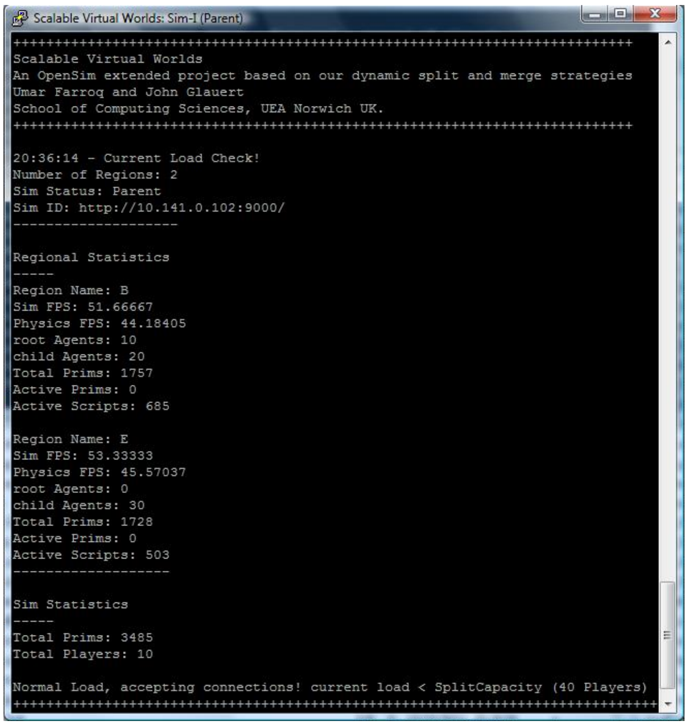
Figure 7.4: The status of a parent Sim (Sim-I) showing content and interactive players for a 4-region world.

connections. SimFPS is more than fifty for both the Sims showing that our system scales with improved performance. Based on further load, both will further share their load with additional Sims. In the case that their combined load goes below, or equals, twenty players, Sim-II will initiate the merge operation to release a resource.

7.3 Extended ARA Algorithm

The ARA algorithm was developed to aggregate regions based on a number of strategies and to obtain contiguous areas for assignment. It initially takes regional grids of n×n dimensions as an input normally based on split strategies of JoHNUM infrastructure. It repeatedly assigns different parts of the pre-processed space to additional Sims and it has to cope with varied shapes of spaces. In theory, the basic ARA algorithm should always yield valid combinations but in fact 'practically' it allowed odd combinations for non-square shaped grids. During implementation, it failed to discard odd cases. In other words, starting with a square grid, the first iteration determines valid contiguous spaces but in later iterations, when applied to non-square shaped worlds, it allows odd cases. To illustrate these situations, we can consider a simple square grid of four regions (A, B, D and E), as shown in Figure 7.6(a). If the first iteration divides this grid into two aggregates having A in first and B, D, and E in the second aggregate (see Figure 7.6(a)), the basic ARA algorithm, when applied to the second aggregate, can potentially select BD, which is a diagonal and thus an odd case, as shown in Figure 7.6(b). To achieve a more flexible approach, we extended the current ARA algorithm to eliminate its limitations. In each iteration the extended algorithm checks, if a prospective aggregated space is producing a valid contiguous space or not by using a flood-fill algorithm, as explained below.

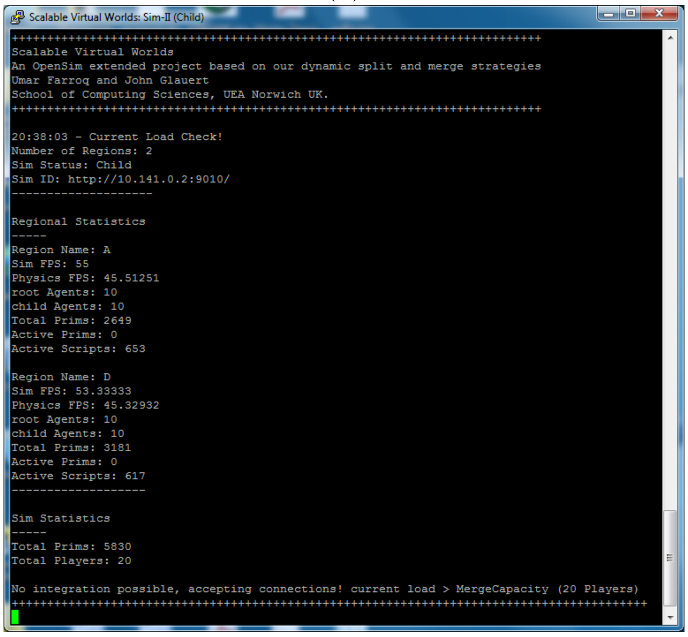
A flood fill algorithm determines an area connected to a given node in a multi-dimensional array beginning with a given node. These algorithms are normally used in bucket fill algorithms of paint programmes, and are employed in board games such as Go and Minesweeper [72]. It has two variations based on the direction of spreading, which are 4-way spreading, and 8-way spreading. In a 4-way



Scalable Virtual Worlds: Sim-I (Parent)

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Scalable Virtual Worlds
An OpenSim extended project based on our dynamic split and merge strategies
Umar Farroq and John Glauert
School of Computing Sciences, UEA Norwich UK.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

20:36:14 - Current Load Check!
Number of Regions: 2
Sim Status: Parent
Sim ID: http://10.141.0.102:9000/
--------------------

Regional Statistics
-----
Region Name: B
Sim FPS: 51.66667
Physics FPS: 44.18405
root Agents: 10
child Agents: 20
Total Prims: 1757
Active Prims: 0
Active Scripts: 685

Region Name: E
Sim FPS: 53.33333
Physics FPS: 45.57037
root Agents: 0
child Agents: 30
Total Prims: 1728
Active Prims: 0
Active Scripts: 503
-------------------

Sim Statistics
-----
Total Prims: 3485
Total Players: 10

Normal Load, accepting connections! current load < SplitCapacity (40 Players)
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

(a)



Scalable Virtual Worlds: Sim-II (Child)

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Scalable Virtual Worlds
An OpenSim extended project based on our dynamic split and merge strategies
Umar Farroq and John Glauert
School of Computing Sciences, UEA Norwich UK.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

20:38:03 - Current Load Check!
Number of Regions: 2
Sim Status: Child
Sim ID: http://10.141.0.2:9010/
--------------------

Regional Statistics
-----
Region Name: A
Sim FPS: 55
Physics FPS: 45.51251
root Agents: 10
child Agents: 10
Total Prims: 2649
Active Prims: 0
Active Scripts: 653

Region Name: D
Sim FPS: 53.33333
Physics FPS: 45.32932
root Agents: 10
child Agents: 10
Total Prims: 3181
Active Prims: 0
Active Scripts: 617
-------------------

Sim Statistics
-----
Total Prims: 5830
Total Players: 20

No integration possible, accepting connections! current load > MergeCapacity (20 Players)
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

(b)

Figure 7.5: The 4-region world presented in Figure 7.4 after a split jointly served by (a) Sim-I (Parent), and (b) Sim-II (Child).

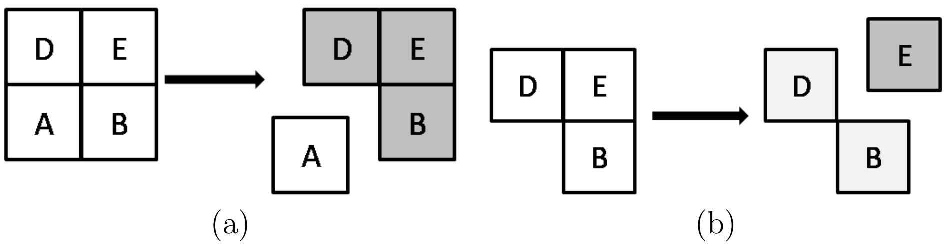


Figure 7.6: Illustrating the limitations in basic ARA algorithm (a) a valid outcome for a square grid, and (b) an invalid outcome for a 3-region world.

spreading, the algorithm looks at east, west, north, and south of a node. However, in 8-way spreading it also considers nodes in north-east, north-west, south-east, and south-west. It can be observed that using an 8-way algorithm considers the diagonal nodes to a node. However, we have identified that diagonals are one of the main reasons for odd cases and, hence, do not use the 8-way variation. The 4-way variation has the potential to identify odd cases and is used with ARA Algorithm to exclude them. We have implemented a variation of flood-fill algorithm with the help of a 2D array and an explicit queue (adapted from [71, 73]). In each step of the aggregation process, we use it to determine whether all the regions in both prospective aggregates are giving valid contiguous spaces or not. It is used during both split and merge operations and it has the capability to determine and exclude odd cases against any size and shape of grid.

7.4 Flexibility of System and Envisioning Scalability

In this section, we examine the flexibility of our system by considering different numbers of resources and VWs of different sizes against user distribution. The basic aim is to identify those factors that restrict VWs to scale. We introduce a new parameter called GridCapacity that represents the total number of potential concurrent users handled by our system. GridCapacity is determined using SimCapacity against basic unit regions in a VW. In theory, our system deals with a bigger world, a uniform distribution of players and the availability of required

Sims scales well and gets a GridCapacity = SimCapacity × Number of unit regions. However, in practice, it is limited by three factors, which are: number of Sims, number of unit regions in a world, and player distribution. Player distribution is the most critical factor. If all players are in a single region of a bigger world, the availability of Sims cannot help to scale the world. Similarly, if no Sim is available to share the load, we are unable to increase capacity. Furthermore, in a situation where each unit region is served by a Sim, additional Sims are unable to help a system to handle more users. Our approach is dynamic and performs better in most of the situations, thus eliminating a number of issues in both static and dynamic systems. It has the potential to scale up to its full capacity. Table 7.3 shows player distribution and the corresponding GridCapcity and number of resources required in different cases for a world of four regions.

Experiment Number	Regional Grid	Players Distribution	GridCapacity (Maximum)	Required Resources
1	D E A B	Populating one region only	60	1
2	D E A B	Populating two regions only	120	2
3	D E A B	Populating three regions	180	3
4	D E A B	Populating four regions	240	4

Table 7.3: Description of experiments and players distribution for a 4-region grid with GridCapacity and number of required resources.

7.5 Merging Strategies

In this section, we present two strategies for merging the load of different Sims for our current implementation. It considers a single parent Sim and a number of child Sims that are directly attached to the parent Sim. However, if a hierarchy goes deeper, as proposed in our theoretical model, then some nodes are both child and parent where a node can only merge if it has no children. Both the strategies for our current environment are initiated by a child Sim and are named Parent Merge (PM) and Child Merge (CM) strategies. They need to be validated against two constraints. The former needs to determine that the combined load of both Sims is less than, or equal to, MergeCapacity. The latter requires that the regions of both Sims form a valid larger space.

In the PM strategy, each child Sim returns its regions to the parent Sim when the integration is validated against the constraints. It is the simplest and easiest way to implement merging; however, regions having no players or fewer than MergeCapacity might be waiting for parent capacity to decrease. Similarly, to validate against the second constraint, it might be waiting for other Sims to integrate first with the parent.
The **CM strategy** integrates its load with a potential sibling Sim if it cannot merge with the parent Sim. It eliminates the issues in PM strategy by reducing the number of resources much quicker than PM strategy. However, it might transfer some regions multiple times between child Sims. This might bring a bad experience to the players in those regions. Similarly, it incurs computation and communication burden on the system. However, CM strategy has potential to become a vital force for implementing load balancing.

In this work, we have implemented and compared both strategies for their trade-offs. The ultimate outcome of both strategies see all the regions integrated back to the parent Sim. However, it depends on player distribution, which can never be predicted.

7.6 Final Experiments

We have performed a large set of experiments on both Windows and Linux environments to evaluate our proposed system. We tested the flexibility of our system by applying it to worlds of both four and nine regions. In this section, we present a summary of experiments for a world of four regions and demonstrate both the expansion and contraction of our system with the help of a nine region world. The basic aim is to show the flexibility of our system and to discuss a number of concepts which are compared for their trade-offs. We use the numeric values identified in section 6.3.4 (of chapter 6) for SplitCapacity, MergeCapacity, and SplitCapacity for these experiments. Regions are populated with their corresponding content, as described in section 7.1.4. The statistics described in section 7.1.5 are used to evaluate and compare our system with other systems. Removing a region is using our Improved-II strategy, as described in section 6.7.2 (of chapter 6). Regions are populated with Bots which are asked to read ScriptT, as described in section 7.1.4.

Equations 7.1- 7.4 represent Region Content Transfer Time (RCT Time), Region Un-availability Time (RUnAv Time), Region Transfer Time (RT Time), and Total Time consecutively. They are based on time parameters whose values are calculated during experiments. Player transfer dominates these activities but transferring regions in turn reduces both region un-availability time and the number of players that suffer from a split.
Equation 7.1 is used to identify the time taken by transferring regional content, and its basic purpose is to show that player transfer is a time consuming activity. It is the same as RUnAv Time and RT Time if a region has no players.

$$\textbf{RCT Time} = \text{ SC Time } + \text{ RR Time} + \text{ CR Time} + \text{ LC Time} \tag{7.1}$$

Equation 7.2 calculates the un-availability time of a regional content. It includes the time taken by activities, from blocking a region to setting it up on the destination Sim. Its outcome is based on the number of players in a region. If a region has no players, its value is the same as RCT Time and RT Time.

$$\textbf{RUnAv Time} = \text{ RCT Time } + \text{ T2T Time} \tag{7.2}$$

Equation 7.3 gives the time taken by a region transfer, including time to teleport players back to the original region on new Sim. It takes longer but has no impact on other regions which are not yet transferred, and whose content is still available.

$$\textbf{RT Time} = \text{ RUnAv Time } + \text{ T2R Time} \tag{7.3}$$

Equation 7.4 is the total time taken by an aggregate transfer and its basic purpose is to compare our system with other systems which transfer the whole space at the same time. It is cumulative time taken by n regions in an aggregate.

$$\textbf{Total Time} = \sum_{r=1}^{n} \text{RT Time}_r \tag{7.4}$$

Based on the experiments in this work, we observed that most of the activities during a transfer take almost a constant amount of time, including player transfer time. However, LC Time is different for each example world and is potentially based on both the content and complexity of the scene. Similarly, the SC Time varies a little with the amount of content in a region. We can easily use these formulae to develop a prediction model for region as well as an aggregate transfer. To determine a relationship between load content time and content itself, we investigated a number of parameters such as prims, and assets/objects in a scene, as well as file size of an OAR file, which are presented in Table 6.1 (of chapter 6). The file size is more closely related to load content time, which can be used for predicting both SC Time and LC Time. However, this will be our future work.

Our current implementation of a prototype improves over the traditional methods due to transferring regions one at a time. It performs better, both in terms of content un-availability time and number of players suffer from a split.

7.6.1 Experiments with 4-Region World

We tested a world of four regions on both Windows and Linux Environments with a wide range of experiments. However, they explained little about all the concepts, and therefore we provide the detailed expansion and contraction based on our work for a 9-region world on the Linux environment. The Windows environment demonstrated that our system scales and shrinks based on our strategies; however, due to limited capabilities, we could not use them for bigger worlds such as those based on nine regions. When nine regions were populated, players started noticing lag during physics and network processing due to limited memory of the system. Therefore, the Windows environment was only used for a 4-region world. However, we could not explore it up to full capacity due to a limited number of resources. The Linux environment overcame not only the scarcity of resources and limited capabilities but it also improved the content load time, which greatly reduced the total time for transferring a region (this is detailed in chapter 6). It also provided a large number of nodes with more computation speed and memory. Therefore, the Linux environment was used to demonstrate scaling worlds of both four and nine regions up to a level where each Sim was running a unit region. These environments provided the trade-offs between the number of Sims and inter-sim crossing in response to the scaling process. It identified a number of issues such as odd cases giving better distribution with odd combinations excluded by the ARA algorithm and the trade-offs between merging strategies. It showed an improved performance compared with the existing methods for transferring load due to sending regions one at a time. These issues are demonstrated with the 9-region world in the next section.

7.6.2 Experiments with 9-Region World

In this section, we demonstrate our system for a 3×3 grid, and explain various issues and compare it with traditional systems using two different cases due to their similar outcomes and shortage of space. GridCapacity in each case is constrained by the number of regions being populated. A 9-region world can be scaled up to 540 players when players are distributed in all regions. However, using bigger worlds based on more unit regions can potentially scale further. The Linux nodes, we used have the potential to host worlds based on sixteen and twenty-five regions being tested, but not demonstrated. Uniform distribution of players is used to demonstrate the full potential of a 9-region world initially running over parent Sim (Sim-I) using nine Sims. Both Parent Merge (PM) and Child Merge (CM) strategies are then used to merge the load back to Sim-I when players' capacity is decreased. They are compared for tradeoffs based on the outcome of merging processes. We also present a case where three regions out of nine are populated to identify the behaviour of our system when a sufficient number of Sims are available to scale a world.

A). Scalability and Time Analysis

We add a new Sim as the load goes up. Table 7.4 shows the potential points during these experiments and the ultimate capacities they achieve against possible constraints. The SplitCapacity for this work is forty players. Experiment 1 is con-

strained by players distribution and it scales up to 180 players using three Sims. It highlights the fact that our current implementation transfers a large number of regions with no players; however, it has no impact on performance as it transfers them in turn. It transfers a total of fifteen players in two regions at step one and, therefore, reduces the number of players and total time they suffer from the split. A traditional technique using our approach, and transferring the aggregate at once, resumes the complete process in 425 seconds compared with the most time consuming region H, which take 169 seconds. Similarly, fifteen players have to wait for about 425 seconds to resume, compared with a maximum of ten players waiting for 169 seconds. A region in our approach is un-available for a maximum of 88 seconds, compared with 306 seconds (the sum of un-availability time of regions in the aggregate) of un-availability for the aggregated content via traditional methods. Similarly, step two also shows that our method get improvement over existing mechanisms. Our approach takes very little time to transfer a region and make it available when it has no players in it. However, in a traditional systems it is unavailable until the complete aggregated content is transferred. Our approach improves the overall performance by sending regions in turn, thus alleviating the effect of transferring other regions or players in a region on a region transfer. We believe that players in our current implementation have a better experience.
In experiment 2, players are uniformly distributed among nine regions and it scales up to full potential reaching a player capacity of 540 players. Each region has a small number of players, and Sim-I transfers three regions at step one and two to Sim-II and Sim-III, respectively. In step one, a total of nineteen players are transferred, which are distributed among regions A, D and G with A having ten players. However, only ten players are restricted for the duration of region A being transferred taking about 215 seconds, compared with nineteen suffering for about 413 seconds. However, the content of region A is only un-available for about 131 seconds compared with the un-availability of the aggregated content of A, D and G for about 256 seconds. The region unavailability time and region transfer time for regions D and G are considerably improved. In step two, Sim-I transfers regions C, F and I with thirteen , four and three players correspondingly to Sim-III. Region C takes much longer among these regions due to transferring a large portion of total players, but this still improves the overall performance. Regions F and I are

Table 7.4: Illustrating important steps during scaling a world of 9 regions.

Exp./ Step	Regional Players	From Sim	To Sim	Reg- ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
1/1	G:00 H:10 I:25	Sim-I	Sim-II	8	A	7Sec	-	6Sec	2Sec	38Sec	-	53Sec	53Sec	53Sec	425Sec
	D:00 E:00 F:05				B	6Sec	-	7Sec	1Sec	11Sec	-	25Sec	25Sec	25Sec	
	A:00 B:00 C:00				C	3Sec	-	5Sec	2Sec	5Sec	-	15Sec	15Sec	15Sec	
					D	8Sec	-	7Sec	2Sec	32Sec	-	49Sec	49Sec	49Sec	
					E	4Sec	-	5Sec	1Sec	5Sec	-	15Sec	15Sec	15Sec	
					F	2Sec	41Sec	3Sec	2Sec	3Sec	38Sec	10Sec	51Sec	89Sec	
					G	3Sec	-	4Sec	1Sec	2Sec	-	10Sec	10Sec	10Sec	
					H	2Sec	78Sec	3Sec	2Sec	3Sec	81Sec	10Sec	88Sec	169Sec	
1/2	G:00 H:15 I:40	Sim-II	Sim-III	2	G	2Sec	-	3Sec	2Sec	3Sec	-	10Sec	10Sec	10Sec	259Sec
	D:00 E:00 F:25				H	2Sec	118Sec	4Sec	1Sec	3Sec	121Sec	10Sec	128Sec	249Sec	
	A:00 B:00 C:00				-	-	-	-	-	-	-	-		-	
1/3	G:00 H:60 I:60	-	-	-	-	-	-	-	-	-	-			-	
	D:00 E:00 F:60				-										
	A:00 B:00 C:00				-	-	-	-	-	-	-			-	
2/1	G:04 H:05 I:03	Sim-I	Sim-II	3	A	7Sec	82Sec	5Sec	1Sec	36Sec	84Sec	49Sec	131Sec	215Sec	413Sec
	D:05 E:02 F:04				D	8Sec	40Sec	6Sec	2Sec	30Sec	42Sec	46Sec	86Sec	128Sec	
	A:10 B:04 C:03				G	2Sec	30Sec	2Sec	2Sec	3Sec	31Sec	9Sec	39Sec	70Sec	
2/2	G:09 H:15 I:03	Sim-I	Sim-III	3	C	3Sec	106Sec	4Sec	2Sec	6Sec	102Sec	15Sec	121Sec	223Sec	352Sec
	D:10 E:02 F:04				F	2Sec	33Sec	2Sec	1Sec	3Sec	32Sec	8Sec	41Sec	73Sec	
	A:15 B:04 C:13				I	3Sec	25Sec	2Sec	2Sec	2Sec	22Sec	9Sec	34Sec	56Sec	
2/3	G:15 H:15 I:13	Sim-II	Sim-IV	1	A	7Sec	118Sec	7Sec	3Sec	38Sec	122Sec	55Sec	173Sec	295Sec	295Sec
	D:10 E:12 F:04				-										
	A:15 B:04 C:13				-										
2/4	G:25 H:15 I:13	Sim-II	Sim-V	1	D	8Sec	116Sec	6Sec	2Sec	28Sec	121Sec	44Sec	160Sec	281Sec	281Sec
	D:15 E:12 F:04				-										
	A:25 B:04 C:13				-										
2/5	G:40 H:25 I:13	Sim-I	Sim-VI	2	B	7Sec	35Sec	5Sec	1Sec	13Sec	33Sec	26Sec	61Sec	94Sec	299Sec
	D:25 E:12 F:04				E	3Sec	94Sec	4Sec	2Sec	4Sec	98Sec	13Sec	107Sec	205Sec	

Continued on next page

Table 7.4 – continued from previous page

Exp./ Step	Regional Players	From Sim	To Sim	Reg-ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
	A:40 B:04 C:13				-										
2/6	G:40 H:25 I:13 D:25 E:22 F:04 A:40 B:19 C:13	Sim-VI	Sim-VII	1	B - -	5Sec	148Sec	6Sec	1Sec	12Sec	153Sec	24Sec	172Sec	325Sec	325Sec
2/7	G:40 H:25 I:23 D:25 E:22 F:04 A:40 B:19 C:13	Sim-III	Sim-VIII	2	C F -	4Sec 2Sec	102Sec 33Sec	6Sec 4Sec	2Sec 2Sec	7Sec 4Sec	108Sec 31Sec	19Sec 12Sec	121Sec 45Sec	229Sec 76Sec	305Sec
2/8	G:40 H:25 I:23 D:25 E:22 F:24 A:40 B:19 C:18	Sim-VIII	Sim-IX	1	C - -	3Sec	141Sec	5Sec	1Sec	5Sec	148Sec	14Sec	155Sec	303Sec	303Sec
2/9	G:60 H:60 I:60 D:60 E:60 F:60 A:60 B:60 C:60				- - -	-	-	-	-	-	-				

un-available for only 41 and 34 seconds in this case and this gives a much better performance. Sim-II at step three and four delegates regions A and D with fifteen players each to Sim-IV and Sim-V consecutively. In these cases, both mechanisms take a similar amount of time to complete a transfer, and no improvement is achieved. However, it minimises the region unavailability time. Assignment at lower stages, when the Sims are handling a small number of regions, has similar outcomes; however, we believe that these cases are normally rare. Transfers in these cases are dominated by player transfer time, though the content transfer time is quite reasonable. Step five and seven show an improved performance. However, step six and eight are unable to achieve improved performance and take a considerable amount of time by transferring nineteen and eighteen players (almost the worst possible case). The split at step eight completes the assignment process and no further splits are possible due to each Sim hosting a unit region. However, each Sim accepts connections until it reaches its SplitCapacity, which is sixty players in this work.

B). System Statistics and Analysis

In this section, we present a summary of experiments presented in Table 7.4 in terms of number of used resources, inter-sim crossings, and total number of transfers during the scaling process. These parameters are presented in Table 7.5, which also shows the maximum capacities obtained against the constraints in each case. Figure 7.7 provides trends for resource utilisation and inter-sim crossings against capacity. It can be noted that we can possibly scale a world under constraints up to certain limits. The most vital constraint is player distribution, which limits the scaling process even if a world is based on a large number of regions and additional Sims are available to share the load (see experiment 1 in Table 7.4). However, we utilise a limited number of resources compared with static infrastructures. Since it handles a limited number of players using a few resources, the inter-sim crossings are minimum. When more resources are used to cope with an increasing number of players, it increases inter-sim crossings greatly, as shown in Figure 7.7(b). The inter-sim crossings are further increased if odd cases are allowed.

Experiment No.	Step	Current Capacity	Inter-Sim Crossings	Player Transfers (Cumulative)	Number of Sims
1	1	0	0	0	1
	2	40	44	15	2
	3	80	112	30	3
	4	180	238	30	3
2	1	0	0	0	1
	2	40	46	19	2
	3	75	100	39	3
	4	101	146	54	4
	5	126	182	69	5
	6	176	296	85	6
	7	201	346	104	7
	8	211	372	121	8
	9	236	420	139	9
	11	540	960	139	9

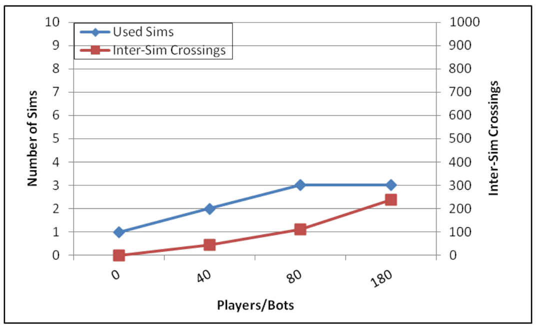
Table 7.5: Number of resources, inter-sim crossings and player transfers (cumulative) against current capacity while scaling a 9-region world based on population of regions (based on experiments from Table 7.4).

We use three other metrics to show the performance of our system: Sim utilisation, Player disruption, and Transfers per player during the scalability process. We use the statistics of experiment 2, presented in Table 7.4, for these illustrations.

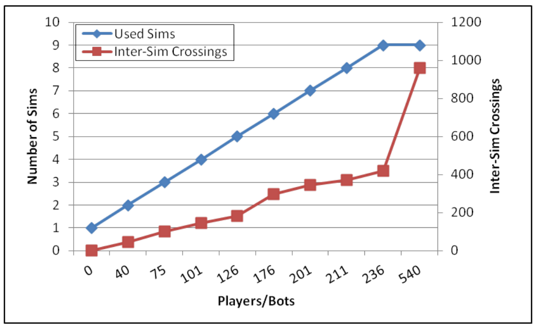
Sim Utilisation demonstrates that our system never allows more players than its SimCapacity and, therefore, it always maintains the minimum frame rate for SimFPS, which is thirty as described previously. Figure 7.8(a) shows that on average each Sim serves a lesser number of players than SplitCapacity except when each unit region was assigned to a single Sim (see the last observation regarding having 560 players).

Player disruption presents a slight variation as we used a very simple mobility model where each Bot makes only two moves during the whole duration. We believe that player disruptions (connections/disconnections) will greatly increase if more dynamic mobility models are simulated for longer. However, it still shows that average player disruptions increased when we added more Sims, as shown in Figure 7.8(b).

Transfers per player shows that our system transfers a very small number of players on average compared with other load balancing techniques. It is clear that player transfers only happen until each unit region is assigned to a single Sim. Figure 7.9 shows that, during the split processes based on current scenarios, almost every second player is transferred. However, after getting to a point where each region is assigned to a single Sim, no further splits are permitted. This illustrates that our system is efficient and transfers a very small number of the total players. For our system to be completely populated, on average the fourth player is transferred throughout the given cases. It can been seen in Table 7.5 that the cumulative player transfers are usually less than the maximum possible limit. It demonstrates that our ARA algorithm reduces the number of players which suffer from transfers.

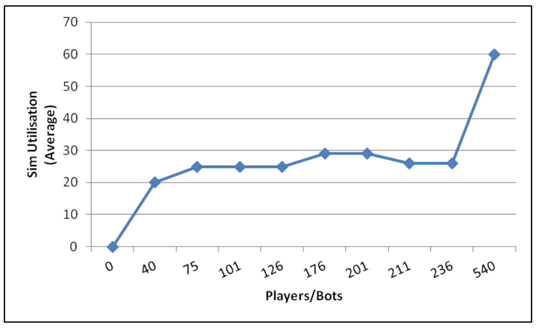


(a)

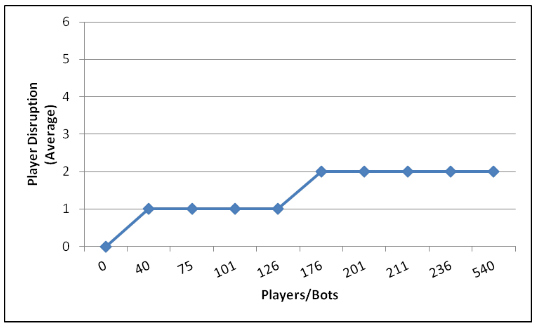


(b)

Figure 7.7: Number of resources and inter-sim crossings with an increase in players capacity for a world of 9 regions with players (a) populating three region. (b) populating nine regions.



(a)



(b)

Figure 7.8: Illustrating (a) Sim utilisation. (b) Player disruption (disconnections/connections).

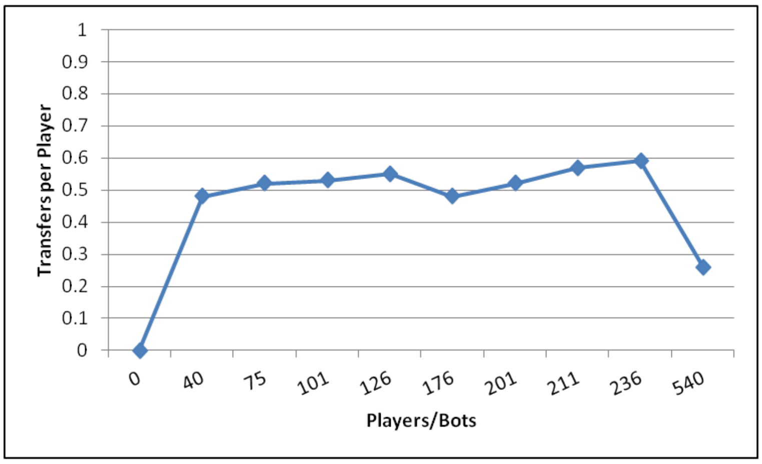


Figure 7.9: Illustrating transfers per player.

C). Merging and Time Analysis

In this section, we present a summary of the merging process for both PM and CM strategies applied to the final stage of experiment 2, obtained by the scaling process that is presented in Table 7.4. The PM strategy is demonstrated using Table 7.6, and CM strategy is presented in Table 7.7. Table 7.8 presents a summary of both merging strategies in terms of number of resources and transfer of regions that are used to compare them.

Table 7.6 presents important steps during merging using PM strategy. Parent Sim (Sim-I) is hosting region H. Table 7.8 gives a clear picture of the regions served by each Sim during the merging process. No merge operation is possible due to the first constraint during the first two steps. In step three, Sim-VI and Sim-I satisfy the first condition but fail the second one. Similarly, the cumulative load of Sim-IV, Sim-VII, and Sim-IX with Sim-I is less than, or equal to, MergeCapacity but their combined spaces are not contiguous and, therefore, no merge is permitted. However, it can be noted that there are a number of points where merge could be allowed by the CM strategy. In the PM strategy, each Sim is waiting for other Sims to merge first, which would help them to integrate. In step five, Sim-II integrates its load with Sim-I after satisfying both constraints. Each child Sim in the following steps, except step six, integrates its load with Sim-I after validating

Table 7.6: Illustrating important steps during merging using Parent Merge (PM) strategy for a world of 9 regions (continued from Table 7.4).

Exp./ Step	Regional Players	From Sim	To Sim	Reg- ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
2/1	G:60 H:60 I:60 D:60 E:60 F:60 A:60 B:60 C:60			-	-	-	-	-	-	-	-		-	-	
2/2	G:20 H:18 I:20 D:20 E:15 F:20 A:10 B:20 C:16			-	-	-	-	-	-	-	-		-	-	
2/3	G:20 H:10 I:20 D:20 E:15 F:20 A:05 B:15 C:16			-	-	-	-	-	-	-	-	-		-	
2/4	G:20 H:10 I:20 D:20 E:15 F:15 A:05 B:05 C:10			-	-	-	-	-	-	-	-	-		-	
2/5	G:10 H:10 I:20 D:20 E:15 F:15 A:05 B:05 C:10	Sim-II	Sim-I	1	G	3Sec	82Sec	4Sec	2Sec	4Sec	78Sec	13Sec	95Sec	173Sec	173Sec
2/6	G:10 H:10 I:20 D:20 E:15 F:05 A:05 B:05 C:05			-	-	-	-	-	-	-	-	-		-	
2/7	G:05 H:05 I:20 D:10 E:15 F:05 A:05 B:05 C:05	Sim-V	Sim-I	1	D - -	8Sec	79Sec	5Sec	1Sec	29Sec	81Sec	43Sec	122Sec	203Sec	203Sec
2/8	G:05 H:00 I:15 D:05 E:15 F:05 A:05 B:00 C:00	Sim-IV	Sim-I	1	A -	7Sec	40Sec	7Sec	2Sec	36ec	42Sec	52Sec	92Sec	134Sec	134Sec
2/9	G:05 H:00 I:10 D:05 E:10 F:05 A:05 B:00 C:00	Sim-VII	Sim-I	1	B - -	5Sec	-	6Sec	1Sec	11Sec	-	23Sec	23Sec	23Sec	23Sec

Continued on next page

Table 7.6 – continued from previous page

Exp./ Step	Regional Players	From Sim	To Sim	Reg-ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
2/10	G:05 H:00 I:10 D:05 E:10 F:05 A:05 B:00 C:00	Sim-IX	Sim-I	1	C - -	3Sec	-	5Sec	2Sec	6Sec	-	16Sec	16Sec	16Sec	16Sec
2/11	G:05 H:00 I:10 D:00 E:10 F:05 A:00 B:00 C:00	Sim-VIII	Sim-I	1	F - -	3Sec	39Sec	3Sec	2Sec	4Sec	41Sec	12Sec	51Sec	92Sec	92Sec
2/12	G:05 H:00 I:10 D:00 E:10 F:05 A:00 B:00 C:00	Sim-VI	Sim-I	1	E - -	3Sec	76Sec	4Sec	1Sec	5Sec	82Sec	13Sec	89Sec	171Sec	171Sec
2/13	G:05 H:00 I:05 D:00 E:05 F:05 A:00 B:00 C:00	Sim-III	Sim-I	1	I - -	2Sec	42Sec	3Sec	2Sec	3Sec	43Sec	10Sec	52Sec	95Sec	95Sec

the constraints and releases itself. Since each Sim returns its region straight to the parent, there are no additional transfers involved. It is important to note at step eleven that the merging process allowed a contiguous space that is never used by the ARA algorithm for assignment; however, it introduces no issues identified by odd cases. Since the merging process is normally initiated when player capacity decreases, it can be noted that each transfer involves less players than in the scaling process. Merging, therefore, reduces the region un-availability time. Since we observed a number of places where the CM strategy can potentially merge with its sibling Sim, the system might release under-utilised Sims quicker than the PM strategy. Therefore, we apply the CM strategy to the same distribution and compare it with the PM strategy.

Table 7.7 presents the significant steps during the merging process using the CM strategy. Table 7.8 manages regions to Sim assignment during the process for clarity. No integration is possible at step one and two due to the combined capacity constraint. In step three, Sim-IV integrates with Sim-VII after validation against the constraints. It can be seen that both Sim-IV and Sim-VI maintain the capacity constraint, but fail the continuity test. In step four, Sim-VII transfers two regions to Sim-IX including region A transferred to Sim-VII in the previous step. It transferred ten players, each region having five players. Sim-II transferred its region with ten players to Sim-I at step five. Sim-VIII relocated a single region to Sim-IX with five players. Our current implementation allows each Sim to merge with a sibling Sim by taking local decisions. If Sim-IX had initiated the process to move its load to Sim-VIII, then we would have more additional transfers by transferring a space comprises of three regions. Sim-V integrated its load with Sim-I at step seven with ten players in a single region. In step eight, Sim-IX moved four regions to Sim-I with ten players equally distributed in two regions. No integrations were possible due to first constraint at steps nine and ten. Sim-VI in step eleven and Sim-III in step thirteen return their regions to Sim-I, which concludes the merging process. The whole world is now again hosted by the parent Sim. It was observed that, as capacity decreased and Sims could perform a merge satisfying the constraints, they did so, thus releasing a resource. However, multiple transfers were observed for different regions such as A and B, that were transferred

Table 7.7: Illustrating important steps during merging using Child Merge (CM) strategy for a world of 9 regions (continued from Table 7.4).

Exp./ Step	Regional Players	From Sim	To Sim	Reg- ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
2/1	G:60 H:60 I:60 D:60 E:60 F:60 A:60 B:60 C:60			-											
2/2	G:20 H:18 I:20 D:20 E:15 F:20 A:10 B:20 C:16			-											
2/3	G:20 H:10 I:20 D:20 E:15 F:20 A:05 B:15 C:16	Sim-IV	Sim-VII	1	A - -	8Sec	41Sec	7Sec	2Sec	38Sec	43Sec	55Sec	96Sec	139Sec	139Sec
2/4	G:20 H:10 I:20 D:20 E:15 F:15 A:05 B:05 C:10	Sim-VII	Sim-IX	2	A B -	7Sec 6Sec	38Sec 41Sec	8Sec 6Sec	1Sec 2Sec	36Sec 13Sec	42Sec 40Sec	52Sec 27Sec	90Sec 68Sec	132Sec 108Sec	240Sec
2/5	G:10 H:10 I:20 D:20 E:15 F:15 A:05 B:05 C:10	Sim-II	Sim-I	1	G - -	2Sec	82Sec	4Sec	2Sec	3Sec	78Sec	11Sec	93Sec	171Sec	171Sec
2/6	G:10 H:10 I:20 D:20 E:15 F:05 A:05 B:05 C:05	Sim-VIII	Sim-IX	1	F - -	2Sec	39Sec	3Sec	1Sec	3Sec	42Sec	9Sec	48Sec	90Sec	90Sec
2/7	G:05 H:05 I:20 D:10 E:15 F:05 A:05 B:05 C:05	Sim-V	Sim-I	1	D - -	8Sec	82Sec	7Sec	2Sec	32Sec	84Sec	49Sec	131Sec	215Sec	215Sec
2/8	G:05 H:00 I:15 D:05 E:15 F:05 A:05 B:00 C:00	Sim-IX	Sim-I	4	A B C F	8Sec 4Sec 5Sec 2Sec	38Sec - - 42Sec	6Sec 5Sec 5Sec 3Sec	1Sec 2Sec 2Sec 1Sec	38Sec 11Sec 6Sec 3Sec	41Sec - - 40Sec	53Sec 22Sec 18Sec 9Sec	91Sec 22Sec 18Sec 51Sec	132Sec 22Sec 18Sec 91Sec	263Sec
2/9	G:05 H:00 I:10 D:05 E:10 F:05	-	-	-	- -	-	-	-	-	-	-	-		-	

Continued on next page

Table 7.7 – continued from previous page

Exp./ Step	Regional Players	From Sim	To Sim	Reg- ions	Reg. Name	SC Time (Sec)	T2T Time (Sec)	RR Time (Sec)	CR Time (Sec)	LC Time (Sec)	T2R Time (Sec)	RCT Time (Sec)	RUnAv Time (Sec)	RT Time (Sec)	Total Time (Sec)
	A:05 B:00 C:00				-										
2/10	G:05 H:00 I:10 D:05 E:10 F:05 A:05 B:00 C:00	-	-	-	- - -	-	-	-	-	-	-	-		-	
2/11	G:05 H:00 I:10 D:00 E:10 F:05 A:00 B:00 C:00	Sim-VI	Sim-I	1	E - -	3Sec	81Sec	4Sec	2Sec	4Sec	79Sec	13Sec	94Sec	173Sec	173Sec
2/12	G:05 H:00 I:10 D:00 E:10 F:05 A:00 B:00 C:00	-	-	-	- - -	-	-	-	-	-	-	-		-	-
2/13	G:05 H:00 I:05 D:00 E:05 F:05 A:00 B:00 C:00	Sim-III	Sim-I	1	I - -	2Sec	40Sec	3Sec	2Sec	2Sec	38Sec	9Sec	49Sec	87Sec	87Sec

three and two times respectively. We believe that it brings a bad experience to players although there are normally small numbers of them during the merging process.

Table 7.8 summarises the merging process for both PM and CM strategies. It provides the number of resources and cumulative region transfers against current capacity and provides a clear picture for Sims hosting different regions. Figure 7.10 shows the trends between resources and region transfers for both strategies. It suggests that CM strategy minimises the number of resources and achieves an improved utilisation of resources; however, some regions were transferred multiple times. PM strategy has no additional transfers, but more than the required number of resources were used for longer durations.

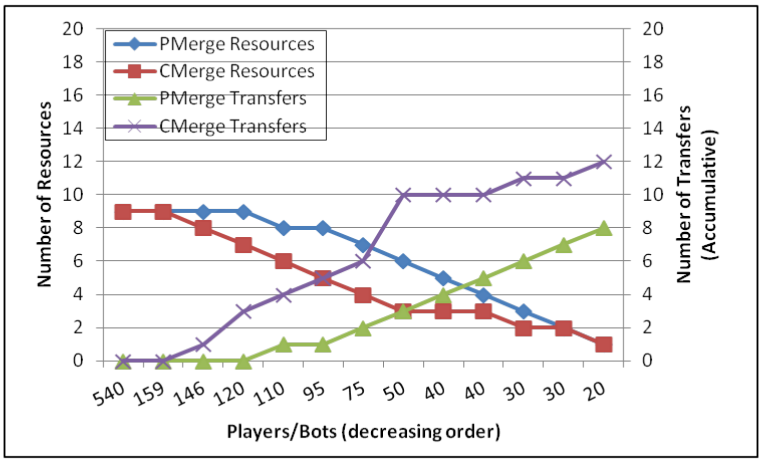


Figure 7.10: Comparison of Parent Merge (PM) and Child Merge (CM) strategies for both number of resources and region transfers.

7.6.3 Discussion

In this section, we demonstrated that our system is flexible that expands and contracts based on system capacity. It uses additional Sims during expansion, which are reduced later in response of merging. A region is only un-available when it is transferred, compared with traditional systems keeping it off limits until the whole space is transferred. Based on the fact that we never transfer more than

Exp./	Players	PM Strategy					CM Strategy				
Step		Sims Regions			Resources	No. of Transfers	Sims Regions			Resources	No. of Transfers
2/1	540	II	I	III	9	0	II	I	III	9	0
		V	VI	VIII			V	VI	VIII		
		IV	VII	IX			IV	VII	IX		
2/2	159	II	I	III	9	0	II	I	III	9	0
		V	VI	VIII			V	VI	VIII		
		IV	VII	IX			IV	VII	IX		
2/3	146	II	I	III	9	0	II	I	III	8	1
		V	VI	VIII			V	VI	VIII		
		IV	VII	IX			VII	VII	IX		
2/4	120	II	I	III	9	0	II	I	III	7	3
		V	VI	VIII			V	VI	VIII		
		IV	VII	IX			IX	IX	IX		
2/5	110	I	I	III	8	1	I	I	III	6	4
		V	VI	VIII			V	VI	VIII		
		IV	VII	IX			IX	IX	IX		
2/6	95	I	I	III	8	1	I	I	III	5	5
		V	VI	VIII			V	VI	IX		
		IV	VII	IX			IX	IX	IX		
2/7	75	I	I	III	7	2	I	I	III	4	6
		I	VI	VIII			I	VI	IX		
		IV	VII	IX			IX	IX	IX		
2/8	50	I	I	III	6	3	I	I	III	3	10
		I	VI	VIII			I	VI	I		
		I	VII	IX			I	I	I		
2/9	40	I	I	III	5	4	I	I	III	3	10
		I	VI	VIII			I	VI	I		
		I	I	IX			I	I	I		
2/10	40	I	I	III	4	5	I	I	III	3	10
		I	VI	VIII			I	VI	I		
		I	I	I			I	I	I		
2/11	30	I	I	III	3	6	I	I	III	2	11
		I	VI	I			I	I	I		
		I	I	I			I	I	I		
2/12	30	I	I	III	2	7	I	I	III	2	11
		I	I	I			I	I	I		
		I	I	I			I	I	I		
2/13	20	I	I	I	1	8	I	I	I	1	12
		I	I	I			I	I	I		
		I	I	I			I	I	I		

Table 7.8: Comparison of number of resources and number of regions transferred for both Parent Merge (PM) and Child Merge (CM) strategies for a world of 9 regions.

twenty players, the region un-availability time on the Linux environment never exceeds 220 seconds in our experiments with the OpenVCE world loaded in the region that has the most expensive content in terms of loading the content. On the other hand, current methods takes a very long time based on the number of regions and players in an aggregate. In the worst cases, our approach requires the same time as traditional systems, mostly when a world has fewer regions. For bigger worlds with players being distributed in multiple regions, we obtain a considerable amount of improvement both in terms of number of players and total time they suffer from a split. Players are never unavailable, thus giving them a better experience.

Two merging strategies (PM and CM) are implemented and compared in this work and each has value and limitations. Both of them ultimately return the whole world back to the parent Sim. The PM strategy takes more time and holds resources for longer than the CM strategy. However, the CM strategy potentially transfers regions between Sims multiple times and degrades performance. Normally, a merge operation is initiated when player capacity is not high. However, in the worst cases, it might need to transfer up to twenty players, same as in scaling process that happens rarely. The PM strategy is simple but Sims might be waiting for a parent Sim, handling nothing at certain stages. The CM strategy copes with the issues in PM strategy and release resources much quicker. However, it possibly transfers a large number of players and introduces additional transfer of content. Because of multiple transfers, it brings a bad experience to the users. We have demonstrated both the strategies, and both could be adopted according to requirements. Odd combinations are rejected by both the strategies. To manage bigger worlds and the un-predictable nature of users, we suggest using CM strategy as PM might be blocked for longer. However, both have the potential to cope with resource under-utilisation issues.

7.7 Bigger VWs

The Linux nodes used in this work have the potential to host bigger worlds over a single Sim based on a large number of regions such as sixteen, twenty-five or greater. However, it depends on the content populating world regions. We have tested it for worlds up to sixteen regions but we demonstrated it for square shaped worlds of four and nine regions. We believe that our system is flexible and is capable of managing worlds of any size and shape, but of a coherent space. We have demonstrated that, for uniform distribution, it scales up to potential capacity which is greatly increased for further larger systems. For sixteen and twenty-five region worlds, it could be scaled up to sixteen and twenty-five Sims with a maximum capacity of 960 and 1500 players consecutively. Since our current model uses a single parent Sim with the a large number of child Sims, it maintains parent-child relationship at a single level. We believe that it has the potential to reduce delays and complexities compared with hierarchical methods for the implementation of our consistency model in future. Based on our observations, we believe that bigger worlds greatly improve performance by normally transferring a small number of players in each region. For worlds based on a small number of regions, we noticed that our system gives little improvement over traditional systems compared with worlds with a large number of regions.

7.8 Comparison with Existing Systems

In this section, we provide a comparison of our infrastructure with the static and dynamic configurations used for well-known infrastructures, such as SL and Matrix.

7.8.1 Static Configurations

Our current implementation of JoHNUM infrastructure starts a Sim (called parent) with a large number of regions, compared with methods that statically assign a limited number of regions. It provides resources based on load, and solves both over-provision and under-provision issues of resources. The scaling process with additional Sims overcomes resource under-provisioning problems while the merging process resolves the over-provisioning problem.

Our method scales up to exactly the same capacity as static infrastructures. Since the population of VWs is unpredictable and constrained by player distribution, our method is significantly improved over static infrastructure by using fewer

resources.
Since our ARA algorithm potentially distributes a non-uniform load, it is similar to static infrastructures with some regions having fewer players than others. However, we apply the merging process to reduce extra resources in order to avoid resource under-utilisation, and we never dedicate resources to host regions without players.

7.8.2 Dynamic Configurations

Our current implementation of JoHNUM infrastructure assumes a world being pre-partitioned into a number of regions and transfers regions in an aggregate in turn which greatly minimises the number of players and the total time they suffer from a split. It also greatly reduces the content un-availability time.
Since we maintain a single parent Sim, additional resources are all managed in a single additional level. We believe it keeps our model simple and potentially reduces system complexity and communication overhead by minimising the number of levels in a hierarchy. For applications with conservative nature, a system with multiple levels introduces longer delays and, therefore, our model will help to implement our consistency model in the future.
Our approach is simple and takes purely local decisions with no central component, like Matrix Controller, that could become a system bottleneck. Each child Sim is totally independent and directly controlled by grid services, which allows it to keep serving the part of space it hosts on behalf of the parent Sim even if the parent is down for some reason. Furthermore, it has no additional management levels such as those needed for managing Matrix Servers in Matrix. Both scaling and merging processes are initiated directly by a Sim based on increase and decrease in its load.
Our infrastructure distributes and accepts non-uniform load to provide a better experience than complex systems which spend most of their time obtaining uniform load generally based on global strategies. Our current implementation uses merging strategies to overcome resource under-utilisation issues where a Sim can integrate its load and release itself. However, full load balancing is not yet implemented.

7.9 Conclusions and Future Work

In this chapter, we presented an abstract framework for scalable VW based on our informal model, whose development and investigation was detailed in chapter 6, using the OS architecture together with RAd and OAR functionalities. We used numeric values for SplitCapacity and MergeCapacity based on our load model. SplitCapacity takes an earlier decision to avoid a bad experience, and MergeCapacity uses a very small value to avoid frequent splits, thus improving performance.
We extended the basic ARA algorithm to make it flexible by adding a flood-fill algorithm that is used to validate an aggregate against odd cases at each level of assignment. This method of validation is also used during merge operations. Our current implementation of the ARA algorithm selects the best aggregate for transfer that has the least number of players. It normally improves performance, but, it might transfer a large number of regions with players only in a small number of regions.
A bigger OS world is constructed by placing a number of regions side by side, where the number of regions a Sim can handle depends on system capabilities. The Windows environment was used to determine that our implementation works. Due to its limited capabilities, we used it only for a world based on four regions. This demonstrated that Load Content (LC) operation is the most time consuming activity that greatly increases a region transfer time. Linux nodes greatly reduced the time taken by LC process and slightly improved player transfer time to reduce the overall time taken by transferring a region. It allowed us to test our work for worlds based on both four and nine regions.

We implemented our work as a Plug-in application over .NET framework using C# language. However, it also works well with Linux/Mono framework. It used grid mode to give a coherent view of a world hosted jointly by different Sims at later stages, but initially started with a parent Sim running the whole world. The modular structure of OS helped us to transfer regions in an aggregate independently, thus reducing the number of players that suffers from a transfer as well as total time taken by a transfer. Players are transferred to a local transit region during a transfer instead of freezing them. It allows players to teleport to other regions or keep themselves busy with simple activities until the maintenance is over. Transit regions provide very little content and have no real impact on system performance. Since they are managed locally, they put no burden on the grid.

Our framework assigns resources based on need, and achieves the same level of scalability as static configurations. Since we assume a world being pre-partitioned into the number of regions in a world, regions are all assigned to child Sims at a single additional level. Our current work implemented two merging strategies to cope with over-provision of resources. Both strategies maintain merging constraints and result in contiguous areas. They have their worth and limitations and could be used according to requirements.

For evaluation and comparison purposes, we used a number of time and system statistics. Time statistics such as content transfer time (RCT Time), region un-availability time (RUnAv Time), region transfer time (RT Time), and Total Time are based on time information of the different activities used during a transfer operation. It used a number of system statistics such as number of regions transferred, content transferred, number of players transferred, number of used resources, number of inter-sim crossings, Sim utilisation, player disruptions, and transfers per player.

The experimental results demonstrated that our proposed methods improve system performance in terms of Sim utilisation, and number of times a player is transferred.

The following points are identified as future work:

In future, we intend to explore much bigger worlds. Our current implementation sends players of a region into a transit region during a region transfer. However, it was seen to be a bottleneck after considerably reducing the RCT Time. In future, we will look into ways to reduce player transfer time. It could be interesting to compare our current method with traditional methods such as freeze and restore. In our current implementation, we have used an OS region as a basic unit region, and this has limitations in handling large numbers of players. It might be interesting to investigate the trade-offs between scalability and inter-sim crossings by reducing the size of an OS basic region. The OS architecture manages each region as a single entity and it involves complex intra-sim crossings. Our current implementation of the ARA algorithm maintains contiguous spaces that minimise inter-sim crossings, but the concept of megaregions can be extended to make it dynamic and this has the potential to overcome communication and intra-sim crossings issues.

Our current implementation uses OAR functionality to transfer a region with current state with the basic OS persistence step being disabled. The LC operation is the only operation that takes a considerable amount of time. More intelligent and improved methods could be developed to reduce LC Time. We need to identify how load content time is related to the content. Since there is a huge difference in timings for loading content and storing content operations, we might investigate and look into improving the load content algorithm. Direct database transfers might have the potential to achieve improved times but they might need to carefully consider the backup process as this is a time consuming activity.

Since our implementation improves performance when players are distributed among multiple regions, more intelligent strategies based on trade-offs between the number of regions and players for the ARA algorithm might be interesting to investigate and develop. We have used a greedy approach for the CM strategy, and a child Sim merges with another child Sim when both the constraints are validated, and might transfer a large number of players. It might be interesting to investigate shifting load from a Sim that has fewer players and regions to one that has more players and regions.

Our CM strategy could be further investigated as a load balancing method to reduce resource under-utilisation. Similarly, instead of using a new additional Sim for an overloaded Sim, load balancing can give better distribution of load. Further detailed analysis of trade offs between balancing the load and performance degradation due to multiple transfers of the same content might be interesting to study.

We also intend to develop a prediction model for predicting the timing information for the transfer function.

Chapter 8

Conclusions and Future Work

This chapter summarises the work undertaken during this study and discusses the value and limitations of our scalable and consistent infrastructure for VWs. It recommends a number of areas that could be investigated in future to further improve the performance of these systems.

8.1 Conclusions

This thesis examined a novel approach that combines two contemporary infrastructures to solve the issues in the mechanisms that are currently used to develop scalable and consistent VWs. The limitations of these approaches are presented in detail in chapter 2. This new approach uses a constrained hierarchical approach to manage system resources while targeting scalability and load distribution. It uses a P2P infrastructure with a constrained communication model based on inherent properties of VWs, while managing the temporal order among events. The consistency approach extends the basic capabilities of VWs to support conservative applications, thus making it a strong candidate for the future 3D web. This work used simulation studies to investigate the existing mechanisms for targeting scalability, load distribution, and consistency in VWs, and then developed a prototype to implement scalability and load distribution as an extension to OS.

The following goals were achieved during this study:

Scalability and Load Distribution

The JoHNUM infrastructure achieved scalable VWs with dynamic allocation of resources, and solved the issues of both over-provisioning and under-provisioning of resources. Simulation results showed that it performs better than game middleware Matrix in terms of levels in an RMT and interactive user experience. It further reduced resource utilisation and communication overhead in the same way as Matrix (see section 3.1 of chapter 3 for further detail).

Our ARA algorithm is capable of choosing contiguous and regular spaces for assignment based on aggregation strategies. It balances the load as much as possible, and speeds up the aggregation process using intelligent strategies. It has demonstrated that the communication and implementation cost and inter-sim crossings are reduced by excluding odd cases that might balance the load better than the proposed strategies (see section 3.2 of chapter 3). It is flexible and assigns resources strictly on requirements which greatly minimise them.

This work investigated the capabilities of the OS framework in detail and presented an extension to it to incorporate features for both scalability and consistency (see chapter 5 for further detail).

This study presented a generic load model based on our investigations that is capable of determining the points when a system needs to initiate a split or stop accepting more connections using different values of SimFPS. It can be used to determine approximated numeric values for different concepts such as splitting and merging. During this work, it was found that systems using simulation centric architecture do not scale with additional resources.

We studied different database options and suggest that a localised MySQL provides an excellent choice that has the potential to reduce communication overhead and avoid longer delays.

Our investigations revealed that removing a region from a Sim and loading content from an OAR file take a considerable amount of time. We presented two improved strategies that compared with the OS basic methods, significantly reduced the time taken by removing a region (see section 6.7 of chapter 6 for further detail).

We developed a prototype for implementing the scalability and load distribution strategies as a Plug-in to the OS framework and tested it on both Windows/.Net and Linux/Mono platforms. It used the concept of transferring players to a transit region during a transfer instead of freezing them to improve user interactive experience. Our implementation achieved improvements over traditional systems by transferring regions in an aggregate in turn, both in terms of content unavailability time and the number of players that suffer from a split. We tested our system for worlds of up to nine regions for both expansion and contraction, and demonstrated that it achieves the same level of scalability as a static configuration but uses fewer resources based on player distribution. Dividing a VW into more regions, as in our implementation, demonstrated that levels in an RMT are significantly improved.

We used a number of time and system statistics for evaluation and comparison purposes. We investigated the number of Sims being used and the number of inter-sim crossings being introduced due to the scaling process. It was shown that our approach performs better in terms of Sim utilisation, and reduces the average number of transfers per player.

We wrote additional methods to fix a number of bugs in the OS architecture.

To cope with resource under-utilisation, our current implementation presented two merging strategies. We have provided trade-offs between the two and they could be utilised according to requirements.

We also presented an extension to our basic ARA algorithm by incorporating a flood-fill algorithm to make it capable of choosing valid contiguous spaces when applied to any shape of world. Merging also utilises the flood-fill algorithm to maintain contiguous spaces (see chapter 7 for further detail).

Consistency Management

This thesis also presented a decentralised synchronisation approach. It utilises the inherent properties of VWs, and each federate directly interacts with its neighbouring federates. This is illustrated with the help of both simple flat and hierarchical scenarios. Simulation results showed that it achieves the correct temporal ordering for randomly generated events. Furthermore, an abstract model demonstrated that it has the potential to perform better than hierarchical approaches (see chapter 4 for further detail).

8.2 Benefits and Limitations

Our work has the following main strengths and limitations:
Both under-provisioning and over-provisioning issues in static assignment systems are fixed with our dynamic split and merge strategies. It achieves the same capacity as static infrastructures such as SL but with fewer resources. Our approach is simple, dynamic and it has proved that it greatly reduce the number of levels in an RMT compared with current dynamic methods.
The performance is significantly improved over the existing methods based on spatial partitioning with our improved strategies and using the OS capabilities for transferring regions in turn. It reduces the content un-availability time, but the content load operation still takes considerable amount of time. Transferring players into a transit region gives a better user experience but currently it takes a long time to transfer. Our system needs to maintain an additional transit region by each Sim, but this has no impact on system performance as it provides very little content.

The ARA algorithm distributes the load by selecting a lower number of players to transfer. However, it potentially transfers a large number of regions in certain situations. It uses a localised approach to provide better interactive experience and achieve better performance, but it distributes a non-uniform load. Furthermore, it reduces inter-sim crossings.
Merging strategies minimises resource under-utilisation, but full load balancing is not yet implemented.
Unlike similar studies, this work has developed a prototype to evaluate system capabilities by using real world content.
Our work is the only current project using the OS framework targeting the issues of over-provision and under-provision of resources in VW. It is compatible with the current release of the OS framework and we aim to introduce it as a component to the OS framework.

Our consistency mechanism is simple and decentralised in nature, which accommodate conservative applications in VWs. It maintains the consistent state of a space but it is not implemented. Our system might be blocked temporarily at different spots but these situations are potentially resolved quickly.

8.3 Future Directions

The following areas are identified for future research, based on the current study:
This work demonstrated that transferring players is an expensive operation. In our future work, we will look into different ways to reduce this time. It might be interesting to investigate and compare alternative methods, such as freeze and restore, with our approach.
Our current implementation used the standard size of an OS region that could potentially hold a large number of players. It could be further investigated if the basic region size can be reduced to a smaller one than 256m×256m. Trade-offs between scalability and inter-sim crossings would also be interesting to investigate.
A load model was developed based on testing the scene parameters SimFPS and PhysicsFPS, using static and dynamic content as well as interactive users. Further performance tests could be conducted to identify the response of other parameters that might help to extend the current load model for different requirements.
Our approach with aggregation strategies greatly reduced inter-sim crossings, but the concept of megaregions could be extended to reduce intra-sim crossings. It would be interesting to investigate improvements of megaregions over the standard representation of multiple regions.
Other intelligent strategies for the ARA algorithm need to be investigated to avoid situations that transfer a large number of regions, or a minimum number of players but in a very small number of regions. The trade-off between the number of regions and players might also be valuable to investigate.
The Load Balancing issue is of vital importance for improving resource underutilisation, which is not yet implemented. Our merging strategies could be extended to implement load balancing. Trade-offs between Sims and both communication overhead and degradation of interactive experience due to potential transfer of the same regions multiple times would be interesting to examine. More-

over, future work would include developing more intelligent merging strategies to overcome the limitations of existing strategies.

Our current implementation uses OAR functionality to transfer a region. It is determined that loading content is an expensive operation and we intend to develop improved algorithms to minimise their timing information in our future work. Other alternative methods could be interesting to investigate compared with our current methods.
In the future, we intend to develop a prediction model for predicting both region and aggregate transfer time.
The grid mode of OpenSim provides UGAIM (User, Grid, Asset, Inventory, and Messaging) services as a centralised application that is a possible bottleneck, and this could be extended to a distributed one for better performance.
We intend to incorporate our prototype as a component to the OS framework and work with the OS community to further develop load balancing strategies.
Communication as a vital part of VW systems needs to be investigated against the communication overhead introduced by the number of clients, as well as scene complexity, that usually have a negative effect on performance.

Our consistency management approach needs to be simulated for bigger worlds. In our future work, we intend to develop a prototype using the OS framework to investigate how it behaves in a real world example. It would be interesting to investigate the impact of federates on each other and find out how quickly this resolves the temporarily blocking states. Furthermore, it seems fascinating to compare it with hierarchical approaches for parameters such as delay, complexity and communication overhead.

Appendix A

An Introduction to Grid Computing

A.1 Background

The work in this thesis used Grid infrastructures and presented techniques to introduce dynamic abilities to scale virtual worlds. Most of the Grids used for the environments (such as SL Grid) in this work are of static nature. Our work introduced an additional level of resources to share an excessive load with a Sim hosting an arbitrary number of regions from an available pool of resources. Since in general an individual or an organisation might not be able to obtain enough resources, the concept of dynamic grids can help to overcome this limitation. Therefore, we include an introduction to the concept of grid computing, its base architecture, and the existing solutions in this thesis for a reference.

A.2 Introduction

Grid computing, in original concern, was devised to solve computation and data extensive problems. It emerged as an inspiration from the electric power grid and is an alternative to cluster computing [11, 75]. Instead of the homogeneous resources of an organisation, it utilises the heterogeneous resources of different organisations or individuals. The Literature broadly visualises two views of grid environments named static and dynamic. A static grid integrates resources of partner organisations. The resources it uses are heterogeneous in nature, and these systems are mostly classified as Client Multi Server (CMS) systems that are integrated by high bandwidth links using advanced communication and management patterns. The core of a dynamic Grid is called virtual organisation (VO), a temporary alliance of distributed heterogeneous resources of different organisations and individuals over the Internet. These resources are used to solve problems by adopting common usage policies [77, 78, 122]. The important characteristics of a Grid include multiple administrative domains and autonomy, heterogeneity, scalability, and dynamicity/adaptability [78]. Resource management and scheduling are the most primitive and challenging issues in Grid Computing and need special attention to better utilise the Grid Infrastructure [46, 94, 129].

In Grid Computing, a problem solution is initiated by a user via a Grid Resource Broker. On behalf of a user, the broker performs resource discovery and scheduling, and assigns application jobs to distributed resources [78]. A resource owner registers resources with Grid Information Service (GIS) with their usage policies [47]. The Grid Resource Broker accesses GIS to find suitable resources for a solution to a problem against deadline and budget constraints. The introduction of economy to grid infrastructure brings additional challenging aspects that need to be resolved. It provides incentives for resource owners to participate in grid environment. It also leads the grid computing (using the Internet as an underlying technology) from being a computing infrastructure to a business platform. Different computational economy frameworks and algorithms for resource management are presented to cope with these challenges [27, 29, 224]. Service oriented grids have shown a great impact on Grid Computing [232]. A revolutionary approach of grid infrastructure is presented in [9].

A.3 Grid Architecture and Existing Solutions

To define and integrate system components, a number of architectural design issues need proper attention. A Grid uses cross-organisational resource sharing via a VO and, therefore, a Grid architecture requires ways to establish and manage resources [19]. Interoperability issues between resource owners and users are given special attention. Resource management at both individual and integration levels also needs special attention. Trading, security and QoS must be addressed. Discovery services, co-allocation, scheduling, and monitoring and diagnostic services issues need special considerations. To realise flexibility and reliability, data replication services and workload management must be given proper attention. Materialising these issues often result in a layered architecture that implement them as low level and high level services [75]. The architecture of grid presented by Global Grid Forum (GGF) comprises of five layers called: Application Layer, Collective Layer, Resource Layer, Connectivity Layer, and Fabric Layer [75]. Application Layer manages the construction of domain specific applications and utilises lower layer services. To facilitate layered integration, a number of Application Provider Interfaces (APIs) and Software Development Kits (SDKs) are provided at different layers, including application layer. Collective Layer is responsible for global resource management and interaction with resources. It implements a variety of shared behaviours. Resource Layer handles issues of a single resource by utilising communication, information, and management protocols to control issues such as accounting, monitoring and secure negotiation. Connectivity Layer manages core communication issues. It requires communication and authentication protocols. Communication protocols assist in data exchange between fabric layers of resources. Authentication protocols guarantee secure authentication and data exchange between a user and resources. Fabric Layer defines shared physical and logical resources. Logical resources (such as a computer cluster) utilise their own internal protocols for distributed networks.

Different implementations exist that manage the issues in different numbers of layers. Baker et al. [11] describe such an implementation in terms of Applications, User Level Middleware, Core Middleware and Fabric layers. Application layer handles applications and portals. User Level Middleware performs resource selection, management, and aggregation. It implements resource broker and provides developmental environments and tools. Core Middleware implements distributed coupling services such as security, information, data, trading and QoS. Fabric Layer concerns heterogeneous resources and their local management. A number of services and infrastructures are implemented based on these layers, such as schedulers, brokers, trading servers, and programming environments. The current implementations of integrated grid environments include NetSolve [33], Ninf [191] and Unicore [58]. Globus [76], Gridbus [26, 28] and Legion [90] are the implementations of core middleware services. Condor-G [79] and Nimrod-G [29] are user level implementations of scheduling services. Programming environments include MPICH G, Nimrod Parameter Programming tools and Cactus [25]. The well known grid application development efforts include European DataGrid [97], SETI@Home [7] and Virtual Laboratory [30]. A number of other architectures for similar issues are presented with different orientations in [9, 102, 185].

Bibliography

[1] IEEE Std 1516.1-2000. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - Federate Interface Specifications. Technical report, IEEE., Piscataway, NJ, USA, 2001.

[2] IEEE Std 1516.1-2000. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - Framework and Rules. Technical report, IEEE., Piscataway, NJ, USA, 2001.

[3] Dewan Tanvir Ahmed and Shervin Shirmohammadi. A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Dewan Tanvir Ahmed and Shervin Shirmohammadi. A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments. In *IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2008.*, pages 86 –91, July 2008.

[5] Thor Alexander, editor. *Massively Multiplayer Game Development 2*. Charles River Media, 2005.

[6] R. M. Andreu, P. G. Lopez, C. P. Gavalda, and A. F. G. Skarmeta. Tracking the Evaluation of Collaborative Virtual Environments. *UPGRADE*, 8(2), 2006.

[7] SETI@HOME (A Grid Application). `http://setiathome.berkeley.edu/`. last accesed in December, 2011.

[8] AssetServer. `http://opensimulator.org/wiki/AssetServer`. last accesed in December, 2011.

[9] Malcolm Atkinson, David DeRoure, Alistair Dunlop, Geoffrey Fox, Peter Henderson, Tony Hey, Norman Paton, Steven Newhouse, Savas Parastatidis, Anne Trefethen, Paul Watson, and Jim Webber. Web Service Grids: An Evolutionary Approach. *Concurrency and Computation: Practice and Experience*, 17(2-4):377–389, 2005.

[10] Avatar. `http://wiki.secondlife.com/wiki/Avatar`. last accesed in February, 2012.

[11] M. Baker, R. Buyya, and D. Laforenza. Grids and Grid Technologies for Wide-Area Distributed Computing. *International Journal of Software: Practice and Experience*, 32(15):1437–1466, 2002.

[12] Miranda Baladi, Henry Vitali, Georges Fadel, Joshua Summers, and Andrew Duchowski. A Taxonomy for the Design and Evaluation of Networked Virtual Environments: Its Application to Collaborative Design. *International Journal on Interactive Design and Manufacturing*, 2(1):17–32, 2008.

[13] Rajesh Krishna Balan, Maria Ebling, Paul Castro, and Archan Misra. Matrix: Adaptive Middleware for Distributed Multiplayer Games. volume 3790/2005 of *Lecture Notes in Computer Science*, pages 390–400. Springer Berlin/Heidelberg, 2005.

[14] Woodrow Barfield and Thomas A. Furness III, editors. *Virtual Environments and Advanced Interface Design.* Oxford University Press, Inc., New York, NY, USA, 1995.

[15] Jon Louis Bentley. Multidimensional Divide-and-Conquer. *Communications of the ACM*, 23(4):214–229, April 1980.

[16] Helmut Berger, Michael Dittenbach, Dieter Merkl, Anton Bogdanovych, Simeon Simoff, and Carles Sierra. Opening New Dimensions for e-Tourism. *Virtual Reality*, 11(2):75–87, June 2007.

[17] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association.

[18] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, August 2004.

[19] David Hilley Bikash Agarwalla, Nova Ahmed and Umakishore Ramachandran. Streamline: Scheduling Streaming Applications in a Wide Area Environment. *Multimedia Systems*, 13(1):69–85, 2007.

[20] C. M. Bowman, D. Lake, and J. Hurliman. Designing Extensible and Scalable Virtual World Platforms. In *Extensible Virtual Worlds Workshop (X10)*, 2010.

[21] D. Bruneo, A. Zaia, and A. Puliafito. Agent-based Middleware to Access Multimedia Services in a Grid Environment. *Multiagent and Grid Systems*, 1(1):41–59, January 2005.

[22] R. E. Bryant. Simulation of Packet Communication Architecture Computer Systems. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977.

[23] Steve Bryson. Virtual Reality Applications. chapter Approaches to the Successful Design and Implementation of VR Applications, pages 3–15. Academic Press Ltd., London, UK, 1995.

[24] A. M. Burlamaqui, M. A. M.S. Oliveira, A. M. G. Goncalves, G. Lemos, and J. C. De Oliveira. A Scalable Hierarchical Architecture for Large Scale Multi User Virtual Environments. In *IEEE International Conference on Virtual Environment, Human Computer Interfaces and Measurement Systems*, pages 114–119, 2006.

[25] R. Buyya. *Economic Based Distributed Resource Management and Scheduling for Grid Computing.* PhD in Computer Science, School of CS&SE - Monash University, Melbourne, Australia, 2002.

[26] R. Buyya. The Gridbus Toolkit: Enabling Grid Computing and Business. http://www.cloudbus.org/middleware/, 2008.

[27] R. Buyya, D. Abramson, and J. Giddy. An Economy Driven Resource Management Architecture for Global Computation Power Grids. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, pages 239–248, Las Vegas, USA, 2000. CSREA Press.

[28] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *Proceedings of 1st IEEE International Workshop on Grid Economics and Business Models, (GECON 2004)*, pages 19–66, 2004.

[29] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 1*, Beijing, China, 2000.

[30] Rajkumar Buyya, Kim Branson, Jon Giddy, and David Abramson. The Virtual Laboratory: A Toolset to Enable Distributed Molecular Modelling for Drug Design on the World-Wide Grid. *Concurrency and Computation: Practice and Experience*, 15(1):1–25, 2003.

[31] W. Cai, G. Li, S. J. Turner, B.-S. Lee, and L. Liu. Automatic Construction of Hierarchical Federations Architecture. In *DS-RT '02: Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 50–58, Washington, DC, USA, 2002. IEEE Computer Society.

[32] Wentong Cai, Stephen J. Turner, and Boon Ping Gan. Hierarchical Federations: An Architecture for Information Hiding. In *PADS '01: Proceedings of*

the fifteenth workshop on Parallel and distributed simulation, pages 67–74, Washington, DC, USA, 2001. IEEE Computer Society.

[33] Henri Casanova and Jack Dongarra. Netsolve: a Network-Enabled Server for Solving Computational Science Problems. *The International Journal of High Performance Computing Applications*, 11(3):212–223, 1997.

[34] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[35] Luther Chan, James Yong, Jiaqiang Bai, Ben Leong, and Raymond Tan. Hydra: A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, NetGames '07, pages 37–42, New York, NY, USA, 2007. ACM.

[36] K. M. Chandy and J. Misra. Distributed Simulations: A Case Study in Design and Verifications of Distributed Programs. *IEEE Transactions on Software Engineering*, 5(5):440–452, 1978.

[37] F. Chang, C.M. Bowman, and W. Feng. XPU: A Distributed Architecture for Metaverses. Technical report, Department of Computer Science, Portland State University, 2010. Technical Report 10-04.

[38] Jin Chen, Baohua Wu, Margaret Delap, Björn Knutsson, Honghui Lu, and Cristiana Amza. 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 '05, pages 289–300, New York, NY, USA, 2005. ACM.

[39] Roman Chertov and Sonia Fahmy. Optimistic Load Balancing in a DVE. In *16th International Workshop on Network and Operating System Support for Digital Audio and Video*, Newport, Rhode Island, 2006. ACM New York, NY, USA.

[40] Comparison of Virtual Worlds. http://www.virtualenvironments.info. last accesed in December, 2011.

[41] Anthony Cramp, John P. Best, and Michael J. Oudshoorn. Time Management in Hierarchical Federation Communities. In *2002 Fall Simulation Interoperability Workshop*, 2002.

[42] Anthony Cramp and Michael J. Oudshoorn. Employing Hierarchical Federation Communities in the Virtual Ship Architecture. In *Twenty-Fifth Australasian Computer Science Conference*, pages 41–50, Melbourne, Australia, 2002.

[43] Croquet: Current Release. `http://www.opencroquet.org/index.php/System_Overview`. last accesed in December, 2011.

[44] Croquet: Introduction. `http://en.wikipedia.org/wiki/Croquet_project`. last accesed in December, 2011.

[45] CSI Virtual World Archive. `http://labs.greenbush.us/CSI-Opensim.zip`. last accesed in December, 2011.

[46] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A Resource Management Architecture for Metacomputing Systems. *Job Scheduling Strategies for Parallel Processing*, 1459/1998:62–82, 1998.

[47] Karl Czajkowski, Carl Kesselman, Steven Fitzgerald, and Ian Foster. Grid Information Services for Distributed Resource Sharing. In *(HPDC '01): 10th IEEE International Symposium on High Performance Distributed Computing*, page 0181, San Francisco, California, USA, 2001.

[48] Judith S. Dahmann, Richard M. Fujimoto, and Richard M. Weatherly. The Department of Defense High Level Architecture. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 142–149, Washington, DC, USA, 1997. IEEE Computer Society.

[49] Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar, and Kevin McGee. NetEffect: A Network Architecture for Large-Scale Multi-User Virtual World. In *ACM Symposium on Virtual Reality Software and Technology*, pages 157–163. ACM New York, NY, USA, 1997.

[50] Roy C. Davies. Adapting Virtual Reality for the Participatory Design of Work Environments. *Computer Supported Cooperative Work*, 13(1):1–33, January 2004.

[51] Jauvane C. de Oliveira and Nicolas D. Georganas. VELVET: An Adaptive Hybrid Architecture for Very Large Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 12(6):555–580, 2003.

[52] Rina Dechter. From Local to Global Consistency. *Artificial Intelligence*, 55(1):87–108, 1992.

[53] OpenSim: Definitions. `http://opensimulator.org/wiki/Talk:Definitions`. last accesed in December, 2011.

[54] Second Life: Agent Domain. `https://wiki.secondlife.com/wiki/Agent_Domain`. last accesed in December, 2011.

[55] P. du Pont. Virtual Reality in Engineering. chapter Applied Virtual Reality, pages 153–167. Institution of Electrical Engineers, Stevenage, UK, 1993.

[56] EducationSim Archive. `http://odomia.com/educasim.tar.gz`. last accesed in December, 2011.

[57] Tulga Ersal, Mark Brudnak, Ashwin Salvi, Jeffrey L. Stein, Zoran Filipi, and Hosam K. Fathy. Development of an Internet-Distributed Hardware-in-the-Loop Simulation Platform for an Automotive Application. *ASME Conference Proceedings*, 2009:73–80, 2009.

[58] Dietmar W. Erwin. UNICORE: A Grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[59] FairieCastle Archive. `http://www.mediafire.com/file/l70hqvtcb8ub7z6/FairieCastle-v0.1.oar`. last accesed in December, 2011.

[60] Umar Farooq and John Glauert. ARA: An Aggregate Region Assignment Algorithm for Resource Minimisation and Load Distribution in Virtual Worlds. In *NDT '09: Proceedings of the first IEEE International Conference on Networked Digital Technologies*, pages 404–410, 2009.

[61] Umar Farooq and John Glauert. Joint Hierarchical Nodes based User Management (JoHNUM) Infrastructure for the Development of Scalable and Consistent Virtual Worlds. In *DS-RT '09: Proceedings of the 13th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications*, pages 105–112, Washington, DC, USA, 2009. IEEE Computer Society.

[62] Umar Farooq and John Glauert. Managing Scalability and Load Distribution for Large Scale Virtual Worlds. In *Proceedings of the UEA School of Computing Sciences Symposium*, pages 20–27, 2009.

[63] Umar Farooq and John Glauert. A Decentralised Synchronisation Approach for Complex Hierarchical Models of Virtual Worlds. In *PDCS '10: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 218–224, 2010.

[64] Umar Farooq and John Glauert. A Dynamic Load Distribution Algorithm for Virtual Worlds. *Journal of Digital Information Management*, 8(3):181–189, June 2010.

[65] Umar Farooq and John Glauert. A Hybrid Infrastructure for Scalable and Consistent Virtual Worlds. In *WSC '10: Proceedings of the IEEE 42nd Winter Simulation Conference*, 2010.

[66] Umar Farooq and John Glauert. Time Management for Virtual Worlds based on Constrained Communication Model. In *NetGames '10: Proceedings of the 9th ACM/IEEE Annual Workshop on Network and System Support for Games*, pages 18:1–6, 2010.

[67] Umar Farooq and John Glauert. Scalable Virtual Worlds: An Extension to the OpenSim Architecture. In *ICCNIT '11: Proceedings of the IEEE International Conference on Computer Networks and Information Technology*, pages 29–34, 2011.

[68] J. Filsinger. HLA Security Guard Federate. In *1997 Spring Simulation Interoperability Workshop*, Orlando, Florida, USA, March 1997.

[69] M.C. Fischer. Aggregate Level Simulation Protocol (ALSP) Managing Confederation Development. In *Winter Simulation Conference Proceedings, 1994.*, pages 775 – 780, Dec 1994.

[70] Xavier Fischer and Daniel Coutellier. *Editorial. International Journal on Interactive Design and Manufacturing*, 1(1):1–4, 2007.

[71] Flash Flood Fill Implementation. `http://www.emanueleferonato.com/2008/06/06/flash-flood-fill-implementation/`. last accesed in December, 2011.

[72] Flood Fill Algorithm. `http://en.wikipedia.org/wiki/Flood_fill`. last accesed in December, 2011.

[73] Flood Fill: Lode's Computer Graphics Tutorial. `http://lodev.org/cgtutor/floodfill.html`. last accesed in December, 2011.

[74] Basic Support for Cooperative Work. http://www.bscw.de/english/index.html. last accesed in December, 2011.

[75] I. Foster. What is the Grid: A Three Point CheckList. *GRIDToday*, 2002.

[76] Ian Foster and Carl Kesselman. Globus: a Metacomputing Infrastructure Toolkit. *The International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

[77] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. 1999.

[78] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid - Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):290–315, 2001.

[79] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[80] R. M. Fujimoto. Lookahead in Parallel Discrete Event Simulation. In *International Conference on Parallel Processing*, pages 34–41, 1988.

[81] R. M. Fujimoto, T. Mclean, K. Perumalla, and I. Tacic. Design of High Performance RTI Software. In *International Workshop on Distributed Simulation and Real Time Applications*, pages 89–96, 2000.

[82] Richard M. Fujimoto. Time Management in the High Level Architecture. *Simulation*, 71(6):388–400, 1998.

[83] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems.* Wiley Interscience, 2000.

[84] Richard M. Fujimoto. Parallel and Distribution Simulation Systems. In *2001 Winter Simulation Conference*, pages 147–157, 2001.

[85] Deborah A. Fullford. Distributed Interactive Simulation: Its Past, Present, and Future. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 179–185, Washington, DC, USA, 1996. IEEE Computer Society.

[86] T. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[87] Zihui Ge, Ping Ji, and Prashant Shenoy. Design and Analysis of a Demand Adaptive and Locality Aware Streaming Media Server Cluster. *Multimedia Systems*, 13(3):235–249, 2007.

[88] Genecys: Introduction and Downloads. http://sourceforge.net/projects/genecys/. last accesed in December, 2011.

[89] Genecys: The Official Website. http://www.genecys.org/. last accesed in December, 2011.

[90] Andrew S. Grimshaw, Wm. A. Wulf, and CORPORATE The Legion Team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[91] Groove 2007: A P2P Co-operative Environment (Upgraded to "SharePoint Workspace 2010"). http://office.microsoft.com/en-us/groove/default.aspx. last accesed in December, 2011.

[92] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White. Scalability for Virtual Worlds. In *Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE '09)*, pages 1311–1314, 2009.

[93] Thorsten Hampel, Thomas Bopp, and Robert 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 '06, New York, NY, USA, 2006. ACM.

[94] Gernot Heiser, Fondy Lam, and Stephen Russell. Resource Management in the Mungi Single-Address-Space Operating System. In *Proceedings of Australasian Computer Science Conference*, Perth, Australia, 1998.

[95] J. Helmer. Second Life and Virtual Worlds. Technical report, Learning Light Limited, UK., 2007.

[96] M. Hori, T. Iseri, K. Fujikawa, S. Shimojo, and H. Miyahara. Scalability Issues of Dynamic Space Management for Multiple-Server Networked Virtual Environments. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, volume 1, pages 200–203, 2001.

[97] Wolfgang Hoschek, Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Data Management in an International Data Grid Project. In *Grid Computing - Grid 2000*, volume 1971/1997, pages 333–361, 2000.

[98] The Virtual Environment: Habbo Hotel. `http://www.habbohotel.com`. last accesed in December, 2011.

[99] Toby Howard, Roger Hubbold, and Alan Murta. MAVERIK: A Virtual Reality System for Research and Teaching. *Presence: Teleoperators and Virtual Environments*, 10(1):22–34, 2006.

[100] HTTP. `http://www.w3.org/Protocols/rfc2616/rfc2616.html`. last accesed in April, 2012.

[101] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *IEEE Network*, 20(4):22–31, 2006.

[102] Jiung Yao Huang, Yi Chang Du, and Chien Min Wang. Design of the Server Cluster to Support Avatar Migration. In *IEEE Virtual Reality*, pages 7–14. IEEE Computer Society Washington, DC, USA, 2003.

[103] IDC. Butterfly.net: Powering next generation gaming with on-demand computing. Technical report, IBM: An IDC e-Business Case Study, 2004.

[104] Quake II. http://www.idsoftware.com/games/quake/quake2/. last accesed in December, 2011.

[105] Takuji Iimura, Hiroaki Hazeyama, and Youki 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 '04, pages 116–120, New York, NY, USA, 2004. ACM.

[106] Importing OARs into megaregions. `http://www.metaverseink.com/blog/?p=28`. last accesed in December, 2011.

[107] Intel Research Labs. ScienceSim: A Virtual Environment for Collaborative Visualization and Experimentation. White paper, Intel Labs, 2010.

[108] Argentum Online: Introduction and Downloads. `http://sourceforge.net/project/showfiles.php?group_id=67718`. last accesed in December, 2011.

[109] Arianne: Introduction. http://arianne.sourceforge.net/. last accesed in December, 2011.

[110] Beyond 2: Introduction. http://asbahr.com/beyond.html. last accesed in December, 2011.

[111] Diamonin: Introduction and Downloads. http://www.daimonin.com/. last accesed in December, 2011.

[112] FreeTribes: Introduction. http://developer.berlios.de/projects/freetribes/. last accesed in December, 2011.

[113] Irrlicht Engine: Introduction and Downloads. http://irrlicht.sourceforge.net. last accesed in December, 2011.

[114] Isotope: Introduction, Downloads, and Documentation. http://isotope.sourceforge.net/. last accesed in December, 2011.

[115] Janthus: Introduction and Downloads. http://janthus.sourceforge.net/. last accesed in December, 2011.

[116] Quake II: Introduction. `http://en.wikipedia.org/wiki/Quake_II`. last accesed in December, 2011.

[117] WarZone: Introduction. http://wz2100.net/. last accesed in December, 2011.

[118] Inventory. `http://wiki.secondlife.com/wiki/Inventory`. last accesed in April, 2012.

[119] Sankar Jayaram, Uma Jayaram, Young Jun Kim, Charles DeChenne, Kevin W. Lyons, Craig Palmer, and Tatsuki Mitsui. Industry Case Studies in the Use of Immersive Virtual Assembly. *Virtual Reality*, 11(4):217–228, 2007.

[120] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(2):404–425, 1985.

[121] N. Johnson. The Educational Potential of SecondLife. The Ohio State University, USA, 2006.

[122] Joshy Joseph and Craig Fellenstein, editors. *Grid Computing*. IBM Series, 2004.

[123] Priscilla Kan John and Alban Grastien. Local Consistency and Junction Tree for Diagnosis of Discrete-Event Systems. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 209–213, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[124] The Virtual Environment: Kaneva. `http://www.keneva.com`. last accesed in December, 2011.

[125] Beob Kyun Kim and Kang Soo You. A Hierarchical Map Partition Method in MMORPG based on Virtual Map. In *Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops*, volume 4331/2006 of *Lecture Notes in Computer Science*, pages 813–822. Springer Berlin/Heidelberg, 2006.

[126] Jae-Hyun Kim and Tag Gon Kim. Proposal of High Level Architecture Extension. In *Artificial Intelligence and Simulation*, pages 128–137. Springer Berlin / Heidelberg, 2005.

[127] Jae-Hyun Kim and Tag Gon Kim. Hierarchical HLA: Mapping Hierarchical Model Structure into Hierarchical Federation. *M&S-MTSA'06*, pages 75 – 80, July 2006.

[128] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *IEEE INCOMM*, pages 107–112, 2004.

[129] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software: Practice and Experience*, 32(2):135–164, 2000.

[130] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating Computer Simulation Systems: An introduction to the High Level Architecture*. Prentice Hall PTR, 1999.

[131] Sanjeev Kumar, Jatin Chhugani, Changkyu Kim, Daehyun Kim, Anthony Nguyen, Pradeep Dubey, Christian Bienia, and Youngmin Kim. Second Life and the New Generation of Virtual Worlds. *Computer*, 41(9):46–53, September 2008.

[132] Dan Lake, Mic Bowman, and Huaiyu Liu. Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, NetGames '10, pages 19:1–19:6, Piscataway, NJ, USA, 2010. IEEE Press.

[133] Simon St. Laurent, Joe Johnston, Edd Dumbill, and Dave Winer. *Programming Web Services with XML-RPC*. O'Reilly Media, 2001.

[134] Dongman Lee, Mingyu Lim, and Seyunhyun Han. ATLAS: A Scalable Network Framework for Distributed Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 16(2):125–156, 2007.

[135] Kang-Won Lee, Bong-Jun Ko, and Seraphin Calo. Adaptive Server Selection for Large Scale Interactive Online Games. In *14th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 152–157, Cork, Ireland, 2004. ACM New York, NY, USA.

[136] Kyungmin Lee and Dongman Lee. A Scalable Dynamic Load Distribution Scheme for Multi-Server Distributed Virtual Environment Systems With

Highly-Skewed User Distribution. In *ACM Symposium on Virtual Reality Software and Technology*, pages 160–168, Osaka, Japan, 2003. ACM New York, NY, USA.

[137] Second Life. `http://en.wikipedia.org/wiki/Second_Life`. last accesed in December, 2011.

[138] Buquan Liu, Yiping Yao, and Huaimin Wang. An Efficient Algorithm in the HLA Time Management. In *Proceedings of the 2007 Winter Simulation Conference*, pages 585–593, 2007.

[139] H. Liu and M. Bowman. Scale Virtual Worlds through Dynamic Load Balancing. In *DS-RT '10: Proceedings of the 2010 14th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 43–52, Washington, DC, USA, 2010. IEEE Computer Society.

[140] H. Liu, M. Bowman, R. Adams, J. Hurliman, and D. Lake. Scaling Virtual Worlds: Simulation Requirements and Challenges. In *Proceedings of the 2010 Winter Simulation Conference (WSC '10)*, pages 778–790, 2010.

[141] Load Balancer Project. `http://forge.opensimulator.org/gf/project/loadbalancer/`. last accesed in December, 2011.

[142] Fengyun Lu, Simon Parkin, and Graham Morgan. Load Balancing for Massively Multiplayer Online Games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, New York, NY, USA, 2006. ACM.

[143] John C. S. Lui and M. F. Chan. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Transaction on Parallel and Distribution Systems*, 13(3):193–211, 2002.

[144] Gerry Magee, Graham Shanks, and Pete Hoare. Hierarchical Federations. In *Simulation Interoperability Spring Workshop*, Orlando, Florida, March 1999.

[145] Massiv: Documentation and Downloads. `http://forge.objectweb.org/project/showfiles.php?group_id=149`. last accesed in December, 2011.

[146] Massiv (Massively Multiplayer Online Game Middleware). http://massiv.ow2.org/. last accesed in December, 2011.

[147] Maverik: A MicroKernal for Large Scale VEs. `http://linuxjournal.com/article/4035`. last accesed in December, 2011.

[148] Maya Pyramid Archive. `http://www.gomaya.com/glyph/opensim_dp/maya3.oar`. last accesed in December, 2011.

[149] D. C. Miller and J. A. Thorpe. SIMNET: The Advent of Simulator Networking. *Proceedings of the IEEE*, 83(8):1114–1123, 1995.

[150] Mark P. Mobach. Do Virtual Worlds Create Better Real Worlds? *Virtual Reality*, 12(3):163–179, 2008.

[151] Björn Möller and Lennart Olsson. Practical Experiences from HLA 1.3 to HLA IEEE 1516 Interoperability. In *2004 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, September 2004.

[152] P. Morillo, M. Fernandez, and N. Pelechano. A Grid Representation for Distributed Virtual Environments. In *GRID COMPUTING*, volume 2970/2004 of *Lecture Notes in Computer Science*, pages 182–189. Springer Berlin/Heidelberg, 2004.

[153] P. Morillo, J. M. Orduna, and J. Duato. A Scalable Synchronization Technique for Distributed Virtual Environments based on Networked Server Architecture. In *International Conference on Parallel Processing Workshops*, pages 74–81, 2006.

[154] Matthias Müller, Leonard McMillan, Julie Dorsey, and Robert Jagnow. Real-time Simulation of Deformation and Fracture of Stiff Materials. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 113–124, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[155] Michael D. Myjak, Duncan Clark, and Tom Lake. RTI Interoperability Study Group Final Report. In *1999 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, September 1999.

[156] Michael D. Myjak and Sean T. Sharp. Implementation of Hierarchical Federations. In *1999 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, September 1999.

[157] The Virtual Environment: Neopets. `http://www.neopets.com`. last accesed in December, 2011.

[158] netPanzer. http://www.netpanzer.org/. last accesed in December, 2011.

[159] Beatrice Ng, Antonio Si, Rynson W. H. Lau, and Frederick Li. A Multi-server Architecture for Distributed Virtual Walkthrough. In *ACM Symposium on Virtual Reality Software and Technology*, pages 163–170. ACM New York, NY, USA, 2002.

[160] D. M. Nicol. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. *Journal of the Association for Computing Machinery*, 40(2):304–333, 1993.

[161] GU Ning and MAHER Mary Lou. Dynamic Designs of 3D Virtual Worlds using Generative Design Agents. In *Computer Aided Architectural Design Futures*, pages 239–248, 2005.

[162] Giovanni Novelli, Giuseppe Pappalardo, Corrado Santoro, and Emiliano Tramontana. A Grid-based Infrastructure to Support Multimedia Content Distribution. In *Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks*, UPGRADE '07, pages 57–64, New York, NY, USA, 2007. ACM.

[163] Second Life: Structure of Region Domain. `https://wiki.secondlife.com/wiki/Region_Domain`. last accesed in December, 2011.

[164] World of Warcraft. http://us.battle.net/wow/en/. last accesed in December, 2011.

[165] Cory Ondrejka. A PIECE OF PLACE: Modeling the Digital on the Real in Second Life. Working Paper Series. University of South California, 2004.

[166] The Virtual Environment: Gaia Online. http://www.gaiaonline.com. last accesed in December, 2011.

[167] Open Grid Protocol. http://wiki.secondlife.com/wiki/Open_Grid_Protocol. last accesed in December, 2011.

[168] Open Virtual Collaboration Environment (OpenVCE.net). http://openvce.net. last accesed in December, 2011.

[169] OpenCobalt. http://en.wikipedia.org/wiki/Open_Cobalt. last accesed in December, 2011.

[170] OpenCobalt: The Official Website. http://www.opencobalt.org/. last accesed in December, 2011.

[171] OpenMetaverse. http://openmetaverse.org/. last accesed in December, 2011.

[172] OpenSim Architectures. http://opensimulator.org/wiki/Configuration. last accesed in December, 2011.

[173] OpenSim Archive (OAR) Functionality. http://opensimulator.org/wiki/OpenSim_Archives. last accesed in December, 2011.

[174] OpenSim: Existing Worlds Content. http://www.opensimworlds.com/. last accesed in December, 2011.

[175] OpenSim: Megaregions. http://opensimulator.org/wiki/Setting_Up_Mega-Regions. last accesed in December, 2011.

[176] OpenSim: UGAIM Services and Region Server. http://opensimulator.org/wiki/OpenSim:Introduction_and_Definitions. last accesed in December, 2011.

[177] OpenSimulator (OpenSim): An introduction. http://opensimulator.org/wiki/Main_Page. last accesed in December, 2011.

[178] Ke Pan, Stephen John Turner, Wentong Cai, and Zengxiang Li. A Hybrid HLA Time Management Algorithm Based on Both Conditional and Unconditional Information. In *PADS '08: Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 203–211, Washington, DC, USA, 2008. IEEE Computer Society.

[179] Ke Pan, Stephen John Turner, Wentong Cai, and Zengxiang Li. A Hybrid HLA Time Management Algorithm Based on Both Conditional and Unconditional Information. *Simulation*, 85(9):559–573, 2009.

[180] Kalyan S. Perumalla. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Proceedings of the 38th conference on Winter simulation*, WSC '06, pages 84–95, 2006.

[181] K. Prasetya and Z. D. Wu. Performance Analysis of Game World Partitioning Methods for Multiplayer Mobile Gaming. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 72–77, New York, NY, USA, 2008. ACM.

[182] Primitive (Prim). `http://wiki.secondlife.com/wiki/Primitive`. last accesed in February, 2012.

[183] Quake II: Manual. `http://quakebase.ktu.edu/quake_stuff/quake2/manual/Manual.html`. last accesed in December, 2011.

[184] Peter Quax, Jeroen Dierckx, Bart Cornelissen, Gert Vansichem, and Wim Lamotte. Dynamic Server Allocation in a Real-life Deployable Communications Architecture for Networked Games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 66–71, New York, NY, USA, 2008. ACM.

[185] Vytautas Reklaitis, Kazys Baniulis, and Toshio Okamoto. Shaping e-Learning Applications for a Service-Oriented Grid. In *Proceeding of the 2005 conference on Towards the Learning Grid: Advances in Human Learning Services*, pages 98–104, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[186] RemoteAdmin Functionality. `http://opensimulator.org/wiki/RemoteAdmin`. last accesed in December, 2011.

[187] Renting/Buying Land in OpenSim. `http://wiki.secondlife.com/wiki/Private_Estate_Management_Companies`. last accesed in December, 2011.

[188] Abdennour El Rhalibi, Madjid Merabti, and Yuanyuan Shen. AoIM in Peer-to-Peer Multiplayer Online Games. In *ACM SIGCHI International conference on Advances in computer entertainment technology*, Hollywood, California, 2006. ACM New York, NY, USA.

[189] P. Rosedale and C. Ondrejka. Enabling Player Created Online Worlds with Grid Computing and Streaming. Gamasutra, 2003.

[190] Maria Roussou, Martin Oliver, and Mel Slater. The Virtual Playground: an Educational Virtual Reality Environment for Evaluating Interactivity and Conceptual Learning. *Virtual Reality*, 10(2):227–240, 2006.

[191] Mitsuhisa Sato, Hidemoto Nakada, Satoshi Sekiguchi, Satoshi Matsuoka, Umpei Nagashima, and Hiromitsu Takagi. Ninf: A Network based Information Library for Global World-wide Computing Infrastructure. In *High-Performance Computing and Networking*, volume 1225/1997, pages 491–502, San Diego, CA, 1997.

[192] Scene. `http://opensimulator.org/wiki/Getting_Started_with_Region_Modules`. last accesed in April, 2012.

[193] ScienceSim Performance Tests. `http://sciencesim.com/wiki/doku.php/opensim/performance_tests`. last accesed in December, 2011.

[194] Script. `http://secondlife.wikia.com/wiki/Script`. last accesed in April, 2012.

[195] Second Life Grid: Concepts. `http://wiki.secondlife.com/wiki/Grid#Grid`. last accesed in December, 2011.

[196] Second Life Grid Extension: Architecture Working Group. `https://wiki.secondlife.com/wiki/Architecture_Working_Group`. last accesed in December, 2011.

[197] Second Life Grid: Introduction. `http://wiki.secondlife.com/wiki/Second_Life_Grid`. last accesed in December, 2011.

[198] Second Life Grid: Motivation for Extension. `http://wiki.secondlife.com/wiki/Project_Motivation`. last accesed in December, 2011.

[199] Second Life Grid: Today and Tomorrow. `https://wiki.secondlife.com/wiki/Structural_Design_Overview`. last accesed in December, 2011.

[200] Second Life: Local and Offline Content. `https://wiki.secondlife.com/wiki/Running_at_Home_and_Offline`. last accesed in December, 2011.

[201] Second Life: The Official Website. http://www.secondlife.com/. last accesed in December, 2011.

[202] Second Life: Central Services. `http://wiki.secondlife.com/wiki/Central_Services`. last accesed in December, 2011.

[203] Shervin Shirmohammadi, Ihab Kazem, Dewan Tanvir Ahmed, Madeh El-Badaoui, and Jauvane C. De Oliveira. A Visibility-Driven Approach for Zone Management in Simulations. *Simulation*, 84(5):215–229, 2008.

[204] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation.* ACM Press/Addison-Wesley Publishing Co., 1999.

[205] Kay M. Stanney, Ronald R. Mourant, and Robert S. Kennedy. Human Factors Issues in Virtual Environments: A Review of the Literature. *Presence: Teleoperators and Virtual Environments*, 7(4):327–351, August 1998.

[206] J. Steinman. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation. In *Advances in Parallel and Distributed Simulation*, pages 95–103, 1991.

[207] SUN. Game Server Technology. White paper, SUN Microsystems Inc., June 2004.

[208] The Virtual Environment: Forterra Systems. `http://www.forterrainc.com`. last accesed in December, 2011.

[209] Duong Nguyen Binh Ta, Suiping Zhou, and Haifeng Shen. Greedy Algorithms for Client Assignment in Large Scale Distributed Virtual Environments. *Simulation*, 84(10-11):521–533, 2008.

[210] Simon J. E. Taylor, Jon Saville, and Rajeev Sudra. Developing Interest Management Techniques in Distributed Interactive Simulation using Java. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future - Volume 1*, WSC '99, pages 518–523, New York, NY, USA, 1999. ACM.

[211] The Virtual Environment: ActiveWorlds. http://www.activeworlds.com/. last accesed in December, 2011.

[212] The Virtual Environment: Barbie Girls. `http://www.barbiegirls.com`. last accesed in December, 2011.

[213] The Virtual Environment: Club Penguin. `http://www.clubpenguin.com`. last accesed in December, 2011.

[214] The Virtual Environment: There. `http://www.there.com`. last accesed in December, 2011.

[215] The Virtual Environment: Zwinky. `http://www.zwinky.com`. last accesed in December, 2011.

[216] Douglas Thomas and John Seely Brown. Why Virtual Worlds Can Matter. Working Paper: Institute of Network Culture, University of Southern California, 2007.

[217] Ultima Online. http://www.uoherald.com/. last accesed in December, 2011.

[218] Peter van Beek and Rina Dechter. Constraint Tightness and Looseness Versus Local and Global Consistency. *Journal of the ACM*, 44(4):549–566, July 1997.

[219] Matteo Varvello, Fabio Picconi, Christophe Diot, and Ernst Biersack. Is There Life in Second Life? In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 1:1–1:12, New York, NY, USA, 2008. ACM.

[220] Argentum Online (Non English Version). `http://ao.alkon.com.ar/`. last accesed in December, 2011.

[221] John Vince. *Virtual Reality Systems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[222] John Vince. *Introduction to Virtual Reality*. Springer, 2004.

[223] Bart De Vleeschauwer, Bruno Van Den Bossche, Tom Verdickt, Filip De Turck, Bart Dhoedt, and Piet Demeester. Dynamic Microcell Assignment for Massively Multiplayer Online Gaming. In *4th ACM SIGCOMM workshop on Network and System Support for Games*, pages 1–7, Hawthorne, NY, 2005. ACM New York, NY, USA.

[224] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[225] D. Waltz. Understanding Line Drawings of Scenes with Shadows. In Patrick Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.

[226] Tianqi Wang, Cho-Li Wang, and Francis C. Lau. An Architecture to Support Scalable Distributed Virtual Environment Systems on Grid. *Journal of Supercomputing*, 36(3):249–264, June 2006.

[227] John A. Waterworth and Eva L. Waterworth. Presence and Absence in Education VR: The Role of Perceptual Seduction in Conceptual Learning. *Themes in Education*, 1(1):7–38, 2000.

[228] BZ Flag: The Official Website. http://bzflag.org/. last accesed in December, 2011.

[229] Crossfire: The Official Website. http://crossfire.real-time.com/. last accesed in December, 2011.

[230] OpenArena: The Official Website. http://openarena.ws/. last accesed in December, 2011.

[231] WorldForge: The Official Website. http://worldforge.org/. last accesed in December, 2011.

[232] Jon B. Weissman and Byoung-Dai Lee. The Service Grid: Supporting Scalable Heterogeneous Services in Wide-Area Networks. In *Proceedings of 2001 Symposium on Applications and the Internet (SAINT'01)*, San Diego, CA, 2001.

[233] The Virtual Environment: Whyville. http://www.whyville.com. last accesed in December, 2011.

[234] Arianne: Wiki. http://stendhalgame.org/wiki/Main_Page. last accesed in December, 2011.

[235] BZ Flag: Wiki. http://my.bzflag.org/w/. last accesed in December, 2011.

[236] Crossfire: Wiki. http://wiki.metalforge.net/doku.php. last accesed in December, 2011.

[237] Irrlicht Engine: Wiki. http://www.irrlicht3d.org/wiki/. last accesed in December, 2011.

[238] OpenArena: Wiki. http://openarena.wikia.com/wiki/Main_Page. last accesed in December, 2011.

[239] WarZone: Wiki. http://warzone2100.wikia.com/wiki/Main_Page. last accesed in December, 2011.

[240] WorldForge: Wiki. http://wiki.worldforge.org/wiki/Main_Page. last accesed in December, 2011.

[241] W. Winn, M. Windschitl, R. Fruland, and Y. Lee. When Does Immersion in a Virtual Environment help Students Construct Understanding? In *International Conference on Learning Sciences*, pages 497–503, 2002.

[242] XML. `http://www.w3.org/XML/`. last accesed in April, 2012.

[243] J. H. Zeigler, H. Praehoper, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, 2000.

[244] Yan Zhang, Zhong Zhou, and Wei Wu. A Hierarchical Time Management Mechanism for HLA-Based Distributed Virtual Environment. *Journal of Computational Information Systems*, 10(2):7–15, 2006.

[245] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with Local and Global Consistency. In *NIPS*, 2003.