Massivizing Networked Virtual Environments on Clouds

Siqi Shen

Massivizing Networked Virtual Environments on Clouds

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
woensdag 8 april 2015 om 15:00 uur

Door

Siqi SHEN

Master of Engineering in Computer Science and Technology
National University of Defense Technology, China
geboren te Zhaoan, China

This dissertation has been approved by the

promotor: Prof.dr.ir. D.H.J. Epema and
copromotor: Dr.ir. A. Iosup

Composition of the doctoral committee:

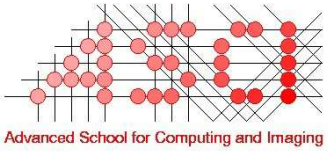
Rector Magnificus	
Prof.dr.ir. D.H.J. Epema	promotor
Dr.ir. A. Iosup	copromotor
Dr.ir. F.A. Kuipers	other, not independent member
Independent members:	
Prof.dr.ir. H.J. Sips	EWI, Technische Universiteit Delft
Dr. M. Abdallah	Pierre and Marie Curie University, France
Prof.dr. Y. Dou	National University of Defense Technology, China
Prof.dr. M.J. van Kreveld	Universiteit Utrecht
Prof.dr. E. Visser	Technische Universiteit Delft, reservelid

Published and distributed by: Siqi Shen. E-mail: siqishen@gmail.com

ISBN: 978-94-6203-815-8

Keywords: Networked virtual environments, Online games, Online social networks, Clouds, Resources scheduling, Scalability, Consistency, Characterization, Modeling, Simulation, Implementation.

Printed in The Netherlands by: CPI Wöhrmann Print Service.



The work described in this thesis has been carried out in the ASCI graduate school. ASCI dissertation series number 326.

This work was supported by the China Scholarship Council (CSC).

Acknowledgements

Since the first day I arrived in the Netherlands, it will have been 1,647 days till the day of my defense. Albeit PhD study is a long and tough journey, it is the people who experienced with me, shared with me, and helped me make the journey a pleasant one. I would like to express my gratitude to them.

I would like to thank my supervisors Dick Epema and Alexandru Iosup. Dick, thanks for being my promotor, thanks for your kindness, visions shared with me, your patience with me, and your guidance which improves my technical English writing, my research, and my thesis.

Alex, thanks for your guidance and help, during my PhD study. I remember many times when I gave you my draft paper, and in return you gave me the same paper which was full of red correction marks. Not only did I learn many research techniques from you, but also I learned views of research and life from you. They are beneficial for me now and in the future. You are my luckily golden boss :)

Henk, thanks for accepting me to become a part of the PDS group and the arrangements when I arrived at PDS. Thanks for the interesting chats with you; they make me more familiar with research, Delft, and Europe.

I would like to thank China Scholarship Council, for supporting my PhD study. I would like to thank Fernando Kuipers, Yong Dou, Henk Sips, Maha Abdallah, Marc van Kreveld for accepting to be part of my committee, and for their valuable time spent on this thesis.

I would like to thanks my collaborators: Shun-Yun Hu on scaling networked virtual environments, Yong Guo on analyzing game traces, Yunhua Deng on load-balancing and scaling networked virtual environments, Niels Brouwers on analyzing the virtual-world mobility, Adele Lu Jia, Ruud van de Bovenkamp, and Fernando Kuipers on analyzing online social networks, Assaf Israel, Walfredo Cirne, and Danny Raz on datacenter availability, and Vincent van Beek on workload characterization.

Otto, thanks for sharing many stories about the Netherlands and PDS with me and helping me with many different issues. Thanks for working with me in my PhD research. Ana, thanks for giving me advices about research and others. Boxun, thanks for the helpful discussions. I learned a lot of the differences between Europe and China from

you. Jianbin, you and me arrived at PDS at approximately the same time, thanks for experiencing the Netherlands and research with me. Adele, Yong, and Jie, thanks for making Delft warmer for me. Wing, thanks for working late in the office with me and helping me to improve my English. Sietse, it is nice to have you as my office-mate.

Nezih and Ozan, thanks for being my office-mate during the beginning of my PhD. It is interesting to discuss with you about research and cultures. Bogdan, Lipu, Niels, Alex (small), Mihai, Rahim, Johan, Jesse, Tim, Stefan, Marcin, Riccardo, Alexy, Paolo, Elvan, Boudewijn, and Arno, thanks for the lunch times, the gaming nights, and the movie nights. I enjoyed having pizza and playing games with you. Nitin, Ana, Lucia, Dimitra, Rameez, and Tamás, thanks for sharing stories of PDS with me, it is nice to chat with you.

I would like to thanks the ICT support of Paulo, Stephen, and Munire. Your help makes my research progress faster. I would like to thank Ilse, Rina, Shemara, Tamara, Franca, and Jessica. Your support simplifies many paperwork and administrative issues.

I am lucky to have many friends in the Netherlands: Kefeng, Chao, Xian, Shanfei, Ke, Jian, Yunhua, Chang, Zhang, Mingxin, Xiaoqin, Tiantian, Shanshan, Jian, Yue, Xian, Yuan, Yongjia, Chang, Liu, Wenbo, Ruijun, Jing, Yihui, Xi, Weichen, Lilan, Lei, Zhijie, Xinyu, Rongqing, Shuhong, Linfeng, Huayang, Nick, and Rodrigo. Thanks for your friendships which make the PhD journey less lonely.

Furthermore, I want to express my thanks to my teachers and friends from NUDT: Ji, Zhenbang, Liqian, Wangwei, Rui, Shengdong, Renjian, Zhaofei, Pei Fan, Jie, Longming, Yufeng, Xianjin, Jingwen, Jiahong, Banghu, Jingde, Jianchao, Yuan, and Yao. Thanks for your guidance, help, and hospitality when I went to NUDT. I also want to express my thanks to my friends in my hometown: Xiaowei, Yixin, and Chenye. Thanks for making the distance between China and the Netherlands shorter.

Last but not least, I would like express my appreciation to my parents; thanks for always love, guide, and support me. I love you.

Contents

Chapter 1

Introduction

Networked Virtual Environments (NVEs) are providing service to overall hundreds of millions users, and range from online games and enterprise training to disaster-scenario analysis and education. The predominant industry approach to the design and operation of NVEs is self-hosting, that is, to buy and operate large-scale infrastructures. This practice cannot scale when the number of users per NVE surges, is too wasteful when the number of users decreases, and is too restrictive for small companies due to up-front costs. To *massivize* NVEs, that is, to scale their architecture to allow massive amounts of users, various approaches have been proposed [137, 231]. In recent years, when many self-hosted NVEs were shut down or not even deployed [113], a new approach has emerged: *cloud-based NVEs*, that is, hosting NVEs on elastic sets of resources, provisioned flexibly, when needed, only for as long as needed, and only paying for what is consumed, from clouds. In this thesis, we analyze the workloads of NVEs and propose several mechanisms and architectural approaches to scale NVEs on clouds.

NVEs have large numbers of users. The most popular types of NVEs are online games that entertain about 700 million online users in a global market of tens of billions of Euros per year [205]. Besides in entertainment, the techniques developed for NVEs are increasingly used in enterprise training [152,235], for example using complex simulations that require cooperation across multiple continents and advanced visualizations [86], in disaster scenarios [225] such as fire-fighting in crowded neighborhoods and evacuation of large-scale disaster areas, and in education [2] such as Massive Open Online Courses.

Designers of NVE systems are increasingly searching for ways to scale their NVEs, thus being able to provide service to an ever-increasing user-base. In recent years, Cloud computing has emerged as a new computing paradigm which enables the delivery of computing resources (e.g., servers, storages, software) over networks [9, 34]. NVE designers can easily access distributed computing resources in a pay-as-you-go manner through the Internet. In addition, computing resources can be quickly provisioned, accessed on-demand, and released. Thus, NVE designers can use the computing resources provided

by cloud computing with minimal management effort and at low cost.

NVEs are characterized by distributed real-time virtual world simulation and interactions among users. Consequently, to allow tens of millions of users to participate in a single NVE concurrently, large amounts of computing resources are needed. It is non-trivial to manage the computing resources to serve NVE users with high quality of experience, in a scalable and highly available way. Moreover, utilizing cloud resources to serve NVEs efficiently is challenging.

In this thesis, we address several challenges such as scalability, cost-efficiency and availability related to massivizing NVEs on clouds, with a focus on online games. Although we validate our main contributions using only online gaming data, we believe that the fundamental findings of our work also apply to other promising new areas of NVE-technology applications.

The rest of this chapter is structured as follows. Section 1.1 presents a general overview of NVEs and Clouds. Section 1.2 formulates the research questions we address. Section 1.3 summarizes the research contributions and presents an outline of this thesis.

1.1 Networked Virtual Environments and Clouds

Networked Virtual Environments (NVEs) [12] are virtual environments which are distributed across multiple physical hosts connected by networks. Figure 1.1 depicts an architecture of a cloud-based NVE, in which users connected to the servers hosted in clouds. The NVE creates a virtual world through which users can interact with others, and forms a social network for users. In this section, we give a brief overview of NVE classification in Section 1.1.1, of online games in Section 1.1.2, of clouds in Section 1.1.3, and of the requirements for cloud-based NVEs in Section 1.1.4.

1.1.1 NVE classification

An NVE aims to generate a virtual world through which distributed users can interact with each other. Based on the applications and characteristics of NVEs, we classify them into three types: virtual conferencing, tele-operation, and shared virtual environments. This classification is not exclusive, as some NVEs may belong to multiple types. For example, Second Life belongs to both the virtual conferencing and the shared virtual environment categories. Other classifications exist [12, 231]: for example, based on how data and computations are distributed, NVEs can be classified into client-server NVEs, peer-to-peer NVEs, and hybrid NVEs.

Virtual conferencing systems connect distributed users to attend virtual conferences, thus the users can interact with each other in real-time. Usually, these systems make

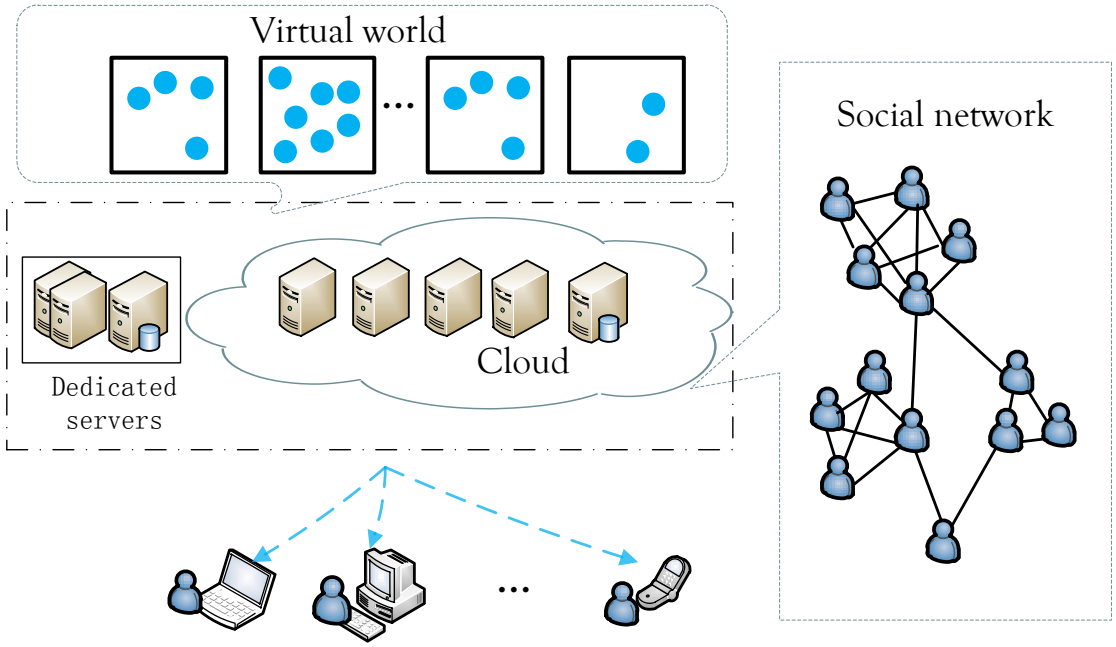


Figure 1.1: An architecture of a cloud-based NVE.

use of videos and audios, and map the user inputs onto some avatar(s). Unlike video-conferencing systems, virtual-conferencing systems focus on the quality of social interactions that are improved by adding more real-life elements such as gaze direction.

Tele-operation systems allow a user to experience a sense of tele-presence while working at a remote location. Typically a user is equipped with a remote device such as a mechanical robot which acts according to the user's actions. The remote device can provide the user with force feedback, which is important for environments that require precise control of objects such as in the case of tele-surgery.

Shared Virtual Environments (SVEs) allow multiple users to interact with each other and with virtual world objects. The most popular SVEs are online games. Users can cooperate to achieve certain goal in SVEs, e.g., painting or collaboratively editing shared documents; or users may not have any specific goal in virtual worlds, such as in Second Life.

1.1.2 Online Games

Online games, the most popular type of NVEs, are video games played online using the Internet or a local area network as a communication channel. Online games may feature interactions of globally distributed players in *virtual worlds*. Usually, the virtual worlds are created with a very high degree of immersion with 3D graphics, and are shared among players. The players can have real-time interactions with virtual world objects

(e.g., loots and non-playing-characters), and interact and socialize with other players, thus entertaining themselves and others.

Players interact with objects of the virtual world and with the other players through their in-game representation: *avatars*. Depending on the number of avatars each user can control, we classify online games into Single Avatar Virtual Environments (SAVEs) and Multi-Avatar Virtual Environments (MAVEs). For SAVEs, each user controls a single avatar to explore/interact with the virtual world and socialize with the other players. For MAVEs, each player can control multiple avatars simultaneously.

The most popular SAVEs are First Person Shooter (FPS) games [50] and Role-Playing Games (RPG) [50]. Most FPS games center on weapon-based combat through a first person perspective, such as Counter-Strike[1] where players, grouped in teams, compete in realistic battle scenarios. RPG center on role-playing: each player can choose different roles, and controls an avatar to act following different rules assigned to the roles. As an example of RPG, World of Warcraft (WoW)[2] mimics a fictional medieval-like and magic world in which the avatars of players can have different races, professions, and skills. Players control their avatars to explore the landscape, completing different in-game tasks, fighting monsters, and interacting with non-playing characters and the other players.

Real Time Strategy (RTS) games [32, 50] and Construction and Management (CM) games [97] are the most popular MAVEs. RTS games such as Blizzard's StarCraft[3] and Microsoft's Age of Empires[4] series are essentially Internet-based real-time world simulations in which players control avatars to gather resources, to construct buildings, to train combat avatars, to explore unknown territories, to trade, and to conquer. For CM games, such as the SimCity[5] series, each player acts as a manager to build and interact with the virtual world.

Besides experiencing a virtual world with others in-game, the players communicate with others off-game. To share in-game knowledge and ideas, and to socialize with others, social networks around online games are formed, such as Xfire[6] and Steam[7], in which like-minded players can discuss and share game-related media with the other players. Social networks based on online games are an important aspect of online games, which can boost their popularity and proliferation.

[1] www.counter-strike.net
[2] www.warcraft.com
[3] www.starcraft.com
[4] www.ageofempires3.com
[5] www.simcity.com
[6] www.xfire.com
[7] store.steampowered.com

1.1.3 Clouds

With the recent advance of virtualization and datacenter technologies, cloud computing has emerged as a new distributed computing paradigm where infrastructure, platform, and software are provided to users on a pay-as-you-go manner. Clouds offer users the illusion that computing resources are "infinite", and can be as easily accessed as electricity and water. In addition, cloud users can focus on their main business, instead of addressing infrastructure and other resource problems. Moreover, with the pay-as-you-go feature, cloud customers can reduce the risk of investment in infrastructure; if a software product does not succeed, the resources leased from a cloud for its operation and distribution can simply be released.

Leading industry vendors provide various cloud computing services. For example, Amazon offers Elastic Compute Cloud (EC2), Google provides App Engine, and Microsoft delivers Azure Cloud. In general, cloud providers offer their services in the following three models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). For IaaS, the cloud providers offer computing resources such as server, storage, and network to users. For example, users can lease virtual machines (VMs) from Amazon EC2, and deploy their own applications on top of these VMs. For PaaS, cloud providers deliver computing platforms such as programming language execution environments, databases, and web servers to users. For example, users can develop their own applications based on the computing platform without buying and managing the underlying computing platform using Google's App Engine. For SaaS, the cloud providers deliver software to users. The cloud providers manage infrastructures and platforms that run the applications. Google's gmail is an example of SaaS.

Clouds can be deployed as private, public, or hybrid. Private clouds are virtualized computing infrastructures that are self-owned and operated for a single organization only. Different from private clouds, public clouds offer services open to public use. In general, private clouds have higher capital investment but better security than public clouds. Hybrid clouds are the composition of public and private clouds: an organization can run its sensitive service inside its private clouds, and use public clouds to run its less-sensitive service.

Due to the elasticity, flexibility, availability, and pay-as-you-go features of cloud, cloud computing is gaining popularity as resource providers for NVEs. A few examples of cloud-based NVEs are Zynga's FarmVille[8] that uses both Amazon's EC2 and Zynga's private cloud, Cmune that builds the back-end platform of UberStrike on Amazon's cloud service [6], ROVIO that hosts its Angry Birds platform in Google's App Engine [88], and many MineCraft servers that are hosted on clouds [71].

[8] www.farmville.com

1.1.4 Requirements for Cloud-based NVEs

As a type of large-scale distributed applications, NVEs require such properties as scalability, consistency, reliability, heterogeneity, and security [160, 186]. The requirements this thesis focuses on are described in this section. Together, these requirements listed below represent the ability of massivizing NVEs, and in particular online games.

Scalability is defined as the ability of an NVE to host a large number of users simultaneously. As some NVEs, such as MMOGs, may involve millions of users, making the systems scalable is a key requirement. To achieve scalable NVEs, in-NVE scalability techniques should be developed to accommodate a large amount of users. Beside in-NVE scalability, efficient resource scheduling of computing resources is needed to make NVEs scale ”by credit card”.

Consistency Due to the geographical spread of NVE users, concurrent or even conflicting updates may be executed in different sites, resulting in inconsistent NVE states. These inconsistencies may dissatisfy users who are likely then to quit the game for an alternative, thus, in NVEs there should be some form of consistency control. Achieving strong consistency usually leads to large latencies which could reduce the responsiveness of NVEs [144]. For NVEs, especially for online gaming, as interactivity is of essence to the success of gaming, game designers usually trade off consistency for scalability [17, 144]. NVE designers need to determine which levels of consistency are needed, and they have to design appropriate consistency-ensuring protocols, mechanisms, and architectures.

Cost-efficiency refers to how efficiently monetary resources are used in an NVE system to serve its users. The NVE market is very competitive, and so it is necessary for NVEs to provide a high quality of experience to their users. However, NVEs are fashion-driven, with the workloads of NVEs fluctuating over time. It is not wise to spend large amounts on purchasing self-owned computing resources based on the peak workloads of NVEs, because the self-owned computing resources are hardly ever fully utilized. Cloud computing resources can be used to reduce the operational cost of hosting NVEs.

Availability Like many other Internet services, high availability is a desired property for NVEs. Large-scale NVEs adopt large amounts of computing resources to serve their users. Due to the sheer scale, resource failures are bound to happen. NVEs should be designed to tolerate failures to achieve high availability.

Responsiveness is defined as the ability for the NVE systems to provide response to the inputs of users in real-time. The speed with which a user can witness the interactions with other users in a virtual environment is important. If an NVE is not responsive, the users will not be satisfied and may quit the NVE. The responsiveness requirements vary between different NVEs. For FPS, the time it takes between the input from a user and the associated effect should be lower than 100 ms [50]. For RTS, the responsiveness can be

Table 1.1: The relationships among the requirements for cloud-based NVEs, the research questions addressed in this thesis, and the chapters answering research questions. "Y" indicates that the answer to the research question addresses a requirement.

	Chapter(s)	Scalability	Consistency	Cost-efficiency	Availability	Responsiveness
Q-1	2	Y	Y	Y	Y	Y
Q-2	3 & 4	Y		Y	Y	
Q-3	5	Y		Y		
Q-4	6	Y	Y			Y
Q-5	7			Y		
Q-6	8			Y	Y	

within the order of 200 to 1000 ms [50].

Other requirements that are important to NVEs exist, but are not the focus of this thesis. Persistence is required to give NVE users a sense of familiarity. As a successful NVE could operate for years, users expect their personal profiles, their virtual items, as well as the places they visited to persist. Usually, persistence of NVEs is treated as a subdomain of NVE consistency, in that an NVE's persistence model determines when and how data are stored [83]. Any system that is not designed to withstand malicious behavior is likely to crash or fail easily, and NVEs are no exception to this empirical rule.

1.2 Problem Statement

To massivize NVEs using cloud computing, the workload characteristics of NVEs should be well understood. Further, based on the observations of NVE characteristics, we design several mechanisms to achieve scalable, cost-efficient, highly-available, cloud-based NVEs. In this thesis, we will address the following research questions. The relationships among these research questions, the requirements for cloud-based NVEs, and the chapters of this thesis are depicted in Table 1.1.

Q-1 How to benchmark NVEs? NVEs have hundreds of millions of users worldwide. This growing population expects new NVE designs and more scalable NVEs every year. However, few benchmarking tools and environments exist for NVE designers and practitioners; of these, even fewer are available to NVE researchers and communities. A benchmarking environment is needed to test the performance of NVEs automatically and reliably. Moreover, the benchmarking environment should be able to configure various aspects of NVEs to help the NVE designers gain in-depth knowledge of NVEs.

Q-2 What are the workload characteristics of the online social networks around NVEs? Social networks can be an important aspect of an NVE, as it allows its users to maintain friendships, to discover new friends, and to share their knowledge of the NVE. NVEs may also lead to relationships that normally are not expressed or even do not exist in regular online social networks, for example, winning together and competing with each

other. Moreover, online social networks may be formed around several NVEs, in which users can socialize with each other on topics across multiple NVEs. Understanding the social network structure of NVEs can help to create innovative services which benefit both NVE operators and users.

Q-3 What are the characteristics of user mobility in NVEs? Much research effort has been spent on scaling NVEs, to manage computing resources, and to reduce communication overhead. The effectiveness of those approaches is affected heavily by the movements, behaviors, and interactions of players. The mobility of the avatars of players can have an important impact on the performance of NVEs such as load-balancing [60]. Despite recent efforts on the workload characterization of NVEs, the characteristics of avatar mobility in NVEs are still not well understood. In-depth understanding of the mobility patterns of users in NVEs can help the NVE designers to develop scalable and cost efficient NVEs.

Q-4 What mechanisms and architectures can make NVEs scalable? Millions of users may participate in one NVE simultaneously. To serve the massive amount of users, various NVE techniques such as area of interest and zoning have been proposed. These techniques have been proved to be efficient for single-avatar NVEs (SAVEs) to host tens of thousands of users in one NVE instance. However, different from SAVEs, the design of multi-avatar NVEs (MAVEs) does not scale using traditional techniques. At most tens of users can join one MAVE instance simultaneously. The traditional techniques for MAVEs are event-based lock-step simulation (EBLS), which ensures high consistency, but low scalability and responsiveness. Developing an MAVE with high scalability, low responsiveness, and good consistency is challenging.

Q-5 What mechanisms can make cloud resource use in NVEs cost efficient? To serve millions of users world-wide, NVE operators need to use massive amounts of machines distributed globally. Although cloud computing techniques promise to reduce the operational cost of NVEs, how to manage the cloud computing resources to maximize the profit of NVE operators yet deliver the QoS required by players is still daunting for NVE operators. Resource provisioning and allocation algorithms that are cost efficient and fast are needed.

Q-6: How to provide a highly available yet cost-efficient NVE service using cloud resources? The NVE operators lease large amounts of cloud computing resources from datacenters to serve globally distributed users. Due to the scale of today's datacenters, failures of computing resources are bound to happen. The failure of cloud computing resources may disrupt the availability of NVEs, leading to user dissatisfaction and revenue loss. Managing the computing resources efficiently to provide a highly available NVE service is needed.

1.3 Contributions and Thesis Outline

The contributions of this thesis are listed below. All the research questions (Q-1 to Q-6) are addressed, one question per chapter, except for Q-2, which is addressed in both Chapter 3 and 4. The relationships between research questions and chapters are described in Table 1.1.

Benchmarking NVEs (Chapter 2) To address research question Q-1 about how to benchmark NVEs, we introduce RTSenv, a benchmarking system for NVEs with a focus on RTS games. RTSenv can configure and manage the main aspects of RTS games, such as maps, computer-controlled units, and game scenarios. RTSenv leverages multi-cluster systems and reactive fault tolerance mechanisms to perform robust, multi-machine, and multi-instance game experiments. Using our reference implementation of RTSenv in DAS-4 (a Dutch multi-cluster system) and Amazon EC2, we show that RTSenv can be used in a variety of scenarios. Our results give evidence that several common assumptions made by researchers about game workloads do not hold in general for RTS games and thus warrant a more detailed investigation. The content of this chapter is based on the following work:

- **Siqi Shen**, Otto Visser, and Alexandru Iosup, "RTSenv: An Experimental Environment for Real-Time Strategy Games," *Annual Workshop on Network and Systems Support for Games (NetGames)*, 2011.

Analyzing Implicit Social Networks in NVEs (Chapter 3) To address research question Q-2 about the social networks of NVEs, we propose a formalism consisting of various ways to map the interactions of players to social structures, and apply this to real-world data collected from three different game genres. We analyse the implications of these mappings for in-game and gaming-related services, ranging from network and socially aware matchmaking of players, to an investigation of social network robustness against player departure. The author of this thesis has closely collaborated for this chapter with Ruud van de Bovenkamp of the Network Architectures and Services group of TU Delft. He has made a major contribution to the data processing and analysis, and to the development of the model in this chapter. The content of this chapter is based on the following work:

- Alexandru Iosup, Ruud van de Bovenkamp, **Siqi Shen**, Adele Lu Jia, and Fernando Kuipers, "An Analysis of Implicit Social Networks in Multiplayer Online Games," *IEEE Internet Computing* 18(3), pp. 36-44, 2014.
- Ruud van de Bovenkamp, **Siqi Shen**, Alexandru Iosup, and Fernando Kuipers, "Understanding and Recommending Play Relationships in Online Social Gaming," *International Conference on COMmunication Systems and NETworkS (COMSNETS)*, 2013.

Analyzing Online Meta-Gaming Networks (Chapter 4) To address research question Q-2 about the social networks of NVEs, we analyze a long-term observation of XFire. Using long-term, large-scale data that we have collected, we present a high-level, marginal distribution- and time-based analysis of XFire: its global network, player activity, user-generated content, and social structure. Ours is one of the first characterizations of a MetaGaming network, with prior work [21] developed concurrently and independently from our own work. We find that XFire is a slowly growing network whose players spend collectively in-game over 100 years, every hour. We quantify the "hardcore"-ness of XFire players, and find that a significant fraction of them have played over 10,000 in-game hours each. We also find that XFire community members are routinely engaged in the creation and consumption of game-related media, such as screenshots and videos. The content of this chapter is based on the following work:

- **Siqi Shen** and Alexandru Iosup, "The XFire Online Meta-Gaming Network: Observation and High-Level Analysis," *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2011.

Analyzing and Modeling in-NVE Mobility (Chapter 5) To address Q-3 about understanding in-NVE mobility patterns, we collect from World of Warcraft (WoW) mobility traces for over 30,000 virtual citizens, and compare these traces with traces collected from Second Life (SL) where the environment is designed and changed significantly by the citizens themselves. Furthermore, motivated by the existence of numerous studies and models of real-world mobility, we systematically compare the characteristics of two NVE and two real-world mobility traces. Our comparative study reveals that long-tail distributions characterize well various mobility characteristics such as pause-duration and area-popularity, and that area-visitation shows personal preferences. We also find several differences between NVE and real-world mobility characteristics.

Based on the observation of mobility in NVEs, we propose SAMOVAR, a Statistical Area-based MObility model for VirtuAl enviRonments. SAMOVAR models four mobility characteristics: pause duration, velocity, area popularity, and distinct visited areas using empirical distributions. It then uses a map generation and a traveling procedure to generate movement trajectories of avatars. We show through simulation that the traces generated by our model can produce many mobility characteristics observed in the virtual world. Moveover, the simulation results obtained from SAMOVAR on a client/server architecture are similar to traces from WoW and SL. The content of this chapter is based on the following work:

- **Siqi Shen**, Niels Brouwers, Alexandru Iosup, and Dick Epema, "Characterization of Human Mobility in Networked Virtual Environments," *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2014.

- **Siqi Shen** and Alexandru Iosup, "Modeling Avatar Mobility of Networked Virtual Environments," *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2014.

Scaling NVEs through the Area-of-Simulation Mechanism and Architecture (Chapter 6) To address Q-4 about scaling NVEs, through empirical analysis, we show that a single Area of Interest (AoI), which is a scalability mechanism that is sufficient for SAVEs (such as Role-Playing Games), cannot meet the scalability demands of MAVEs. To enable scalable MAVEs, we propose Area of Simulation (AoS), a new scalability mechanism, which combines and extends the mechanisms of AoI and Event-Based Lockstep Simulation (EBLS). Unlike traditional AoI approaches, which employ only update-based operational models, our AoS mechanism uses both event-based and update-based operational models to manage not single, but *multiple* areas of interest. Unlike EBLS, which is traditionally used to synchronize the entire virtual world, our AoS mechanism synchronizes only selected areas of the virtual world. We further design an AoS-based architecture, which is able to use both our AoS and traditional AoI mechanisms simultaneously, dynamically trading-off consistency guarantees for scalability. We implement and deploy this architecture, and we demonstrate that it can operate with an order of magnitude more avatars and a larger virtual world without exceeding the resource capacity of players' computers. The content of this chapter is based on the following work:

- **Siqi Shen**, Shun-Yun Hu, Alexandru Iosup, and Dick Epema, "The Area of Simulation Mechanism and Architecture for Multi-Avatar Virtual Environments," *ACM Transactions on Multimedia Computing, Communications and Applications*, under minor revision.

Scaling NVEs Efficiently through Cloud Scheduling (Chapter 7) To address Q-5 about cost-efficient NVEs, we investigate leasing strategies and their policies from a brokers perspective. We propose *CoH*, a Cloud-based, online, Hybrid scheduling policy that minimizes rental cost by making use of both on-demand and reserved instances. We formulate the resource provisioning and job allocation policies as integer programming problems. As CoH needs to be executed online, we limit the time to explore the optimal solution of the integer program, and compare the obtained solution with various heuristics-based policies; then we pick the best one. We show, via simulation and using multiple real-world traces, that the hybrid leasing policy can obtain significantly lower cost than typical heuristics-based policies. The content of this chapter is based on the following work:

- **Siqi Shen**, Kefeng Deng, Alexandru Iosup, and Dick Epema, "Scheduling Jobs in the Cloud Using On-demand and Reserved Instances," *Euro-Par*, 2013.

Making NVEs Robust through the Availability On-Demand Mechanism (Chapter 8) To address Q-6 about highly available and cost-efficient NVEs, we propose Availability-on-Demand (AoD), a mechanism consisting of an API that allows datacenter users to specify availability requirements which can dynamically change, and an availability-aware scheduler that dynamically manages computing resources based on user-specified requirements. The API considers two levels of availability, normal and high, and allows datacenter users to specify dynamic requirements through as little as one API call. The scheduler considers user-specified availability requirements, and dynamically provides high availability (HA) for applications. The mechanism operates at the level of individual service instances, thus enabling fine-grained control of availability, for example during sudden requirement changes and periodic operations. Through realistic, trace-based simulations, we show that the AoD mechanism can achieve high availability at low cost. The AoD approach can consume about the same number of CPU hours but with higher availability than approaches which use HA techniques randomly. Moreover, comparing to an ideal approach which has perfect predictions about failure, the AoD approach consumes 13% to 31% more CPU hours but achieves similar availability for critical parts of applications. The content of this chapter is based on the following work:

- **Siqi Shen**, Alexandru Iosup, Assaf Israel, Walfredo Cirne, Danny Raz, and Dick Epema, "An Availability-on-Demand Mechanism for Datacenters," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015, to appear.

In Chapter 9, we summarize the conclusions of this thesis and we provide suggestions for future research directions.

Chapter 2

Benchmarking NVEs

Among the different NVEs, Real Time Strategy (RTS) games such as StarCraft II (one of the best-selling games of 2010 [22]) are played by millions of players daily. To address the increasing requirements of the gamers and the increasing competitiveness of the market, many new NVE technologies [19, 99] have appeared in the past five years, and a new generation of massively multiplayer RTS games, such as Planetary Annihilation [105], is currently under development. Although the development of a new RTS game can take multiple years to develop [146], and the abundance of challenges in the design, implementation, and testing of RTS games makes experimental tools valuable, few experimental tools are available for RTS game research and development. To address this situation, in this chapter we introduce RTSenv, a benchmarking system for NVEs with a focus on RTS games.

Even the casual players of online RTS games incur a specific near-real-time constraint that limits the acceptable response times for issued commands to 200-300 milliseconds [193]. When this constraint is not met, players have poor gameplay experience, and may quit in favor of other games. The near-real-time constraint is difficult to meet continuously for most consumer-grade computers, even today. As a consequence, online RTS games rely on complex technology that balances the computational, memory, and bandwidth components of the workload on the different system components and players.

The limited availability of experimental tools makes it difficult to generalize or even understand the results of previous studies. It is perhaps symptomatic for the state of this research field that a recent study [150] argues against the practical viability of P2P-based games as a conclusion of real-world experiments, thus contradicting several previous simulation-based studies that indicate otherwise [19,99]. Moreover, while some large gaming companies perform extensive game studies, including comprehensive user studies [120,171], few others can afford building the tools necessary for such approaches; even the few that do, have little expertise with large-scale experimental environments [146]. Despite recent interest in experimental tools [31, 132, 218], there currently exists no pub-

lic experimental RTS system. Although simulators may be appropriate for studying RTS games, they represent simplifications of the real systems and games; when their simplifying assumptions do not hold, their results fail to describe reality. Although many tools already exist for testing distributed systems, RTS games have additional, idiosyncratic, performance-affecting configuration parameters, such as the map size and the number of units.

In this chapter we introduce RTSenv, which is designed for experimental RTS game studies. RTSenv can be used to evaluate the performance of RTS games under a variety of game configurations and scenarios, from traditional performance evaluation to game design. Besides traditional system performance metrics such as CPU, memory, and network consumption, RTSenv can assess RTS-specific operational metrics such as player scores, and the number of active and profitable units. RTSenv can operate on a variety of physical platforms, from multi-cluster wide-area environments such as DAS-4 [10], to cloud computing machines and even single, multi-core desktop computers. Our main contributions are:

1. We analyze the requirements of benchmarking systems for RTS games (Section 2.1);

2. We design and implement RTSenv, a benchmarking system for RTS games (Section 2.2);

3. We show through experiments with a popular RTS game in real multi-cluster and cloud infrastructures how RTSenv supports various scenarios (Section 2.3).

2.1 Background

2.1.1 Use Case: `OpenTTD`, a Real-Time Strategy Game

As main use case for this chapter we focus on `OpenTTD` [167], which is an open-source, popular RTS game. `OpenTTD` has been developed since 2004 by a community of volunteer game developers lead by Remko Bijker as an extension to the commercial game Transport Tycoon Deluxe by Chris Sawyer (MicroProse, 1994). `OpenTTD` is a business simulation game with a wide appeal: it has been downloaded more than one million times and the game is used as textbook companion for college-level business courses [102].

`OpenTTD` has the features of commercial RTS games. The game world, which may be randomly generated or selected from the many community-created maps, emulates the real world through a combination of realistic geography, economy, and demographics. The player has complete control over a transport company: buying transport vehicles

such as buses and trains; building transportation paths such as roads and train tracks; planning and managing the operation of the company; etc. Rival companies compete against each other to achieve the highest profit by transporting passengers and goods. OpenTTD game sessions (games) can be played against human and/or Artificial Intelligence (AI)-controlled players.

OpenTTD follows the typical program structure of an online RTS game, in which the game world is maintained on a server, and each player connects and interacts with the game world through a client application running on the computer/device of the player. One of the players often runs the server alongside a client; alternatively, the server is placed on a "neutral" computer for ranked games and competitions. The game server repeatedly executes a main game loop comprised of the sequence "get and process player input", "update the global game world", and "send (small) updates about the game world to each player"; based on the latter step, each client can reconstruct the state of the world, effectively performing the same updates of the global game world as the server.

OpenTTD vs. other RTS games Although a variety of RTS games exist, the AAA game market has focused since 2005 on games such as the *Age of Empires* and the *Starcraft* series, each offering the player large worlds, the possibility to control large numbers (tens or hundreds) of vehicles, and many options to build real-world-like cities. With the exception of in-game combat, OpenTTD is similar to these games. OpenTTD also employs a non-violent alternative to traditional combat, in that players may use advanced tactical and micro-management skills to restrict the free movement of the vehicles of other players. OpenTTD can also offer sufficient challenge to players of various abilities and experience: from the over 30 AIs created and made public by the game's community, several can give a good challenge to the good human player, while others can win comfortably even against the expert human player. OpenTTD is highly configurable. Before starting the game, the players can specify the map structure, the maximum number of vehicles, the inflation rate of the economy, etc. Players can select the amount of water (via the sea level), the "hilliness" of the map, and the density of economic resources (via the industry and town density). This level of configuration detail, which is common for RTS games [182], makes modeling such games a difficult and recurrent process.

2.1.2 Requirements

Starting from the use case, we synthesize three main requirements for a benchmarking system for RTS games:

1. *Control experiments*: the system must have the ability to control experiments (start, stop, monitor, use results) without input from a human experiment manager. The test infrastructure may be any of desktop machine of the programmer/researcher, local studio cluster, multi-cluster grid of computers shared by multiple studios, or

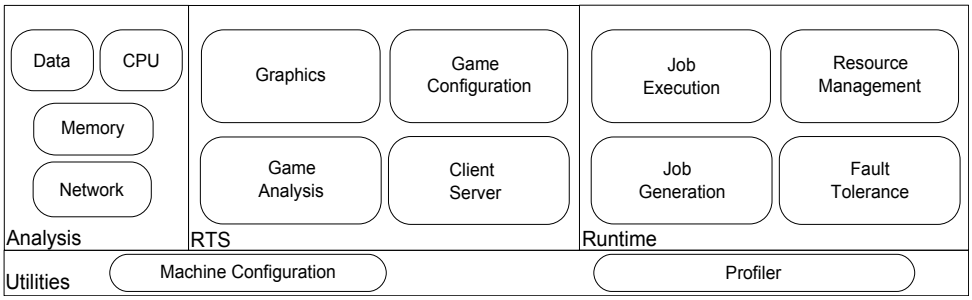


Figure 2.1: The `RTSenv` architecture.

cloud computing infrastructure. The system must be able to perform experiments when real users are present, but also when no real users are present; for the latter, the system must support both the use of traces/replays and the use of AI-controlled players. Last, the system must support RTS-specific configuration parameters, such as map structure, unit count, number of players (including AIs), etc.

2. *Evaluate and model RTS operation*: the system must support traditional system performance metrics; in particular, it should be able to report metrics concerning resource consumption (CPU, network, disk, memory) over time.

   Besides reacting to network conditions, in particular latency and throughput, players may respond to other sources of performance degradation, such as the server's processing (CPU) speed. The system must also support RTS-specific operation abstractions and metrics, such as player scores, and the number of active and profitable units.

3. *Compare game design choices*: the system must have support for comparing the results obtained from different scenarios as support for game design decisions. For example, it is common for game designers to package AI players with commercial games; however, selecting a specific AI type or configuration for a specific (random) map should be based on an in-depth analysis of the AI performance/resource consumption trade-off.

We show in Section 2.3 that a system satisfying these three requirements can be used in a variety of experimental scenarios, from performance evaluation to game design. We conjecture that such a system can be used for unit testing and debugging (both important parts of game development), and for systems research, for example for comparing algorithms, methods, and techniques.

2.2 The RTSenv Benchmarking System

In this section we present the RTSenv benchmarking system in-turn, the architecture, the RTS-specific features, and the implementation and adaptation details.

2.2.1 Architectural Overview

RTSenv consists of four modules (see Figure 2.1), Utilities, Analysis, RTS, and Runtime. The Utilities and Analysis modules together automate the environment- and game-specific configurations, and collect and analyze the environment- and game-specific results of the experiments. The RTS module is used to set up and monitor the gaming environment. The *Runtime* module is responsible for provisioning machines for the single node/cluster/multi-cluster grid/cloud computing environment used by RTSenv. This module is also responsible for organizing, managing, and executing the experimental process, which is based on jobs. Part of the *Runtime* module, the *Resource Management* component is responsible for organizing the execution of experiments; for example, to manage the execution of experimental jobs in multi-cluster environments, it operates an FCFS queue that interfaces with the resource manager of the system. This component is also responsible for acquiring and releasing resources, and for invoking the Job Execution component on each allocated machine. The *Job Generation* component parses the experiment description, then generates configuration files and experiment jobs. Jobs can be single-machine clients, single-machine servers, or multiple-machine instances comprising one server and multiple clients. The *Fault Tolerance* component uses a reactive, retry-based fault tolerance mechanism in which failed jobs are re-submitted for execution until they succeed or the number of failures exceeds a user-defined threshold; failures are detected at the end of after the job execution by checking the output.

Experiment Control RTSenv allows users to run single and multi-machine experiments using a variety of infrastructures, among them, desktop computers, clusters, multi-clusters, and clouds. For confidence in experiment results, users may specify the number of repetitions for single and multiple-instance experiments. RTSenv also allows its users to specify a maximum experiment runtime; it will stop overdue or user-selected experiments. RTSenv assumes that the environment in which it runs achieves synchronization between compute nodes; for example, time synchronization through the use of the NTP protocol, machine start synchronization through multi-machine allocation and co-allocation, etc.

2.2.2 RTS-Specific Features

Many RTS games, such as StarCraft II, provide a in-game recording mechanism (*replay*), which saves all the game commands issued by players. If the game world simulation is

deterministic, replays can be used to reproduce the recorded game. Replays are commonly used by game operators to debug games (*replay-based testing*), and by players to share their gaming achievements and to learn from more skilled players. RTSenv supports *replay-based testing*.

RTSenv already supports many of the abstractions present in modern RTS games. First, RTSenv can control the virtual world's geography by changing the map size and structure (such as density of water or hills). Second, RTSenv can control the level of challenge proposed by the virtual world through parameters such as the number of resources present on the map, the amount of starting resources for each player, the types of resource collectors, etc. Third, RTSenv can control the maximum allowed player presence, for example by limiting the number of units each player can control. Fourth, our tool can control the in-game duration of a game session, for example one year. Besides traditional performance metrics, RTSenv can measure various RTS-specific metrics, such as the player scores, the number of (profitable) units, etc.

2.2.3 Qualitative Analysis of the RTSenv Design

Meeting the requirements RTSenv meets the three requirements formulated in Section 2.1.2. For Requirement 1, RTSenv combines experiment control (described earlier in this section) with replay-based testing abilities. For Requirement 2, RTSenv can evaluate and create simple statistical models of traditional performance evaluation and RTS-specific metrics. For Requirement 3, RTSenv can be used to test and compare the performance of different AIs under various game settings (map structure, etc.)

Fit with the industry game development process Due to the high costs and risks associated with game development processes, the industry has started to mature, and today most games are developed through the same *basic game development process* [76, Ch.2] [146], which includes steps such as concept discovery, prototyping, pre-production, full production, quality assurance (QA), finaling, pre-launch demos, the launch, and post-launch support. We find that RTSenv are particularly useful for the critical production, QA, pre-launch, and post-launch support stages (see [201]).

2.2.4 Implementation

Our reference implementation of RTSenv, which is coded in Python and uses `wireshark` to record the network traffic, is portable and extensible. We have tested our implementation on various Windows, Ubuntu, Centos and Fedora systems; on single desktops, on single and multiple clusters from the DAS-4 [10] multi-cluster/grid computing environment, and on the Amazon EC2 cloud computing environment.

Our implementation is extensible in the sense that a user can add more experiments,

more performance metrics, and more statistical tools by simply adding Python components to the source directory. Adding in RTSenv support for `OpenTTD` required from us several modifications to the game packaging, but no changes to the game design or core implementation. First, we have modified the visualization of `OpenTTD` and add a non-graphical client mode into `OpenTTD`; this is needed for experimenting in cluster environments. Second, we have extended the original in-game limitation to the number of companies (one player per company) from under 16 to 250. Third, we have changed the operation of the `OpenTTD`'s AI module to allow AI-controlled players to operate remotely from the game server; this effectively enables multi-machine game sessions and various scalability tests. Fourth, we have made use of the original debug utilities of `OpenTTD` to implement the replay functionality. Fifth, we have used the load map function of `OpenTTD` to verify the correctness of completed games and to obtain game-specific scores and data.

2.3 Experimental Results

In this section we present the experiments we have performed using a reference implementation of RTSenv. Our experiments show evidence that RTSenv can be used in a variety of experimental scenarios and on a variety of computational platforms. Although made possible by RTSenv, a comprehensive evaluation of `OpenTTD` or building a performance model for RTS games are both outside the scope of this chapter. Overall, we have conducted using RTSenv over 20,000 game sessions, which amount to over 30,000 in-game years and over 7,000 real operational hours. For more results and detailed analysis we refer to our technical report [201].

2.3.1 Experimental Setup

Experiment configuration We configure RTSenv to record the performance of CPU and memory load twice per second. Except for the experiments done for measuring network traffic, each experiment is repeated at least 30 times.

Experimental Environment Unless otherwise noted, we ran our multi-machine experiments using the DAS-4 [10] six-cluster, wide-area system. A typical compute node of DAS-4 has a dual quad-core 2.4GHz Intel E5620 CPU and 24GB memory; the intra-cluster network is 1GBit Ethernet. To avoid interference effects, when possible each game instance is allocated a single node of DAS-4; for example, when experiments do not focus on the network performance of the game.

Game Configuration All the experiments use Artificial Intelligence (AI)-controlled players; the AI algorithms (the AIs) are real, high quality, open-source, and community-provided. Unless otherwise specified, the game sessions are configured as in typical

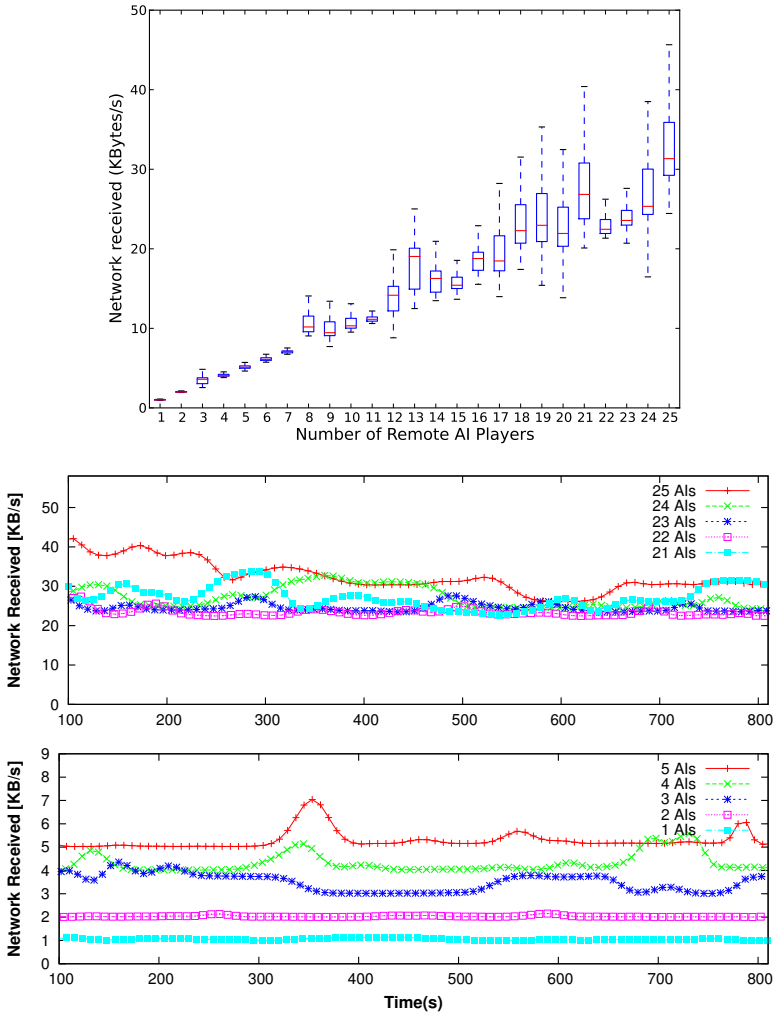


Figure 2.2: Download bandwidth of the `OpenTTD` server for various remote player counts: (*top*) basic statistics, depicted as box-and-whiskers plots; (*middle and bottom*) consumption over time for various number of AIs.

`OpenTTD` AI competitions [168]: default configuration of `OpenTTD`, except for the starting in-game time (1998) and initial capital (high). Each game instance is scheduled to run one in-game year by default, which leads to a *nominal execution time* of about 825 seconds on a typical DAS-4 node. Longer execution times, which result from server overloads, are noticeable to players only if they exceed 105% of the nominal execution times (the *playable execution time range*).

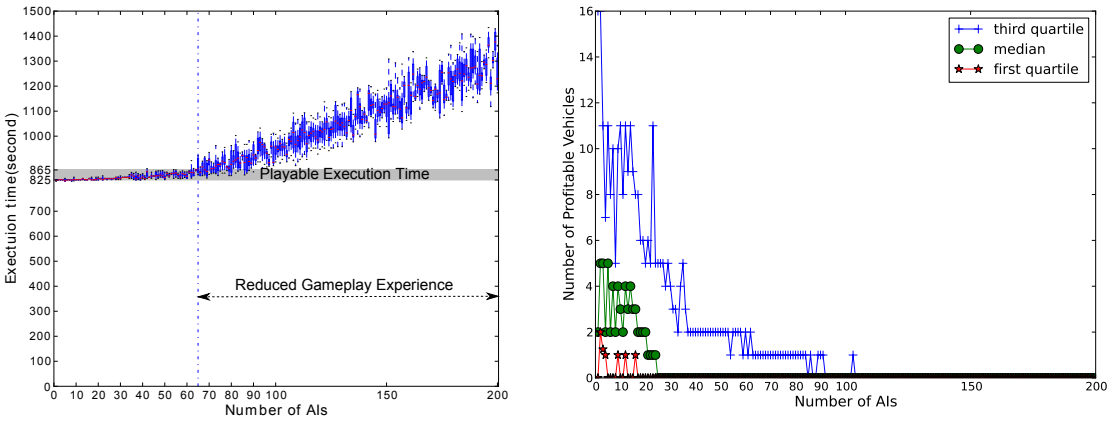


Figure 2.3: System and in-game performance, respectively, for the `OpenTTD` server for various player counts. (*left*) game execution time; (*right*) number of profitable vehicles.

2.3.2 Performance Evaluation Results

Network Measurements, System (Figure 2.2): We conduct multi-player game experiments to assess network consumption. Each AI player connects to the server remotely, first to download the game map, then to issue commands and to receive small updates (for example, each command issued by the each other player)—this scenario emulates real gamers competing through the services of a commercial game server. The server is executed on desktop computer while the clients are executed on cluster's computer node. Figure 2.2 shows the latter type of network traffic, as observed for this scenario when the number of remote players is varied from 1 to 25. The statistical properties of the bandwidth consumption for server download (Figure 2.2 (top)) indicate that the median download traffic size increases nearly linearly to the number of players (the *linear model* commonly assumed for games [191, 228]), however, wide value ranges of the download traffic size indicates that a linear model may be inappropriate. Players with limited bandwidth may find that the real bandwidth consumption exceed significantly than the linear model's prediction. We now show that the wide value ranges also have long duration. The traffic over time in Figure 2.2 (bottom)–the download bandwidth consumption is stable when the number of players(AIs) is low (up to 5 players), but increases and becomes more variable as the number of players increases. Noticeably, minute-long periods of much higher traffic than expected from the linear model occur even for 5 players (see corresponding curve "5 AIs" between 300 and 400 seconds). Periods of such length will be noticed even by beginning players.

Scalability, System Performance and User Experience (Figure 2.3 (left) and (right), respectively): We assess the scalability of the game server by increasing the number of players from 1 to 200, that is, to an order of magnitude more players than in today's commercial games. All the AIs are running on the game server, thus consuming CPU

Table 2.1: Characteristics of the cloud resource (instance) types. The ECU is the CPU performance unit defined by Amazon.

Name	Cores (ECUs)	RAM [GB]	Archi. [bit]	Disk [GB]	Cost [$/h]
m1.small	1 (1)	1.7	32	160	0.1
m1.large	2 (4)	7.5	64	850	0.4
c1.xlarge	8 (20)	7.0	64	1,690	0.8

and memory resources—this is the typical mode of operation for current commercial games [160, 191]. The results are depicted in Figure 2.3 (left). When the number of players is below 30, the game execution time is stable and very close to the nominal execution time (see Section 2.3.1). The playable execution time range is exceeded at about 60 players; afterwards, the game execution time increases quickly in both median value and range, which results in reduced gameplay experience. The game execution time growth is mainly due to CPU consumption; as shown in our technical report [201, Sec.4.2], the maximum memory consumption is below 700MB (`OpenTTD` is not memory-bound). We further analyze the results of the previous experiment from a gameplay experience perspective. Since in our tests we cannot ensure the presence and follow a group of human testers that adhere to the standards of gameplay experience evaluation [120, 171], we use instead a proxy metric for gameplay experience, the median number of profitable vehicles. For scalable gameplay experience, the median number of (profitable) vehicles should be scalable, that is, it should at least not decrease when the number of players increases. When the number of vehicles is not scalable, that is, the number of profitable vehicles is low regardless of the player ability, the players experience a reduced feeling of mastery. A comprehensive user study focusing on the relationship between the number of vehicles and the players' enjoyment falls outside the scope of this chapter. Figure 2.3 (right) shows the statistical properties of the number of profitable vehicles as a function of the number of competing players. As the number of players increases, it becomes increasingly difficult for players to have profitable vehicles. Over 25 players, the median value of the number of profitable vehicles is 0 and the maximum value drops quickly below 5. The observed gameplay scalability limit (25 players) is below the system scalability limit (60 players for our platform) and does not depend on the platform; thus, *Main Finding: for (RTS) games, system scalability needs to be analyzed and improved in conjunction with gameplay experience scalability.*

2.3.3 Comparing Deployment Choices

Cloud resource performance Resources of various configurations can be now leased from commercial cloud providers, such as Amazon, and used to deploy on them game servers. Cloud resources have pre-agreed leasing costs and operational/performance guar-

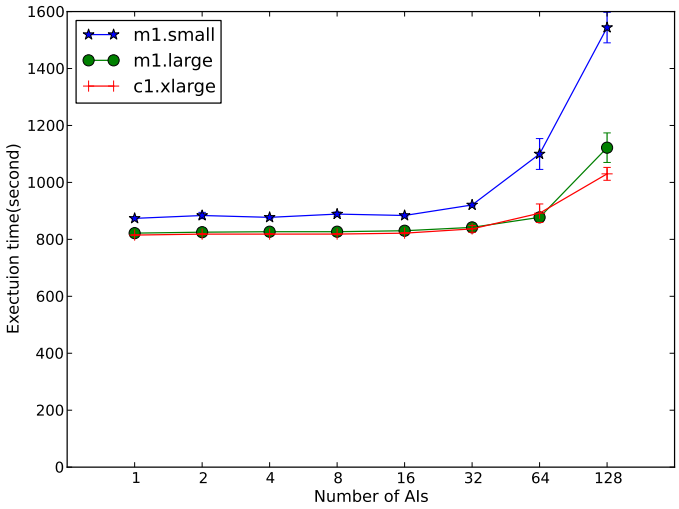


Figure 2.4: Execution time for identically-configured game instances on various cloud resources.

antees, expressed in public Service Level Agreements. Through virtualization, that is, a technology for emulating various computational machines on the same physical machine with little performance loss, a game server can be run without re-implementation or even re-compilation on a machine leased from a cloud. However, a game operator would still have to decide, from the many offers available, which suits best the game to be deployed. To emulate this situation, we ran performance experiments on various types of computing nodes leased from the Amazon EC2 commercial cloud, with the characteristics summarized in Table 2.1. Figure 2.4 depicts the performance (execution time) of identically-configured game executed when run on three different cloud resources. The use of the most expensive resource in this test, `c1.xlarge`, has an impact on performance only for a large number of AIs (close to 128 and above). With this information, the game operator can implement various deployment strategies, from "lease the best resources" to "optimize the game performance/operational cost trade-off". *Main Finding: RTSenv enables the selection of resources for deployment.*

2.4 Related Work

We contrast RTSenv with previous experimental RTS/game studies, experimental RTS/game platforms, and RTS/game simulators, and others in our technical report [201]. In comparison with this body of related work, RTSenv has different scope (system and game-specific experiments), focus (RTS games), and application (a *real* game, `OpenTTD`, and many high quality artificial players.)

Few *experimental RTS game studies* exist. Closest to the experimental part of our work, a study on the prototype RTS game Rokkatan [157] tests scalability with about 400 clients, but the clients are controlled by prototype instead of production, that is, high-quality and CPU-intensive, AIs.

Experimental game platforms exist [119, 132, 218], but they do not have RTS-specific support and the studied resource is mainly the network.

Simulation-based studies of online games exist. Simulators developed using standardized, industrial-grade simulation platforms such as HLA and the older DIS and FIPA, have been used to investigate online game-like scenarios [155]. A simulator was used [159] to study how data centers and cloud computing can support online gaming. A variety of peer-to-peer online game simulators exist [19, 187, 228].

2.5 Summary

The increasing popularity of RTS games fosters demand for new designs and technical solutions, which emphasizes the need for experimental environments. In this chapter, we have introduced RTSenv, a benchmarking system for RTS games that is useful for both researchers and developers. Our system can control and measure many RTS-specific aspects, and enables a variety of experimental scenarios–from performance evaluations to taking game design decisions. RTSenv can operate in several types of computing environments, from single desktop computers to wide-area multi-clusters and commercial clouds, and leverages reactive fault tolerance techniques to perform robust, multi-machine, multi-instance RTS game experiments. We have used RTSenv in DAS-4, a multi-cluster environment, and Amazon EC2, a commercial cloud provider, to conduct an extensive study of the popular RTS game `OpenTTD`.

The experimental results show that RTSenv can help evaluate many RTS-specific features and can help comparing different game design choices. They also show that RTSenv can lead to new findings, such as: the scalability of an RTS game should be evaluated not only from the performance but also from the gameplay experience perspective.

Chapter 3

Analyzing Implicit Social Networks in NVEs

Online games are games that use advances in networking and a variety of socio-technical elements to entertain hundreds of millions of people world-wide. Unsurprisingly, such games naturally evolve into Online Social Games (OSGs): the many people involved organize, often spontaneously and without the help of in-game services, into gaming *communities*. While typical online social networks revolve around friendship relations, new classes of prosocial emotions appear in OSGs. For instance, adversaries motivate each other and together may remain long-term customers in an OSG. Adversarial relationships are one of the implicitly formed in-game relationships we study. Understanding in-game communities and social relationships could help improve existing gaming services such as team formation, planning and scheduling of networking resources, and even retaining the game population.

Few games exhibit a greater need for socially-aware services than the relatively new genre of multiplayer online battle arenas (MOBAs) considered in Section 3.1. Derived from Real-Time Strategy (RTS) games, MOBAs are a class of advanced networked games in which equally-sized teams confront each other on a map. In-game *team-play*, rather than individual heroics, is required from any but the most amateur players. Outside the game, *social relationships* and etiquette are required to be part of the successful clans (self-organized groups of players). Players can find partners for a game instance through the use of community websites, which may include services that matchmake players to a game instance, yet are not affiliated with the game developer.

In the absense of explicitly expressed relationships, understanding the social networks of current OSGs must rely on extracting the *implicit* social structure indicated by regular player activity. However, in contrast to general social networks, a set of meaningful interactions has not yet been defined for OSGs. Moreover, in MOBAs, activities are match- and team-oriented, rather than individual. We address these challenges, in Section 3.2,

through a formalism for extracting implicit social structure from a set of OSG-related, meaningful interactions. We apply this formalism to different game genres to extract implicit social networks, study the extracted social networks, and then analyze whether the extracted social networks can be used to improve gaming experience of players. Our major contribution is four-fold:

1. We propose a formalism which can be used to extract implicit social networks in OSGs.

2. We show that the implicit social structure of OSGs is strong, rather than the result of chance encounters, and that, for MOBAs, the core of the network (the high-degree nodes) is robust over time.

3. We apply our formalism to RTS and Massively Multiplayer Online First-Person Shooter (MMOFPS) games, and, in Section 3.3, show evidence that RTS games exhibit even stronger team structure than MOBAs and indicate that modern MMOFPSs may require operator-side mechanisms to spurn the formation of meaningful social structure.

4. We also show how the extracted implicit social graphs can be useful for improving gameplay experience, and for player and group retention (Section 3.4), for tuning the technological platform on which the games operate, etc.

3.1 Background

Defense of the Ancients (DotA) is an archetypal MOBA game. For DotA, social relationships, such as same-clan membership and friendship, can improve the gameplay experience [68]. DotA is a 5v5-player game. Each player controls an in-game avatar, and teams try to conquer the opposite side's main building. Each game lasts about 40 minutes and includes many strategic elements, ranging from team operation to micro-management of resources.

To examine implicit relationships in DotA, we have collected data for the DotA communities Dota-League and DotAlicious. Both communities, independently from the game developer, run their own game servers, maintain lists of tournaments and results, and publish information such as player rankings. We have obtained from these communities, via their websites, all the unique matches, and for each match the start time, the duration, and the community identifiers of the participating players. After sanitizing the data, we have obtained for Dota-League (DotAlicious) a dataset containing 1,470,786 (617,069) matches that took place between Nov. 2008 and Jul. 2011 (Apr. 2010 and Feb. 2012).

3.2 A Method for Analyzing Implicit NVE Social Networks

3.2.1 Social Relationships in OSGs

A *mapping* is a set of rules that define the nodes and links in a graph. Formally, a dataset D is mapped onto a graph G via a mapping function $M(D)$, which maps individual players to nodes (graph vertices) and relationships between players to links (graph edges).

Instead of proposing a graph model, we focus on formalizing mappings that extract graphs from real data. Because many metrics of social networks only apply to unweighted graphs, relations are often considered as links only if their weight exceeds a threshold. *Thresholding*, therefore, has an important impact on the resulting graph.

Related to our work, interaction graphs [229] map users of social applications to nodes, and events involving pairs of users to links via a threshold-based rule.

3.2.2 Interaction Graphs in MOBAs

A mapping is *meaningful* if it leads to distinct yet reasonable views of implicit social networks appearing in networked games. We identify five types of player-to-player interactions:

SM: two players present in the *Same Match*.

SS: two players present on the *Same Side* of a match.

OS: two players present on *Opposing Sides* of a match.

MW: two players who *Won* a match together.

ML: two players who *Lost* a match together.

To extract the social networks corresponding to various types of relationships, we extract for each mapping a graph by using a threshold n, which reflects the minimum number of events that need to have occurred between two users for a relationship to exist; e.g., for $SM(n = 2)$, a link exists between a pair of players iff they were both present in at least two matches in the input dataset. A second threshold, *t*, limiting the duration-of-effect for any interaction, is less relevant, as explained in Section 3.2.4.

The set of mappings proposed here is not exhaustive. For example, this formalism can support more complex mappings, such as "played against each other at least 10 times, connected through ADSL2, while located in the same country". The interactions in the set are also not independent. For example, the SS mapping can be seen as a specialization of the SM mapping.

3.2.3 Graph Metrics

To compare the graphs extracted using different mappings, we calculate a number of network metrics for the largest component of each extracted graph. The selected metrics all reflect properties related to the degrees and paths between players, and allow us to study the social relations in the gaming community. The metrics used in this section are explained below. For a more in-depth explanation we refer to [51, 163].

Size(s) of the connected component(s) (N, L): The size of the largest and other connected components indicates how many fellow players a player can reach in the network.
Link density (d): The link density is obtained by dividing the number of links in the network by $\binom{N}{2}$ and indicates how densely connected the network is.
Degree distribution: The degree distribution characterises the number of direct neighbours of a node.
Diameter (D): The diameter is the longest shortest path, in terms of hops, in the network.
Average clustering coefficient ($\bar{C}$): The average clustering coefficient is a measure for how many neighbours of a node are also neighbours of each other.

3.2.4 Application to the Examples

We focus in this chapter on three methodological questions:

Q1. Are the relationships we identify the result of players being simultaneously online by chance? To answer this question, we first create a reference model by randomizing, for any window of length w minutes, the interactions observed in the MOBA datasets. The randomization of, for example, the SM mapping is done by taking the players from all matches that started within the current time window and randomly assigning them to matches. Since the SM mapping does not take team information into account, the match assignment comes down to forming random groups of 10 players from the entire list who were active in the time window. A single player can be in the list multiple times and the random groups have to consist of 10 different players.
We run the parameter w from 1 to ∞ and depict the results, together with the original data, in Figure 3.1. Whereas the results for $w = 1$ leave little room for randomization, the results for $w = \infty$ randomize the entire dataset. In Figure 3.1, the curves for $w = 1$ and the original data have a powerlaw-like shape. The curves for various values of w follow the $w = 1$ (original) curve for link weights of up to about 5 matches played together, but take afterwards an exponential-like shape, which indicates they are more likely to be the result of chance than of intended user behavior. The fact that curves are markedly

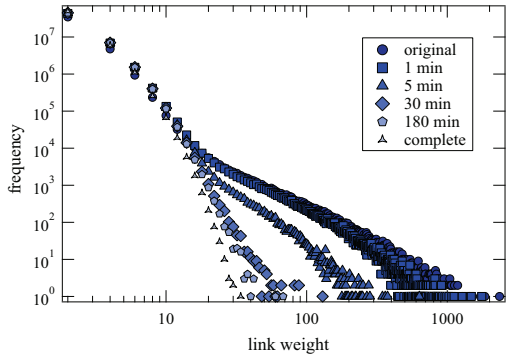


Figure 3.1: Frequency of occurrence of link weights in the reference model for five different intervals and the original data for DotA. The mapping is SM.

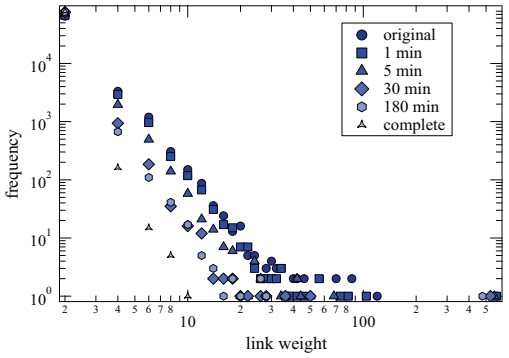


Figure 3.2: Frequency of occurrence of link weights in the reference model for five different intervals and the original data for SC2. The mapping is SM.

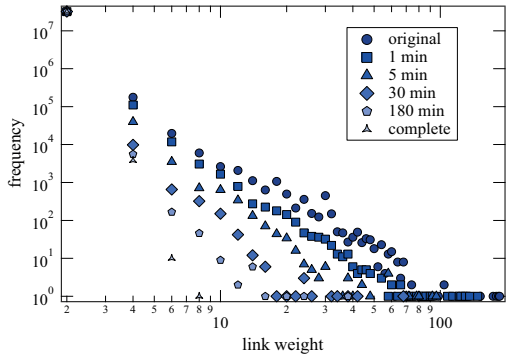


Figure 3.3: Frequency of occurrence of link weights in the reference model for five different intervals and the original data for WoT. The mapping is SM.

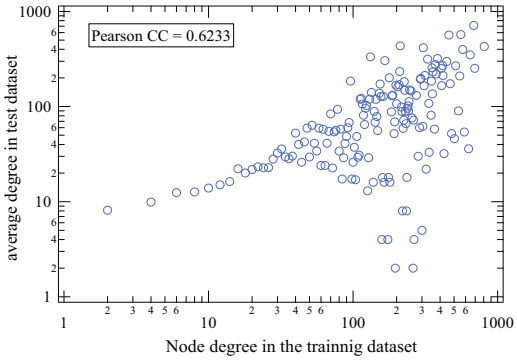


Figure 3.4: Continued activity for gaming relationships in Dota-League (SM with $n = 10$).

different for small time windows shows that it is very unlikely that players play together often simply because they happen to be online at the same time.

The results for the other game genres (genres introduced in Section 3.3, results depicted in Figures 3.2 and 3.3) show similar, yet not so pronounced behavior. Although players do not play nearly as often together in other genres' datasets as in the MOBA datasets, randomization within only small time windows lowers the link weights. We conclude that it is unlikely that the relationships we identify are the result of chance encounters between players and, instead, indicate conscious, possibly out-of-game agreements between players.

Q2. Are players (nodes) preserving their high-degree property over time? If so, then the networks these players form may be robust against natural degradation, with implica-

Table 3.1: Results for methodological question Q3. Metrics for $n = 10$: (top) Data for MOBA games: the Dota-League and DotAlicious datasets. (bottom) Data for other game genres: StarCraft II (RTS) and World of Tanks (MMOFPS). The metrics we present: number of nodes N, number of nodes in largest connected component N_{lc}, number of links L, number of links in largest connected component L_{lc}, link density d, link density of largest connected component d_{lc}, diameter D, average clustering coefficient $\bar{C}$.

	DotA-League					DotAlicious				
	SM	OS	SS	ML	MW	SM	OS	SS	ML	MW
N	31,834	26,373	24,119	18,047	18,301	31,702	11,198	29,377	22,813	21,783
N_{lc}	27,720	19,814	16,256	6,976	8,078	26,810	10,262	20,971	10,795	13,382
L	202,576	85,581	62,292	30,680	33,289	327,464	92,010	108,176	43,240	54,009
L_{lc}	199,316	79,523	54,186	17,686	21,569	323,064	91,354	99,063	29,072	44,129
d ($\times 10^{-4}$)	4.00	2.46	2.14	1.88	1.99	6.52	14.7	0.49	1.66	2.28
d_{lc} ($\times 10^{-4}$)	5.19	4.05	4.10	7.27	6.61	8.99	17.4	2.51	4.99	4.93
D	14	21	24	28	26	17	12	19	20	22
$\bar{C}$	0.37	0.40	0.41	0.41	0.41	0.43	0.27	0.47	0.47	0.49

	StarCraft II					World of Tanks				
	SM	OS	SS	ML	MW	SM	OS	SS	ML	MW
N	907	611	314	95	212	4,340	477	4,251	561	1,824
N_{lc}	31	22	24	9	14	129	118	122	66	57
L	748	404	327	85	200	9,895	3,253	6,543	1,564	2,923
L_{lc}	58	21	44	13	24	2,329	1,243	1,160	519	473
d ($\times 10^{-4}$)	18	22	67	190	89	10.51	286.54	7.24	99.57	17.58
d_{lc} ($\times 10^{-4}$)	1,247	909.10	1,594	3,611	2,637	2,821	1,801	1,572	2,420	2,964
D	2	8	3	2	2	6	3	5	4	3
$\bar{C}$	0.58	0	0.70	0.65	0.65	0.79	0.10	0.78	0.88	0.87

tions for the long-term retention of the most active players. For each MOBA-community, we first divide its last-year's gaming relationships into two parts: the first half year as training data and the second half year as testing data. We only use players who appear in both datasets—about 60% of the training-data players. Then, we plot in Figure 3.4, for different degrees of players in the training dataset, the average number of links formed by these players in the testing dataset. From the high-value and positive correlation-coefficient (0.6233 for Dota-League), we derive that players with higher degrees in the training dataset robustly establish more new links in the testing dataset than the other players.

Q3. Are the mappings we propose meaningful for MOBAs? To answer this question, we first extract the interaction graphs for each of our mappings, compute for each a variety of graph metrics, and summarize the results in Table 3.1. We find that:

- Side-specific interactions (SS and OS) are meaningful. For example, playing on the opposing side (OS) is more likely than playing on the same side (SS), in Dota-League (for example, higher N and L in Table 3.1); for DotAlicious, the reverse is true. Game designers could enable OS links by allowing players to explicitly identify their *foes*.

- Outcome-specific interactions (MW and ML) are meaningful. For example, only

for DotAlicious, MW leads to more relationships being formed. Game operators could exploit this in matchmaking services. Identifying the players who play almost exclusively together can be key to player retention.

3.3 Application to Other Game Genres

Among the most popular genres today, RTS games ask players to balance strategic and tactical decisions, often every second, while competing for resources with other players. Although faster-paced, MMOFPS games test the tactical team-work of players disputing a territory. We could expect RTS and MMOFPS games to lead to similar interaction graphs as MOBAs: naturally emerging social structures centered around highly active players. However, these game genres also have different match-scales and team-vs-team balance than MOBAs. Moreover, RTS games can stimulate individualistic gameplay, while MMOFPS games may have teams that are too large to be robust.

We collect then analyze two additional datasets: for the RTS game StarCraft II (SC2) from Mar. 2012 to Aug. 2013, and for the MMOFPS game World of Tanks (WoT) from Aug. 2010 to Jul. 2013. For each of these popular games, we have collected over 75,000 matches, played by over 80,000 SC2 and over 900,000 WoT players. SC2 matches are not generally played in equally-sized teams, and 92% of our dataset's matches are 1v1-player. In contrast, 98% of WoT matches are 15v15-player, but such large teams can be much harder to maintain over time than the teams found in typical MOBAs, due to inevitable player-churn.

Alone or together? For SC2, the mappings lead to small graphs, with many small connected components. The majority of players participate in 1v1-player matches, but the 8% of players who do play in larger groups tend to play against each other more than together ($N = 611$ for the OS mapping, versus 314 for SS). When players do play on the same side, winning tends to strengthen the teams ($N = 212$ for the MW mapping, versus 95 for ML), just as we saw for the DotaLicious dataset. The connected components are strongly connected, yet small. The connected components of the mappings extracting same-team graphs are highly clustered, whereas the largest component for the OS mapping is even a tree. The clustering coefficients observed in the various RTS networks indicate much stronger team relationships in RTS games than in MOBAs. Because RTS games have not shown a trend of greatly increasing the number of players in the same instance, over the last decade, we hypothesize that RTS games will continue to spawn tightly-coupled teams that always play together; such teams are naturally vulnerable to player departures.

For WoT, the large team-size makes it difficult to organize teams well: the largest connected components for all mappings are not very large. Similarly to SC2 and DotaLicious, in WoT the players who do play often together do so on the same team and, again, players who play together are more likely to win rather than lose together. As modern

FPS games tend to be played in increasingly larger teams, with 32v32-player games now not uncommon, we conclude that MMOFPS games will require additional mechanisms if they are to develop any form of robust social structure. Moreover, even more so than in the SC2 datasets, many players play only one or a few games: 69% of the more than 900,000 players played only once or twice. This is another area where developers could use the emerging social structures among their players to increase the number of players who keep on playing the game.

We conclude that our formalism can be applied to other game-genres, for which it leads to *new findings* vs MOBAs, and suggest that even communities of popular networked games could benefit from new mechanisms that foster denser interaction graphs.

3.4 Application to OSG Services

"How can social-networking elements be leveraged to improve gaming services?" We present in this section two exemplary answers.

3.4.1 Socially and Network-Aware Matchmaking

Matchmaking players at the start of a game can significantly impact the gameplay experience. Gaming services that perform matchmaking while taking into consideration network latency are already deployed by game operators. In contrast, a *socially-aware* matchmaking service assigns players to matches, trying to ensure that players in the same social, rather than latency-based, cluster play together. We propose a socially-aware matchmaking service which is described in the following.

Socially-aware matchmaking algorithm First, for each sliding window ($\tau = 10$ min. interval), the algorithm builds a list of all the players who are online. Second, from the social graph the algorithm computes the cluster membership for each player. Third, from the largest online players' cluster to the smallest, all online players from the same cluster are assigned to new matches if size permits; otherwise, the cluster will be divided into two parts and players from one part will be assigned into new scheduled matches.

Figure 3.5 sketches the algorithm for computing the score for an exemplary match. Team 1 consists of players 'a' to 'e', as can be seen in the column labeled 'Player'; team 2 consists of players 'f' to 'j'. The column labeled 'Cluster' records the cluster identifier for each player. A match receives one point for every same-cluster player present in the match, when at least 2 same-cluster players are present. In this example, 2 points are given for player 'a' and 'c' (cluster 1), and for players 'b' and 'f' (cluster 2); 3 points are given for players 'd','h', and 'j' (cluster 3). Players 'e','g', and 'i' have no fellow cluster-members in the match and will be assigned 0 points. In total, this match is assigned 7

Team 1

Player	Cluster
a	1
b	2
c	1
d	3
e	4

Team 2

Player	Cluster
f	2
g	5
h	3
i	6
j	3

Figure 3.5: Example of scoring for a match.

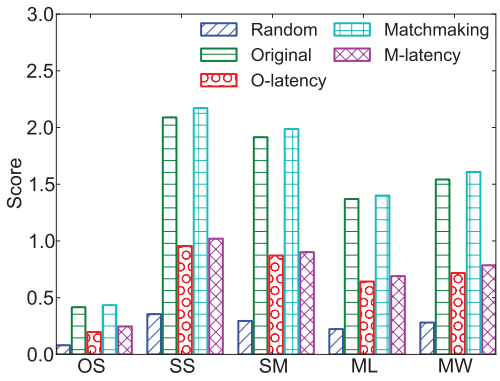


Figure 3.6: Average match scores for various matchmaking approaches.

points. To favour small clusters, which can lead to novel human emotions [148], our scoring system does not consider the largest cluster when assigning points.

We compare our matchmaking algorithm with the algorithms observed in practice in MOBAs in terms of average scores (*utility*), and show selected results in Figure 3.6. When considering network latency, matches with players on several continents score 0 points; the others use the scoring exemplified in Figure 3.5. "Random" denotes matches obtained via randomly matching players who are online during each time interval. "Original"/"O-latency" denote matches observed in the real (raw) datasets without/with network latency considerations. "Matchmaking"/"M-latency" denote matches obtained with our proposed matchmaking algorithm without/with network latency considerations.

Expectedly, random matchmaking, which is still employed by many gaming communities, leads to very low utility. Surprisingly, our simple socially-aware matchmaking algorithm also exceeds the performance of the matchmaking algorithm employed by the operators of DotAlicious; this is because the limited community tools available in practice do not make all players aware that some of their friends are online and thus allow them to join other, lower-utility, matches.

Including network characteristics We use the geographical location gleaned from

MOBA datasets to estimate possible latency conflicts, e.g., same-match players located in Germany and Asia. We analyze the impact of network latency on the score of our matchmaking algorithm and depict the resulting score in Figure 3.6. In this scenario, a significant part of the matchmaking score is lost due to recommendations not taking into account network latency (yet our matchmaking algorithm still outperforms the original matchmaking). We conclude that combining social and network awareness is important for networked gaming services.

3.4.2 Assessing Social Network Robustness

Because social relationships are important in player retention [148], the strength of the social structure may be indicative for the survival chances of the community. If the network starts dismantling, people might lose interest in the game and stop playing. Operators need to assess both the strengths and weaknesses of their games' social structure, to be able to stimulate growth or to prevent a collapse. Conversely, competitors could try to lure away key players (*hubs*), who in turn could sway others.

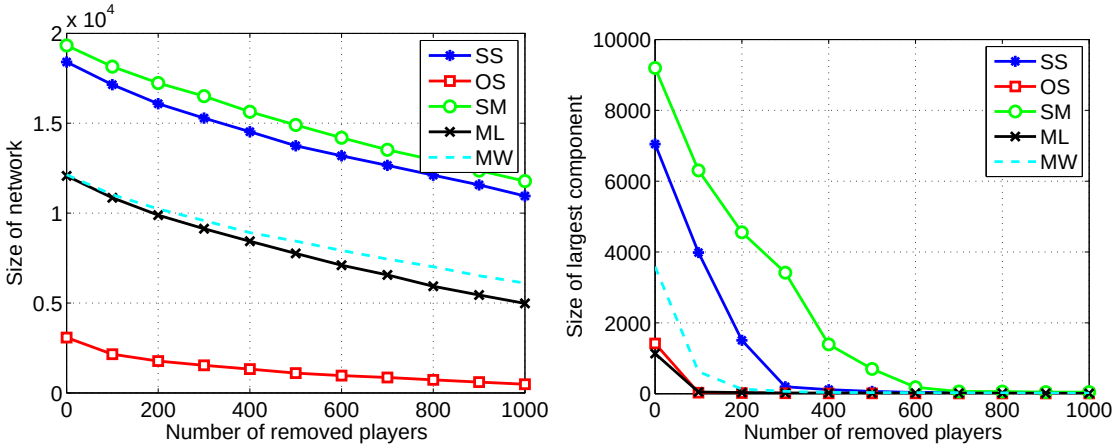


Figure 3.7: Effect of lost players on network-size, during a match attack, for DotAlicious: (left) size of remaining network; (right) size of the largest connected component in the remaining network.

The anatomy of an attack: To assess the social-network robustness, we conduct a *threshold-based degree attack* on the network: for each mapping, we iteratively remove the *top-K players*, according to their degrees in the extracted graph, in decreasing order. Removing a player either also removes their matches (*match-attack*) or also removes their entire connected component (*hub-attack*). Then, we re-apply the mapping to the remaining matches to get a new network, and output the size of the new network and largest component. We perform match and hub-attacks on DotAlicious and DotA-League and depict selected results in Figure 3.7 and 3.8. (We do not conduct experiments in which

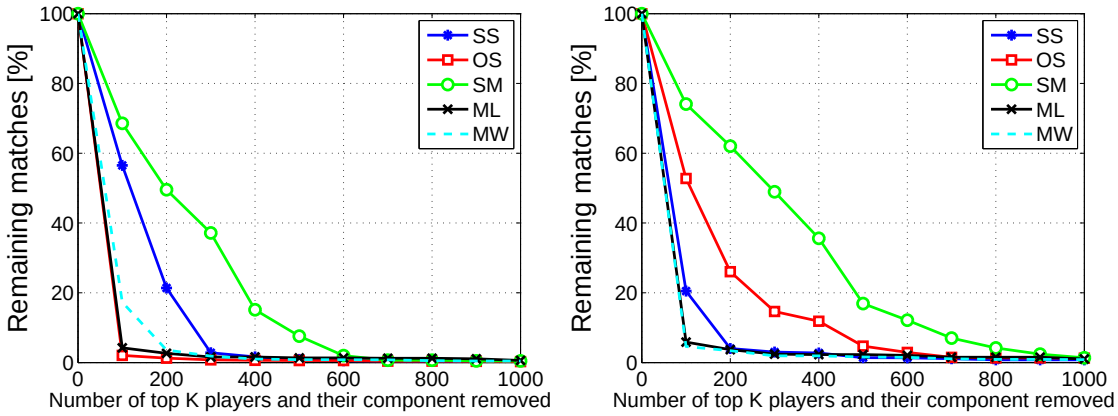


Figure 3.8: Effect of lost players on match count, during a hub attack, for DotAlicious (left) and Dota-League (right).

players form new clans (network rewiring), which represents the opposite of our scenario; in our experience as gamers, when a member of a strongly connected group leaves (for another game), the whole group departs as well.)

The aftermath of an attack: We find that both match and hub-attacks on MOBAs are very efficient. For match-attacks (Figure 3.7), removing the top-1,000 players (1.5%) can reduce the size of the network by 15% up to 60% of its initial size, and the size of largest component to below 10. For hub-attacks (Figure 3.8), removing only the top-100 players can cause the network to implode. A social-network collapse also implies the collapse of network traffic, which may lead to waste of pre-provisioned networked resources.

We conclude that understanding the social relationships between players can help a game operator improve the social-network robustness, by identifying and motivating the key players. Our formalism provides important tools for the former, but the latter remains open.

3.5 Related Work

Social network analysis and complex networks theory have received increasing attention in the past few years, which has readily resulted in a significant body of related research papers. We refer to [67, 207] for an overview of research on complex networks and to [25, 190] for some excellent overviews of the developments and state of the art in social network analysis. Most research on social network analysis, however, only considers or defines one network for one type of link. Instead, we consider the influence of different (social) link definitions and combinations of link definitions on the emerging (complex) network.

Within the application domain of online social games, few studies use network met-

rics to divide players into different classes. In [122], for example, the authors extract a network from an online game by creating unweighted and undirected links between players that ever exchanged information in the game. They define three types of players based on the successive removal of the highest-degree nodes until the largest connected component falls apart. In [202], an implicitly defined network is used to predict future player performance based on the relationship between mentors and apprentices. This analysis could be extended by looking at network-wide properties instead of only local properties. The prediction of the success in games can also be applied to real-world games as is done in [55], where a complex network approach is used to predict the performance of basketball teams. The authors propose a network-based ranking of players as a replacement for current statistics such as assists and points scored to predict the future success of a team.

Other studies use different definitions of links to create different networks from the same dataset. In [211], the authors study a detailed dataset of interactions and friend/enemy relations spanning three years in the online game Pardus. They use the game as a substitute of the real-world and test several hypotheses in the field of social dynamics such as social balancing, network densification and triadic closure. They study three different networks extracted from the dataset: the network of communication between players, the network of friends as indicated by players in the game, and the network of enemies as indicated by players in the game. Although the authors present a detailed analysis of the networks for each type of interaction, and especially contribute to existing work by analysing the network of enemy relations, their links are either interaction based or explicitly indicated by the players.

A related study on guild members in World of Warcraft [7] also studies the differences between networks formed based on different types of interaction. In this chapter social network analysis is used to explore the network structure of interactions between guild members in the online game World of Warcraft. The authors studied 76 players that formed a single guild and extracted networks by creating links between players that communicate amongst each other. Different types of interaction were classified into seven different categories such as asking for help or group management to form seven different networks. An analysis of these networks in terms of reciprocity and topological structure indicates that the different types of interaction lead to different networks.

3.6 Summary

Many current online games provide limited social-networking features, yet rely on their players to self-organize. For example, games in the popular class of MOBA-networked games have fostered the creation of many communities of players. In this chapter, we have shown how a general formalism can be used to extract social relationships from the interactions that occur between networked-game players. We have investigated their implicit

social structures based on five types of interactions, using community traces that characterize the operation of four popular MOBA, RTS, and MMOFPS games, and provided hints on improving gaming-experience through two socially-aware services.

Chapter 4

Analyzing Online Meta-Gaming Networks

Online Meta-Gaming Networks (OMGNs) such as Valve's Steam, Sony's PlayStation Network, and Xfire are for tens of millions of players an important way of integrating in a like-minded, game-oriented society. For example, the almost 20 million gamers participating in the XFire online meta-gaming network may discuss and share game-related media related to over 2,000 different computer games. Understanding the characteristics of meta-gaming networks can lend important help to the design and tuning of OMGNs. However, despite an increase in these network's popularity over the past two decades—one of the first meta-gaming networks was built by America Online (AOL) in the early 1990s—, the characteristics of meta-gaming networks remain relatively unknown. In contrast, much previous work has focused on observing and analyzing other platforms for community-creation [123,134] and media-sharing [37,133,153,176]. To address this gap, in this chapter we report on the long-term observation and resulting high-level analysis of XFire.

Characterizing OMGNs is important for the design and operation of OMGNs. The need for timely and adequate OMGN deployments pressures system designers and operators into taking, on short notice, important decisions about system scalability, security, and usability. Such decisions can greatly benefit from a good understanding of the community size, structure, and activity. For example, provisioning the resources needed for the operation of an OMGN can benefit from an understanding of the evolution of the number of OMGN users, coupled with statistical information about the resource consumption incurred by each user. Inadequate designs can have disastrous consequences, such as the forced shut-down of the Sony PlayStation Network, following a security breach [212].

OMGNs may differ significantly from other (Internet-based) communities and social networks. Meta-gaming networks may add to the study of human communities [84, 90] a new dimension, which stems from their competitive (even adversarial) context—most

participants in OMGNs are gamers. Due to their multi-game coverage, OMGNs such as XFire may also be different from the communities that form around individual game titles, for example Massively Multiplayer Role-Playing Game World of Warcraft [66] or the casual online social game Fighters Club [162]. Similarly, studies of OMGNs may complement earlier studies of in-game player activity and behavior [40, 74] with an out-of-game component.

Our long-term objective is *to create a theoretical and practical foundation for study of analyzing meta-gaming networks*. Much work needs to be done to achieve this objective, among which observing and analyzing OMGNs raise important challenges. The quantitative assessment of OMGNs is made more difficult by the decentralized system designs and by the confidential nature of the data. Often, observing without the cooperation of the OMGN operators is the only way to obtain data. Since human communities are subject to attrition and evolution, data need to be obtained over long periods of time to be conclusive. Even when data are obtained, the problem of extracting useful information from them may require new models and algorithms. As a first step toward our long-term objective, our main research question is, in this chapter, *What are the characteristics of an Online Meta-Gaming Network?* To answer this question, we focus on the observation and high-level analysis of the XFire network, where by "high-level analysis" we mean the analysis of marginal distributions for a number of important characteristics. Our main contribution is threefold:

1. We propose a method for the study of OMGNs, which is based on repeated observation and high-level analysis (Section 4.2).

2. We collect a long-term dataset from XFire (Section 4.3).

3. We present a high-level analysis of the XFire dataset, which focuses on the global network (Section 4.4.1), on gaming activity (Section 4.4.2), on user-generated content (Section 4.4.3), and on social structure (Section 4.4.4).

4.1 Background

In this section we present the context required to understand OMGNs. Context is particularly important for large-scale quantitative studies such as ours, increasing familiarity with the subject (needed to plan observational studies) and allowing for an understanding of potential measurement biases. We first introduce meta-gaming and OMGNs, then discuss the focus of this chapter on a particular OMGN, XFire.

Meta-gaming, defined as "the game beyond the game" [81], refers to game-related (but mostly not in-game) connections between people. The meta-game connections can affect both positively and negatively the way players think about and act within the game. A

large number of players may be persuaded by the social pressure of their game-friends and game-peers to continue playing a game or a game genre over many years. A community of players may exchange information and educate its weaker members about the best strategies of a game. A group of players may collude to influence the outcome of a tournament [156]. A poker player may purposely lose a hand to better understand the bidding behavior of an opponent [81].

An Online Meta-Gaming Network is an online social network [90, 123, 134] that allows its participants to manage their metagame connections, through the following set of **core features**: (instant) messaging; file sharing; screenshooting (capturing and publishing screenshots), videoshooting (capturing and publishing videos); screencasting (live streams of image as seen on the computer); approx like broadcast; etc.

One of the earliest metagaming networks was built for *Neverwinter Nights*, an early MMORPG hosted by AOL since 1991. For this game, the emergence of in-game guilds of like-minded players triggered support from AOL, with installed and operated forums [33, p.160]. One of the first metagames launched simultaneously with the online social game *Animal Crossing* (*Dōbutsu no Mori*), Nintendo, in 2002 (2001). In *Animal Crossing*, players use the metagame network to exchange messages and to send in-game objects as gifts; socialization, both in-game and through the metagame, is the key feature of the game [116, Ch.6].

Tens of millions[1] of players have joined recently online metagaming networks such as XFire (`http://xfire.com`), Valve's Steam, Sony's PlayStation Network, Microsoft's XBox and Games for Windows Live, and Zynga's integration with multiple social networks (a distributed OMGN). All of these networks offer all the core OMGN features, with different implementations.

4.2 A Method for Studying Online Meta-Gaming Networks

In this section we propose a method for the study of OMGNs that is based on repeated environment observation and its high-level analysis. Our method addresses two main problems, the collection of data from OMGNs and the focus of analysis.

Our method is based on the principles of *observational studies* [184] [166, Ch.6.5], that is, that the study does not involve the intervention of the investigator; instead, the investigator can only observe (record and analyze) the environment. The alternative of using intervention studies, that is, studies in which the investigator can alter one or more factors affecting the environment to later study the effects of the alteration, is often uneth-

[1] We will show in Section 4.4.1 that XFire is a community of about 20 million players. The Sony PlayStation Network had at least 77 million accounts [212].

ical and rarely feasible. The negative impact of this (forced) choice is that the root causes of the observed situations cannot be established rigorously.

We further base our method on a repeated cross-sectional design. Among several types of designs for observational studies, *cross-sectional* designs focus on the observation of a part of the environment at a single moment of time. A *repeated cross-sectional* design uses the data and/or results of several cross-sectional studies of the same environment; to improve the statistical relevance of the results, the repetition is periodic. The cross-sectional design can capture net effect changes, such as overall increases and decreases of the population, trends in the population taste and activity, etc., and can support the evaluation of marginal distributions (such as the probability and the cumulative distribution function, or the *PDF* and the *CDF*, respectively) for a wide range of OMGN characteristics. Alternatively, longitudinal designs would sample over time the same population, sometimes of the same age, which could lead to improved statistical power for the study but is impractical for OMGNs due to attrition; surveying and case studies are impractical for OMGNs, where demongraphic information is unavailable before the study.

Bootstrapping the data collection process Because the members of the OMGN are not known before the study (the *bootstrapping problem*), the data collection part of our method needs an adaptation to the context of OMGNs. We propose an approach for the bootstrapping problem, based on *participant self-selection*, where we first observe the participants to the discussions and media-sharing activities of the network for a period, then use them to bootstrap the traversal of the network; exploiting the high-connectivity of social graphs, by traversing only one further connection in the social graph, information may be obtained about a significant fraction of the OMGN members. If incomplete, the traversal should use a random selection of connection traversal. As a possible alternative, unbiased sampling methods are difficult to develop and depend on the properties of the network, which may be unknown before the study, to reduce the bias introduced by bootstrapping [208].

Observed environment variables With the specific goal of building in the future systems that can better and more efficiently support OMGNs, we focus on four main components of the operation of OMGNs: on the global network, on gaming activity, on user-generated content, and on social structure. For the *global network*, we focus on the size of the community and on the time spent collectively by the OMGN members in-game. At the level of individual players, we focus for the *gaming activity* on the number of games played, of the time spent in-game (both for all games and for each game, individually). The *user-generated content* analysis focuses on the production and consumption of content, where content may be any multi-media product that an OMGN member may share with other members. The analysis of production follows the counts of produced items per player and further per game. The analysis of consumption follows the counts of views or other types of uses of produced items. Last, the *social structure* follows the

Table 4.1: The XFire community datasets.

	Bootstrap	Global Network	Player
Period	May 2008 to	Sep 2010 to	Sep 14–16
	Sep 2010	Jun 2011	2010
Samples	1/hour	1/hour	1
Size [GB]	9.2	6.5	15.7
Players	65,908	*not applicable*	544,902
Game genres	25	25	*not applicable*
Games	1,100+	1,400+	*not applicable*

creation of connections in two types of communities: structured (guilds, clans, etc.) and free-form (friends, buddies, etc.)

Application We describe in Section 4.4 the application of our method on XFire, which we select as an exemplar of OMGN. Among the OMGNs introduced in Section 4.1, XFire is the only popular network that is not affiliated with any individual game producer and does not include direct game sale as part of its business model. As a consequence, XFire reveals more information per player than the other OMGNs and does not have a visible incentive to bias recommendations for popular games and derivative multi-media, such as screenshots and videos.

4.3 Datasets

In this section we present the procedure used for collecting data from XFire. The data were acquired without the cooperation of the XFire operator, which is similar to our previous experience with many large-scale communities. The acquisition process is periodic, and consists from a combination between crawling and parsing web data. Our data collection process contains three parts: data collection for solving the bootstrap problem (see previous section), followed by separate data collection for the global network and for the individual players. We have written in Python custom tools for each part.

Data summary XFire offers rich information concerning members, games, and game genres. A summary of the collected datasets is presented in Table 4.1. Overall, we have collected data over a period of more than 3 years, out of which the last 10 months have produced the data analyzed in Section 4.4. The Global Network dataset does not include detailed player information; the Player dataset does not include detailed game and game genre information. Over the period covered by these datasets, we have suffered an infrastructure downtime that affects under 1% of the samples in non-adjacent short periods.

The Bootstrap dataset Our collected XFire data includes member identifiers in the pages related to game genres and individual games; the included members have published game-related content or have been part of a popular community or event. (We intend to investigate the correlation between members and games in a future analysis of this

dataset.) We have collected in the Bootstrap dataset (Table 4.1, column "Bootstrap") information about all the 25 game genres and an evolving number of games tracked by XFire over a period of over 2 years. We have identified in this dataset over 65,000 players.

The Global Network dataset Our collected XFire data includes information about in-game use for each game (in hours, updated every day) and for the most played four games (in minutes, updated at every access), the number of registered and online players (updated at every access), a grouping of games by genre, etc. (We intend to further analyze this rich dataset in the future.) The duration of in-game presence is recorded by the tools provided by XFire, which are installed by each player's gaming system. For this dataset, there are no missing data: when taken, samples are complete. Although cheating is possible, we believe that the closed-source nature of the official tool and the openness of the community prevent wide-spread mis-reporting.

The Player dataset For this dataset we have followed the two-step procedure detailed in Section 4.2: we first collected information from the over 65,000 members identified in the Bootstrap dataset, then collected information about all their recorded friends. For the former, we were able to retrieve information only about 61,229 members; for the later, we were able to collect information about 483,673 more, randomly selected out of the 1,393,090 friends identified in the first step. In total, we have collected detailed, recently-updated information about over 500,000 XFire members: the number of friends (limited by XFire at 1,000), the list of joined communities and their sizes, the list of played games and time spent in them, etc. The total time spent for each game is measured from the date of the player registration, which may be as early as 2003, when XFire was launched. (We intend to further analyze this rich dataset in the future.) The reporting of time spent in-game is done with the same XFire tools as for the data present in our Global Network dataset.

Members or players? A member of XFire does not necessarily need to play games. We have analyzed the Player dataset and found that only 25,208 members (5.01%) have not played at least one game or at least one total hour over all played games. As a consequence, we will use in this chapter the terms "member" ("user") and "player" ("gamer") interchangeably.

4.4 Characterization Results

In this section, we perform a high-level analysis of the XFire dataset.

4.4.1 Analysis of the Global Network

Registered and online players: XFire is a slowly growing community, with a slowly decreasing online presence of about 1% of its registered users. Figure 4.1 depicts the

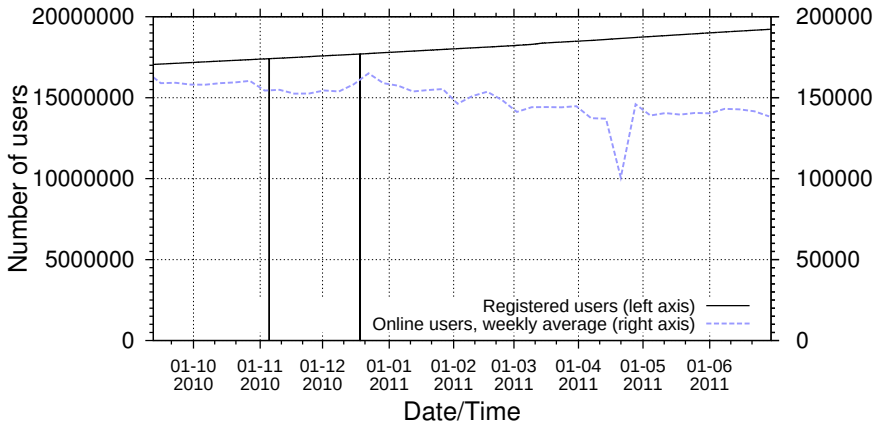


Figure 4.1: Number of players registered and online, over time.

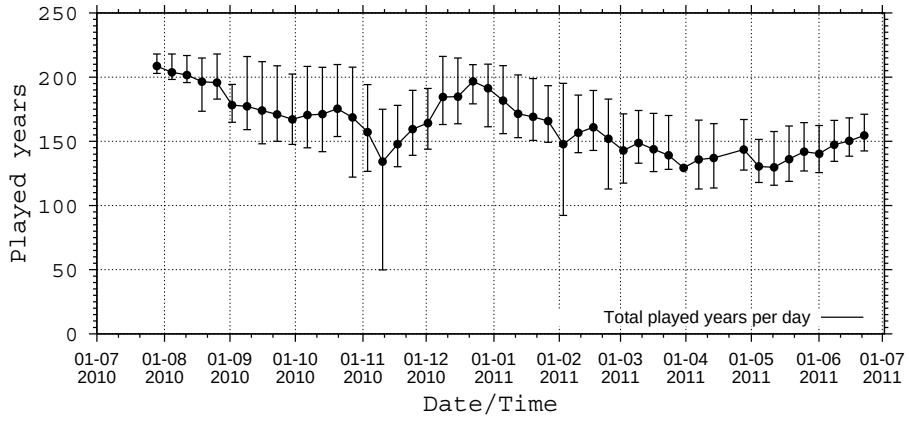


Figure 4.2: Time spent collectively in games by the XFire community, per hour, over time. Curves represent weekly averages; error bars represent weekly minima and maxima.

number of players registered and online, over time. There are about 20 million registered players in XFire. The number of registered users keep a steady increasing rate (330,000 or about 2% more users per month) during the observed period. The number of online players has an average of about 150,000 or 0.8% registered players, with a peak around 165,000 near the Christmas holiday; the online presence seems to be slowly decreasing (about 2% per month). For comparison, RuneScape, which is a free MMORPG that maintains an OMGN, has a much lower ratio of online players, about 200,000 out of over 135 million registrations (0.15%).

Collective in-game time: XFire members spend collectively over 100 years playing, every hour. Figure 4.2 depicts the time spent collectively in games by the XFire community, per hour, over time. The total time spent on games shows monthly effects: players play longer during the August and December vacations. In August, the total time spent on games per hour exceeds 200 years.

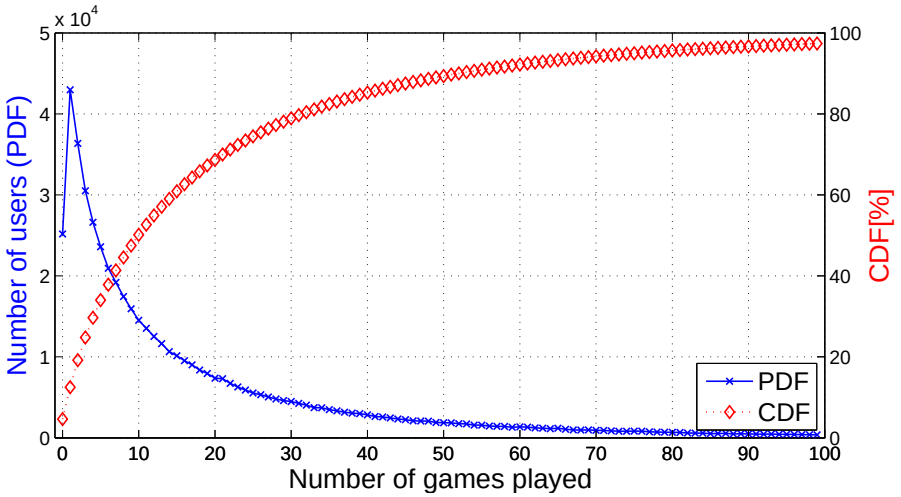


Figure 4.3: Number of games played in total, per player.

4.4.2 Analysis of Gaming Activity

Player activity, played games: The average XFire gamer has played over 20 games. Figure 4.3 depicts the number of games played in total, per player. (In Figure 4.3, the Cumulative Distribution Function (CDF) is depicted against the right vertical axis, while the Probability Distribution Function (PDF) uses the number of recorded observations instead of relative frequency and is depicted against the left axis; we use this graph structure for each subsequent depiction of a Pareto (combined PDF and CDF) graph.) The number of games played by player is rather heterogeneous. The average number of games played by player is 21 while the maximum value is 1,989. Less than 5% of registered users did not play at least one game (see also the discussion at the end of Section 4.3). More than 50% of the registered users played at least 10 games. The graph indicates that the player activity (played games) is very heterogeneous, active players who play more games than the other players may have important effects of the popularity of games.

Player activity, total play time: The average XFire player has spent over a month in-game. Figure 4.4 depicts the amount of time played in total, total per player—computed over all games played by the player, for each player. On average, each player spends in total about 875 hours (over 36 days) in-game. The distribution of amount of time played in total is long-tailed. About 15% of the XFire users played for less than 10 hours; at the other extreme, over 25% of them played for over 1,000 hours. Surprisingly, we found 1,241 (2,977) players who spent over 10,000 (8,000, or about the duration for obtaining a PhD**!**) hours online; the fraction of 0.228% (0.546%) is significant for a player population that nears 20 millions. The maximum total played time is 33,392 hours (3.8 years). Such long total play time may be explained by the "hardcore"-ness of some of the players (although it exceeds by far previously reported numbers [80]), or by the ability of players

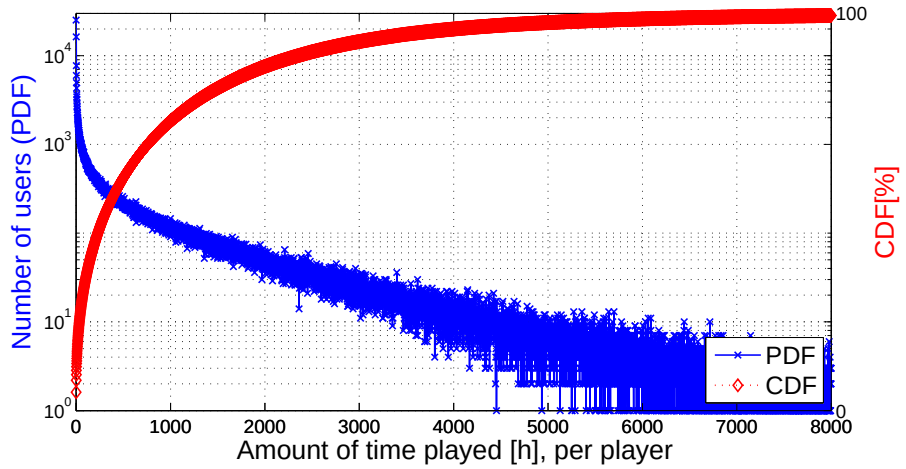


Figure 4.4: Amount of time played in total, total per player (over all games played by one player).

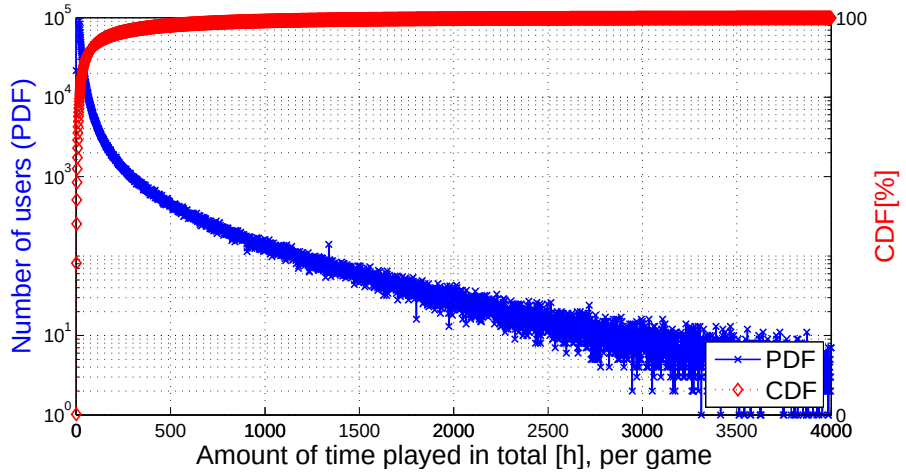


Figure 4.5: Amount of time played in total, per game.

to *multi-clock*, that is, to play simultaneously several games that count towards the total played time (for example, two or more browser-based games such as Zynga's FarmVille and Mafia Wars).

Player activity, play time per game: The average time spent with a game by a player is below one week, assuming 8 hours of play per day. Figure 4.5 depicts the amount of time played in total, per game. The popularity of games is highly skewed. Although the average time played per game is 43 hours, the median time played per game is 3 hours and about 90% of the items represent games played for less than 48 hours. At the other extreme, a player spent about 3.7 years in only one game.

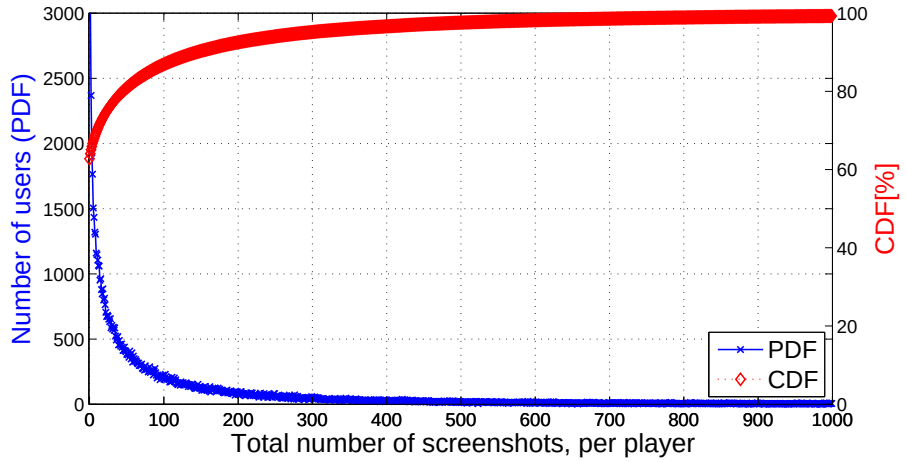


Figure 4.6: Total number of published screenshots, per player.

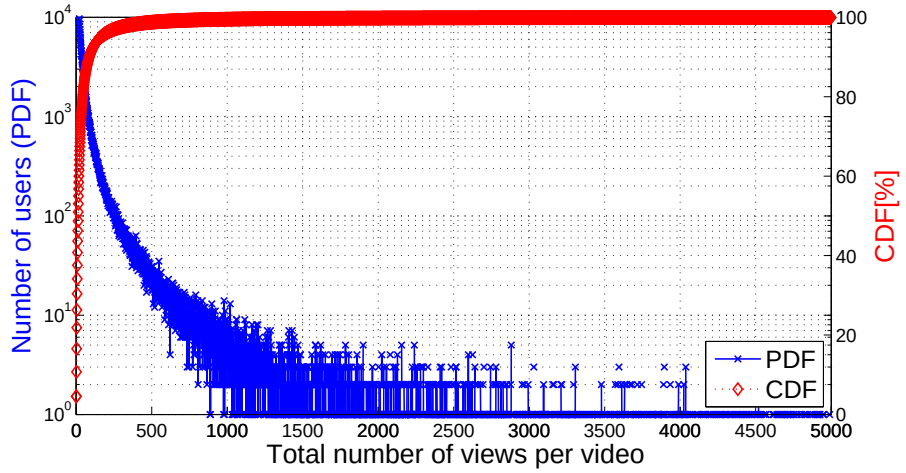


Figure 4.7: Number of views of published videos, per player.

4.4.3 Analysis of User-Generated Content

Content production, screenshots and videos: Screenshot production has a highly skewed distribution, with the top producer having over 8,000 publications. Figure 4.6 depicts the total number of published screenshots, per player. About 60% of players did not publish any screenshots; 80% of the players published less than 45 screenshots. The large imbalance between the maximum of 8,000 screenshots and the other values indicates that a Power-law distribution may be a good fit for the production of screenshots. We have found a similar distribution for the production of videos, but with much lower produced units. About 90% of players did not publish any videos. On average, each player produces only about 1.25 videos, with a maximum number of published videos of only 222.

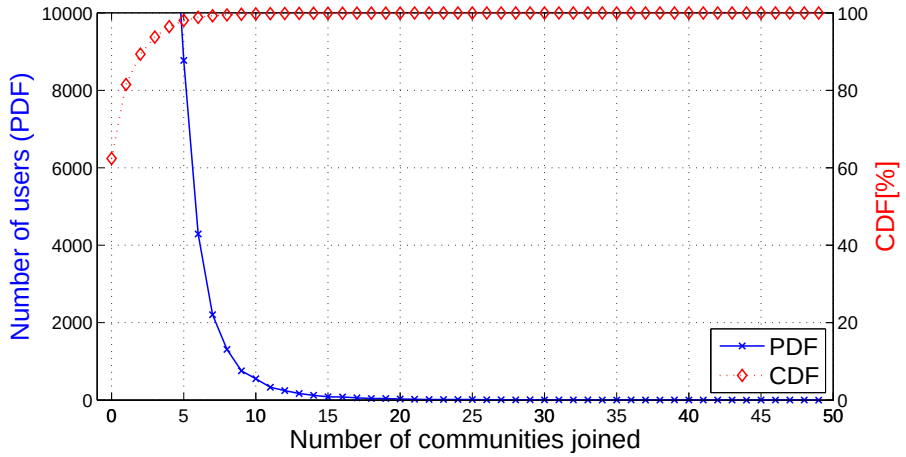


Figure 4.8: Number of communities joined, per player.

Content consumption, videos: Video consumption (viewing) has a highly skewed distribution, with the highest viewed content item totaling over 130,000 views. Figure 4.7 depicts the number of views of published videos, per player. On average, each video was viewed by 53 times. The popularity of videos is highly skewed: 60% of videos were only viewed less than 20 times, while the most popular video was viewed by 131,641 times.

4.4.4 Analysis of Social Structure

Belonging to structured communities (player guilds): Most players do not join a structured community; when they do, they usually join only one community. Figure 4.8 depicts the number of communities joined, per player. About 60% of players did not join any community. Most of the players only join less than 10 communities. The average player joins 1 community, while the maximum communities joined by a single player is 246 communities.

Belonging to free-form communities (player-to-player friendships): XFire players are "social creatures"—the average player has over 60 friends, and about 15% of the players have more than 100 friends. Figure 4.9 depicts the number of friends, per player. On average, each players have 63 friends, which is smaller than average friends number of Facebook (130 friends per person) [72] but still indicates a large social circle. Less than 0.06 % of players have no friends. About 15% of players have more than 100 friends. Because the maximum allowed number of friends in XFire is 1,000, we do not find extremely long friend lists. When the figure is plotted in logarithmic-scale both in horizonal and vertical axis, we observe a power-law-like tail of the distribution number of friends, which is similar with social network research of friendships within a game club [162].

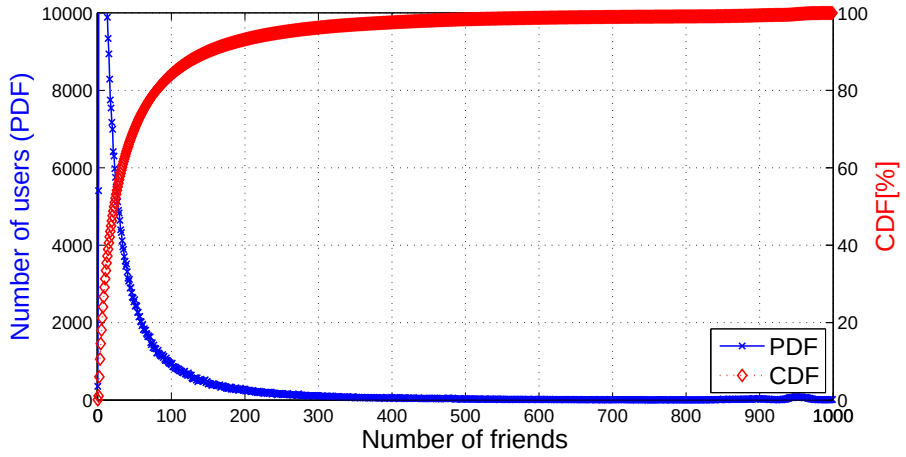


Figure 4.9: Number of friends, per player.

4.5 Related Work

Our work complements the large body of work on the characteristics of human communities and social networks [84, 90], of in-game player activity and behavior [40, 74], and of observing and analyzing platforms for community-creation [123, 134] and media-sharing [37, 133, 153, 176].

The analysis of large-scale online social networks is closest to our work. Previous work in this area has investigated social networks such as FaceBook, Orkut, Flickr [134, 153] and BBO [13, 175]; the systems and social networks underlying media publishing sites such as Youtube and LiveJournal [37, 134, 153]; the loose community built around the the instant messaging network Microsoft Messenger [133]; etc. The social structure of the online game World of Warcraft [66] and the online casual game Fighters Club [162] have also been investigated. We have compared our results with selected results from these previous studies throughout this chapter.

4.6 Summary

The emergence of Online Meta-Gaming Networks pressures system developers and operators to answer questions that require a timely and comprehensive study of their characteristics. In this chapter we have proposed a method for studying the characteristics of OMGNs. Our method is based on the theory of observational studies and employs a repeated cross-sectional design that we have adapted to the specific problems of OMGNs, such as bootstrapping the data collection process and selecting appropriate environment variables to observe. Our method focuses on four classes of environment variables, all of which are important for the design and operation of OMGNs and underlying infras-

tructure: the global network, on gaming activity, on user-generated content, and on social structure. Our method further proposes a high-level, marginal-distribution-based analysis of the observed variables.

We use an implementation of our method to study, that is, observe and analyze, XFire, which is a popular OMGN that services about 20 million users playing over 1,500 games. Our study reveals several interesting observations:

1. OMGN players spend collectively in-game over 100 years hourly;
2. A significant fraction of the players are “hardcore”, having played over 10,000 in-game hours;
3. OMGN members are routinely engaged in the creation and consumption of game-related media, such as screenshots and videos;
4. OMGN members are “social creatures”, having on average over 60 friends.

Chapter 5

Analyzing and Modeling in-NVE Mobility

Networked virtual environments (NVEs), including Massively Multiplayer Online Games (MMOGs) such as World of Warcraft (WoW), already serve tens of millions of users world-wide. Making the current and future NVEs more appealing to their citizens, more scalable to unexpected surges in temporal and spatial popularity, and more efficient in their resource use, depends on understanding user behavioral patterns. Complementing much previous research in the design and tuning of NVE systems, and in particular in collecting, characterizing, and modeling NVE workloads [39, 121, 135], we focus in this chapter on the mobility of NVE citizens. To facilitate the design, validation, and comparison of mobility models and mobility-aware systems, and further motivated by the scarcity of public mobility datasets, we collect for this chapter a large-scale dataset from WoW and share it through the Game Trace Archive [92]. We also conduct a comprehensive, comparative characterization of the mobility of citizens in WoW and other, conceptually different NVE. Furthermore, we also do a high-risk, high-return investigation: motivated by the existence of datasets from networked *real-world* environments (NREs) and by the similarity between some NVEs (e.g., WoW) and NREs, we conduct a comparative analysis of mobility in NVEs and NREs. Based on the observed characteristics of NVEs mobility, we propose a mobility model to generate mobility inputs for NVEs.

Understanding in-NVE mobility can be useful to tune existing designs of NVEs and to innovate in the design of future NVEs. For example, recent advances in server cluster architectures [60] and peer-to-peer overlays [99] need to be validated against mobility workloads and, perhaps, tuned further to specific characteristics, e.g., their structure may need to be tuned to the area visitation characteristics, etc. For cloud-based NVEs such as [101], the load of various servers is strongly correlated with player mobility, due to player interaction [4, 160], cell visitation [60], etc. As has been shown in preliminary work on this topic [131], cloud-based workloads can be much more efficiently supported

if the leasing of resources is in-tune with the workload.

NVE mobility is difficult to understand not only because public datasets are scarce, but also because NVEs cover a broad spectrum of applications. Among the most popular NVEs are MMOGs such as World of Warcraft and user-created NVEs such as Second Life (SL). For WoW, the game developer designs the virtual world to resemble a medieval, albeit fantasy-based, real-world environment. The citizens of WoW need to be highly mobile, to be able to finish quests of the storyline, trade goods, and socialize with the other players. Different from WoW, the virtual world of SL is created by the users themselves; this user-generated content should primarily foster socialization, collaboration, and even supervised learning. We pose and investigate the following research question: *How similar are WoW and SL avatar mobility patterns?* To answer it, we collect a new dataset of WoW mobility traces, and conduct a comprehensive and comparative study across multiple NVE datasets.

The scarcity of NVE mobility datasets is not paralleled by the existence of NRE mobility data. Although few NRE datasets are public, large-scale studies of millions of real-world citizens have appeared in the last decade [87, 177]. A high-risk, high-return idea would be to use these traces in NVE scenarios or even create NVE mobility models based on real world models, for example, when the NVE is by design similar to an NRE for which mobility is well understood, either spatially or w.r.t. the activities that users mostly engage in. WoW and many other NVEs have been designed starting from real-world cities (e.g., medieval cities), and equipped with traditional city-center functions such as meeting and trading. To immerse users, the movement of users in virtual worlds is designed to be as similar as possible to movement in the real world, albeit faster. The high-risk with using NRE traces in NVE scenarios is that the characteristics of NVE and NRE mobility may never match, in spite of the intents of the NVE designers. For example, real-world users do feel the physical effects of movement, including tiredness, legal restrictions, sometimes even cost, etc. The high-return is that the known NRE mobility traces are orders-of-magnitude larger than any of the NVE mobility traces previously reported, and there are many NRE mobility models already developed [118]. Thus, in this chapter we also set to answer the research question *How similar are the characteristics of mobility in NVEs and NREs?* In this chapter, we compare two NVE and two NRE mobility traces, and show evidence that their characteristics share many common patterns. We also point out their main differences, which need future research before NRE mobility related research can be used in NVE studies.

Despite a significant amount of research dedicated for designing NVEs [83, 137, 231], there is little research that validates their approaches against NVEs mobility traces [26, 70] or realistic mobility models. The most commonly used mobility models in NVEs are Random Waypoint mobility model (RWP) and HotSpot model. Although these models can serve as inputs for NVEs architectures, the simulation obtained from NVEs traces and

those simple models can be significantly different [135, 180]. Thus, a realistic mobility model for NVEs is needed to evaluate the performance of NVE designs. In this chapter, we develop SAMOVAR, a Statistical Area-based MObility model for VirtuAl enviRonments. We show through extensive evaluations that SAMOVAR can produce many of the mobility patterns observed in NVEs. Further we show that SAMOVAR can produce similar simulation results compared to traces from WoW and SL.

In summary, our main contributions are listed as follows.

1. We collect a detailed and large-scale mobility dataset from the NVE World of Warcraft (Section 5.2), and share the dataset via the Game Trace Archive [92].

2. We conduct a comprehensive study of human mobility characteristics in both virtual- and real-world environments (Section 5.3). The analysis in this chapter can help NVEs designers better planning resources and provide a base for building a mobility model for simulation.

3. We propose an NVEs mobility model which can model four mobility characteristics: pause duration, velocity, area popularity, and distinct visited areas (Section 5.4).

4. We perform a large amount of simulations to show that the proposed mobility model can reproduce many mobility patterns observed in NVE traces (Section 5.5). We show that the simulation results obtained from our model on a client/server architecture are similar to traces from World of Warcraft and Second Life.

5.1 Background

In this section, we describe the terminology used and the mobility characteristics studied in this chapter.

5.1.1 Terminology

- *Avatars* (players, persons) are the moving entities.

- *Map* in which movement takes place is well modeled by a grid of rectangular, non-overlapping areas.

- *Flight* is a straight-line trip without pause or significant directional change. The "angle model" of Rhee et al. [179] allows for several consecutive straight line trips to be connected into a single flight if the angle between consecutive trips does not change the general direction of the flight.

- *Waypoints* are the positions on a map. An avatar can only pause (stay in a position without movement) in waypoints.
- *Pause duration* is the time spent by an avatar in a waypoint.
- *Velocity* (or movement speed [130], or speed) is the rate of change of position; the average speed is the ratio between the flight length and the time elapsed between the start and the end times of the flight.
- *Visit*, we define a visit to a waypoint only if an avatar pauses in that waypoint.
- *Area popularity* is the popularity of an area defined as the total number of distinct persons visited that area for a specific time duration.
- *Distinct visited areas* is the set of areas that an avatar visits.

5.1.2 Mobility Characteristics

In this chapter, we focus on six mobility characteristics which have been investigated in the past and shown to significantly affect the performance and reliability of NREs. Some of the characteristics have been also shown to have an impact of the performance of NVEs too. These characteristics are:

- **(C1) Long-tail distribution of flight lengths** [87]: human usually travel short distances and occasionally travel long distance.
- **(C2) Long-tail distribution of pause durations** [179].
- **(C3) Heterogenous movement speed** [130]
- **(C4) Skewed popularity of areas** [174, 177]; for example, certain areas of cities are very popular, while others are rarely visited.
- **(C5) Invisible boundary of human movement** [87]: most of the time, people only travel between home and workspace, and around a few preferred locations.
- **(C6) Different personal preferences for areas** [203].

5.2 Datasets

In this section, we introduce the data collection processes and the four datasets used in this chapter.

Dataset	World	Citizens	Space	Time	Granularity
WoW (ours)	Virt.	31,290	4 cites	2w	1s
SL [135]	Virt.	26,714	4 zones	days	10s
GPS [24]	Real	1,366	3 cities	1w	6s
GPS-2 [179]	Real	52	2 campuses	days	30s

Table 5.1: Dataset overview.

5.2.1 Data Collection

We have collected a very large and detailed dataset from a popular virtual world, World of Warcraft (WoW). WoW adopts a sharding architecture with multiple independent realms with same starting scenario. Each realm may have different types of interaction styles: normal, role-playing and player versus player (PVP). We collected 17,583 users' trace from 3 capital cities (Ironforge, Orgrimmar, and Stormwind) of the popular Silvermoon server (normal realm in Europe) and 13,707 users from Stormwind (we call it Stormwind-2 from now on) of the popular Argent-Dawn server (role-playing realm in Europe).

Each virtual citizen of WoW can observe the presence and activity of any other virtual citizen within a radius of about 100 in-game meters. Unlike the real world, the observation range in WoW is not affected by interposing objects such as buildings or other citizens. To collect the WoW dataset, we have developed a tracing script and used it to observe selected cities. Our script logs-in several regular WoW clients and coordinates them to observe a large part of a city. To observe mobility in a city, our client deploys virtual citizens such that their observation areas cumulatively cover a major part of the city. The client uses 6 machines per measurement, each running several WoW clients and collecting their observed data. Due to the availability of machines, which are regular PCs used for coursework at our university, during week days we can only collect data during the night. In total, we have obtained data for 3 complete weekends and about 20 week-day evenings during April and September 2011, resulting in mobility traces for over 30,000 anonymous virtual citizens.

5.2.2 Dataset Description

We use the mobility dataset collected from WoW, and use three other datasets that are collected by others from virtual and real worlds. The characteristics of the datasets are summarized in Table 5.1.

Our dataset, WoW, is large-scale (over 30,000 citizens) and multi-location (4 cities); it was also collected using fine-grained sampling (1 sample every second) over a significant period (several weeks). We have performed typical data sanitation on our datasets by removing un-realistic movements if the movement speed is higher than $100\,m/s$ (probably

caused by avatar teleportation or GPS miscalculation), overall removing less than 1% of the raw data. The SL dataset is collected and released by Liang et al. [135] from Second Life, and includes about 25,000 citizens from 4 zones: Isis, Ross, Pharm and Freebies. The GPS dataset is collected by Bohte and Maat [24] by distributing GPS devices to over 1,300 persons live in 3 cities for transportation research. The GPS-2 dataset is collected by Rhee et al. [179], and contains traces of about 50 persons from two campuses: KAIST and NCSU.

5.3 Characterization Results

In this section, we answer the question *How similar are WoW and SL avatar mobility traces?* and *How similar are virtual and real-world human mobility traces?* To answer this question, we investigate the characteristics (C1)–(C6) (see Section 5.1.2) for WoW, SL and GPS (Section 5.2). We only investigate (C1)–(C3) for GPS-2 due to the relative small sample sizes and the mobility of citizens are limited to campus scenarios.

Where the datasets comprise multiple locations, we analyze both the entire dataset and each location, in turn. Unless otherwise noted, we have obtained similar results for each investigated dataset. To study characteristics among different traces, we look at the basic descriptive statistics, and then use the distribution fitting method (described in Section 5.3.1) to look at the trend and distribution of data. We present here only a selection of representative results.

Our main finding is that the mobility characteristics for the two virtual world (WoW, SL) traces have many similarities. The flight length (C1), pause duration (C2), and area popularity (C4) follow long-tail distributions; avatars' moving velocities are heterogenous (C3), avatars only visit a small portion of virtual cities (C5); and preference to visit only a few, preferred areas does exists (C6). In comparison, for GPS, the flight length is longer; moving speed is less heterogeneous and the personal preference to some areas is higher.

5.3.1 Method for Distribution Fitting

Because virtual worlds may distort the sizes of buildings and the speed of avatars, in comparison with real-world environments, we are interested in study the general trends and distributions of mobility characteristics, besides basic statistics values. For each trace considered in this chapter, we attempt to fit the empirical data corresponding to all the characteristics studied in this chapter with a set of well-known probability distributions that are available in most simulation and experimental toolboxes, namely the the Exponential, the Weibull, the LogNormal, the Gamma, the Normal, and the General Pareto distributions.

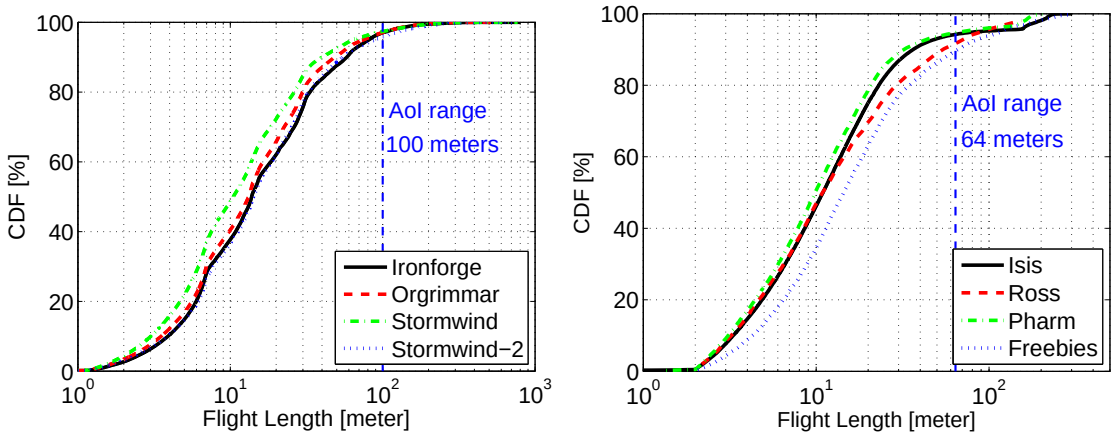


Figure 5.1: Flight length distribution of (left) the WoW dataset, and (right) the SL dataset (horizontal logscale).

The fitting is performed using *maximum likelihood estimation* (MLE), which determines for a distribution the parameters that lead to the best fit with given empirical data. Then, we use a method for assessing the *goodness-of-fit* (GoF) that has been shown to have good results for large datasets in distributed systems studies [125]. In this method, the results of MLE fitting are tested using a *goodness-of-fit* (GoF) procedure that combines the Kolmogorov-Smirnov (KS) and the Anderson-Darling (AD) GoF tests. Using both of these tests provides a more robust GoF test than using any of the KS and AD tests individually, since the KS test is more sensitive to the center of distributions and the AD test is more sensitive to the tail. The method uses 0.05 as the significance level for the p-value, below which the null hypothesis that the fitted distribution represents the empirical data is rejected. The p-value used by this method is the average of 1,000 p-values, each of which is calculated by randomly selecting 30 samples from the empirical data and applying the GoF tests to the selected data. The distribution which passes the GoF test and has the lowest D value, the largest gap between the empirical cumulative distribution function (CDF) and fitted CDF, is selected as the best fit. We call the probability distribution of data is "long-tail" if the tail of the probability distribution is longer than the fitted exponential distribution of data.

5.3.2 Flight Length (C1)

Figure 5.1 shows the cumulative distribution function (CDF) of the flight lengths of WoW (left) and SL (right). The flight lengths of WoW traces are long-tail distributed and the flight lengths for all four cities are similar. The mean values of flight lengths in the four virtual cities are around 20 to 25 meters. Most (about 85% to 90%) of the flights are shorter than the area of interest (AoI) range (100 meters) of WoW. For the SL traces, the

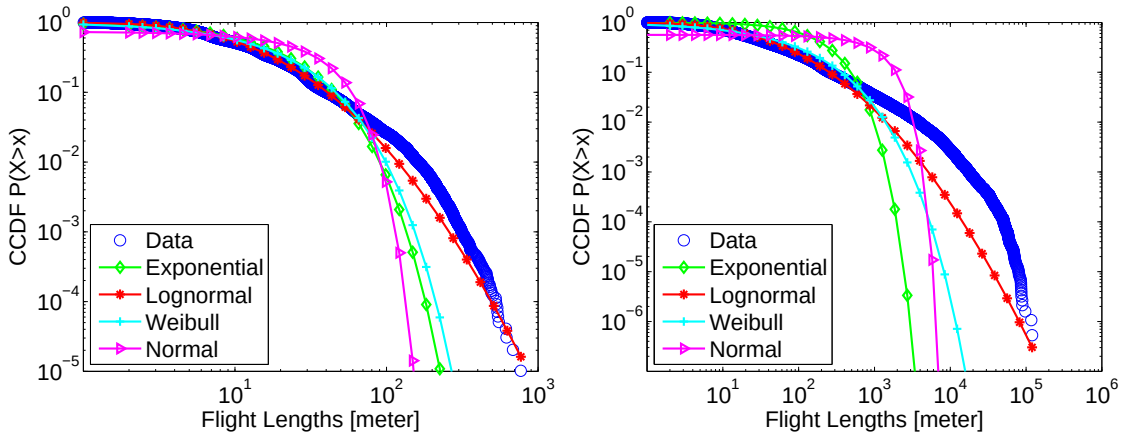


Figure 5.2: Distribution fitting of (left) WoW dataset, and (right) the GPS dataset (all axes logscale).

mean values of flight lengths in the four zones are around 19 to 29 meters. Most (80% to 90%) of the flight lengths are shorter than the AoI range (64 meters). This may suggest that when avatars travel in virtual worlds, most of them travel within the boundary of AoI, and occasionally avatars travel long distances.

For the two real world datasets: the mean value of flight lengths of GPS is $215\,m$, while the mean values of flight lengths for KAIST and NCSU are $61\,m$ and $71\,m$, respectively. The flight lengths of the two real world datasets have longer tail than the two virtual world datasets: the 99% percentiles for the two virtual world datasets are about $150\,m$ to $230\,m$, while the 99% percentiles for the two real world datasets are about $600\,m$ to $4,000\,m$.

Figure 5.2 depicts the results of fitting for Stormwind, WoW and GPS. The vertical axis shows the complementary cumulative distribution function (CCDF) of the flight lengths, in logarithmic scale (Note that the scales of the two figures are different). We find that the best fit for Stormwind is LogNormal distribution (mean $\mu = 2.4$ deviation $\sigma = 1$). For the GPS data, the best fit is a LogNormal distribution ($\mu = 3.4$ $\sigma = 1.65$) (the distribution fitting diverge a bit when the flight lengths are higher than $1,000\,m$). The flight lengths distributions for the two virtual world datasets (WoW and SL) and the two real world datasets (GPS and GPS-2) follow long-tail distributions, and all of them can be best fitted using the LogNormal distribution.

5.3.3 Pause Duration (C2)

Figure 5.3 shows the pause durations distribution of the WoW and the SL datasets. Overall, the pause durations of both datasets are long-tail, about 80% of the pause durations of WoW is shorter than 30 seconds. The pause duration of Stormwind is slightly lower than

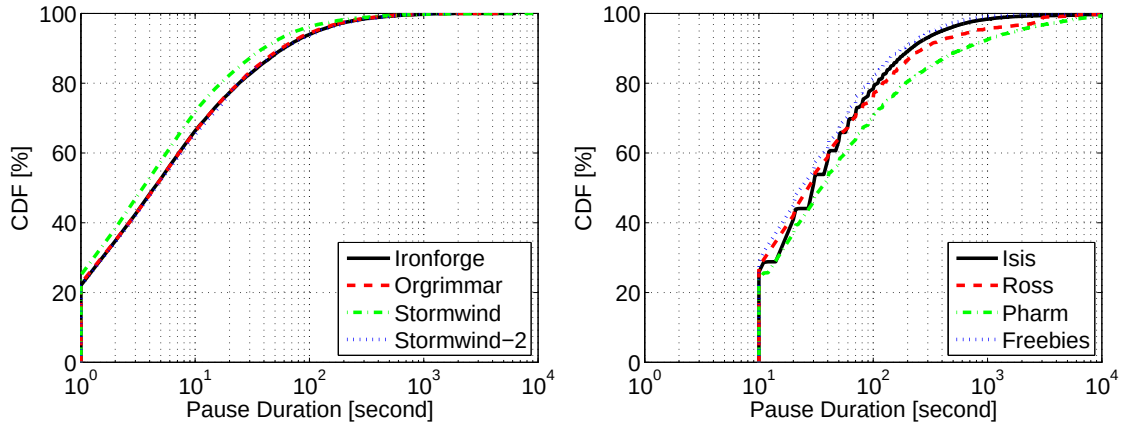


Figure 5.3: Pause duration distribution of (left) the WoW dataset, and (right) the SL dataset (horizontal logscale).

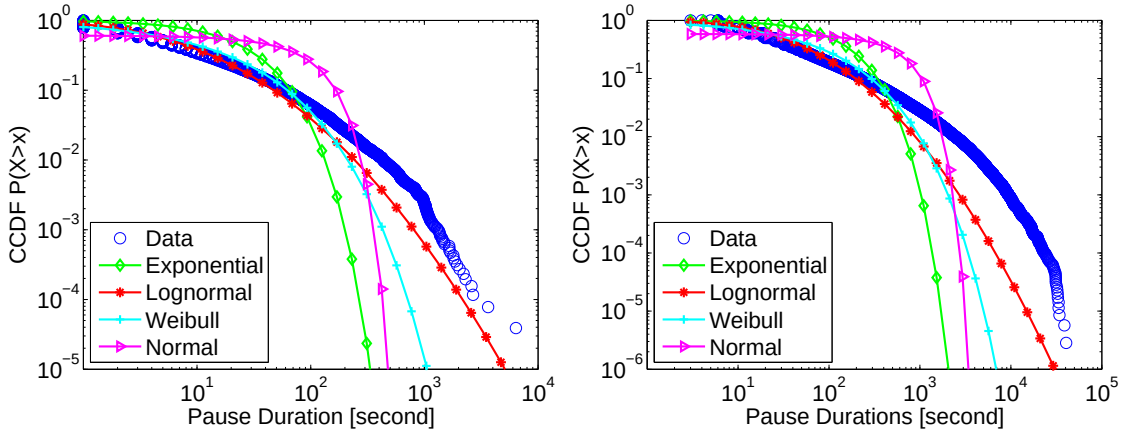


Figure 5.4: Distribution fitting of (left) WoW dataset, and (right) the GPS dataset. (all axes logscale.)

the other three cities, while the other three have very similar distributions. For the SL dataset, about 70% to 80% of the pause durations is shorter than 100 seconds. The pause durations for the Pharm zone is higher than the other because the main activities of that zone is camping (staying in the same location). The pause durations of the WoW datasets are significantly shorter than the SL datasets. The difference of the pause durations for the two datasets may be caused by the design of the two NVEs: SL focuses more on social aspects, while WoW is more task-oriented and the interactivity between players is more frequent.

For the real-world datasets, the pause durations for those datasets are long-tail too. The average pause duration of the GPS dataset is about 2.5 minutes, and the 99% percentile of pause durations is about 40 minutes. For the GPS−2 dataset, the mean values

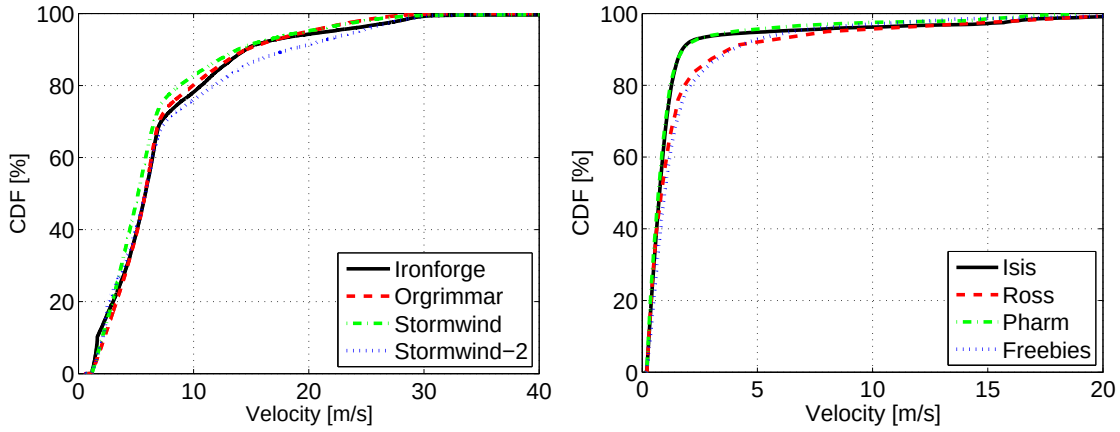


Figure 5.5: Velocity distribution of (left) the WoW dataset, and (right) the SL dataset.

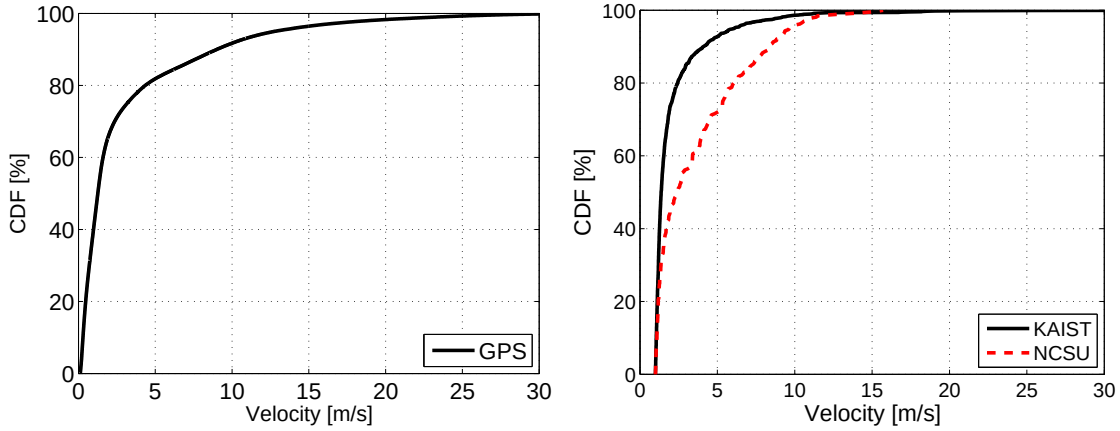


Figure 5.6: Velocity distribution of (left) the GPS dataset, and (right) the GPS−2 dataset.

range from 5.5 to 6 minutes, and the 99% percentiles are around 1.5 hours.

Figure 5.4 depicts the results of distribution fitting for the Stormwind, WoW and GPS. The pause durations observed in Stormwind can be best modeled using the Log-Normal distribution ($\mu = 1.63$ $\sigma = 1.45$). The fitting result for the GPS dataset is the LogNormal distribution ($\mu = 3.41$ $\sigma = 1.44$). In summary, the pause durations distributions for the two virtual world datasets: WoW and SL and the two real world datasets: GPS and GPS−2 follow long-tail distributions, and all of them can be best fitted using the LogNormal distribution.

5.3.4 Velocity (C3)

Figure 5.5 shows the velocity distributions for the WoW (left) and the SL (right) dataset. Overall, the speed distributions of the two virtual worlds are heterogeneous. For the WoW

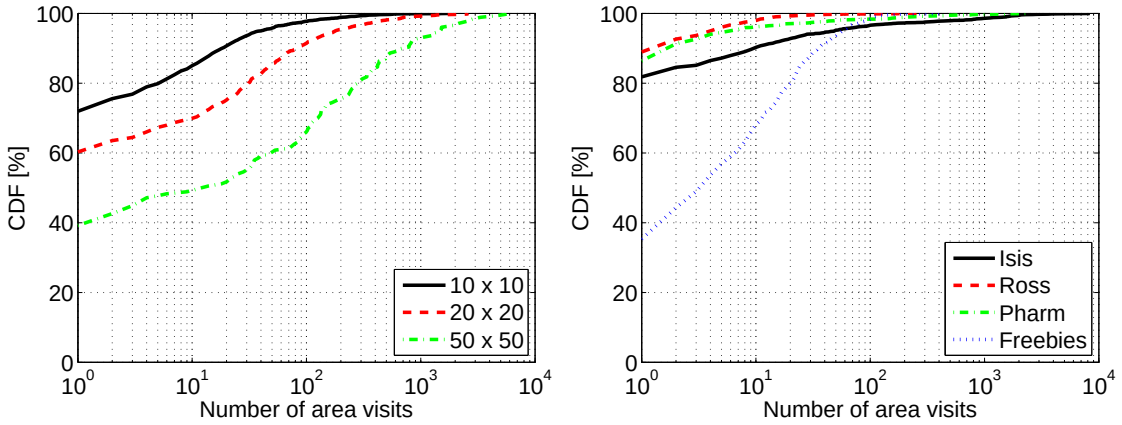


Figure 5.7: Number of area visits of (left) the WoW dataset, and (right) the SL dataset (horizontal logscale).

dataset, the average velocities are about 7 to 8 m/s, and the 99% percentiles are about 27 to 29. For the SL dataset, most of the movement speeds are lower than 1.5 m/s. The velocities across different virtual cities are different: the speeds for Isis and Pharm are slightly slower than that of Ross and Freebies,

Figure 5.6 shows the velocity distributions for the GPS and GPS-2 dataset. The movement speeds are heterogeneous. For the GPS dataset, the mean and median velocities are 1.3 m/s and 3.1 m/s, respectively. For the GPS2 dataset, the movement speeds for NCSU are higher than that of KAIST.

5.3.5 Area Popularity (C4)

To investigate area popularity, we first split the environments into rectangular grids, where each cell is an area. Rectangular grids are convenient for setting up simulation scenarios and may enable fair comparison between different city scenarios. We explore different values for the size of each area, which is the parameter of the splitting procedure; we split maps into areas of $10\,m \times 10\,m$ up to $50\,m \times 50\,m$. For each area size, we quantify the popularity of the resulting areas using two main indicators: the *number of area visits*, defined for each area as the number of pauses observed in that area; and the *number of area visitors*, defined for each area as the number of unique visitors paused in that area. Intuitively, the former indicator quantifies the total traffic through an area well, whereas the latter does not account for returning visitors.

Number of area visits: Figure 5.7 (left) shows the number of area visits of a 1 day trace from Ironforge, by splitting the map into areas of $10\,m \times 10\,m$, $20\,m \times 20\,m$, and $50\,m \times 50\,m$. The visitation count increases with the increasing size of the areas. Large portions of the map are not visited at all, about 75% of the $10\,m \times 10\,m$ areas are

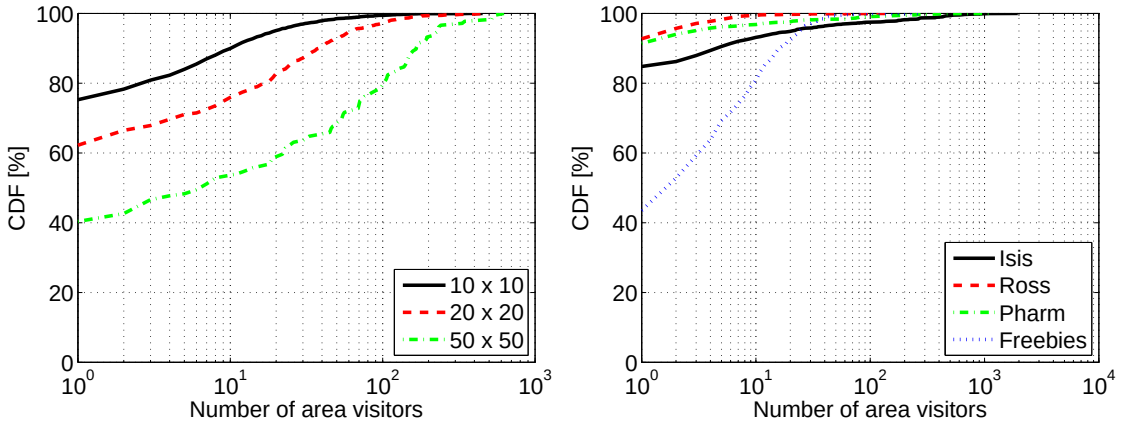


Figure 5.8: Number of area visitors of (left) the WoW dataset, and (right) the SL dataset.

not visited once, and about 40% of the $50\,m \times 50\,m$ areas are not visited. The visitation count is long-tail; for $10\,m \times 10\,m$ areas, the 85% percentile is 10 while the maximal value is about 1,921. Figure 5.7 (right) shows the results for SL, when splitting the map into areas of $10\,m \times 10\,m$ size. Similarly to WoW, large parts of the map are not visited, 3 out of 4 zones have 80% unvisited areas; and the distribution of the number of area visits of SL is long-tail. The number of area visits for GPS is long-tail too, when it is partitioned into areas of $10\,m \times 10\,m$, over 99% of the areas is not visited at all, while the most popular area is visited about 900 times.

Number of area visitors: Figure 5.8 shows the number of area visitors of a 1 day trace from Ironforge and SL. The number of area visitors is smaller than the number of area visits, but it is long-tail too. Figure 5.8 (left) shows the number of visitors for Ironforge, by splitting the map into areas of $10\,m \times 10\,m$, $20\,m \times 20\,m$, and $50\,m \times 50\,m$. For $10\,m \times 10\,m$ areas, the 85% percentile is 6 while the maximal value is about 453. Figure 5.8 (right) shows the results for SL, when splitting the map into areas of $10\,m \times 10\,m$ size. Similar to WoW, large parts of the maps are not visited, and the distributions of the number of area visitors for SL are long-tail. For the GPS dataset, when it is partitioned into areas of $10\,m \times 10\,m$, the most popular areas is visited by 80 persons, and when it is partitioned into areas of $50\,m \times 50\,m$, the most popular area is visited by 173 persons. The distributions of the number of area visitors for the WoW, SL, and GPS datasets are long-tail.

5.3.6 Invisible Movement Boundary (C5)

We now look at the invisible movement boundary, that is, the phenomenon that humans tend to travel mostly within a fixed and reduced set of locations around home and office (see Section 5.1.2). We find that *the invisible movement boundary is present in both real*

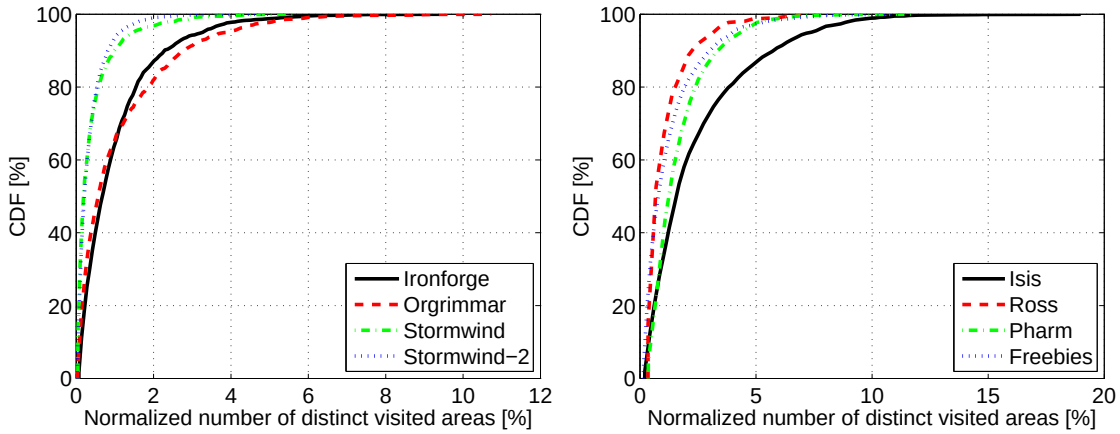


Figure 5.9: Normalized number of distinct visited areas of (left) the WoW dataset, and (right) the SL dataset.

and virtual worlds. To quantify the boundary, we use the proxy metric normalized number of distinct visited areas, measured per person. Figure 5.9 shows the number of distinct areas, normalized by the total number of visited areas per map. The higher this value is, the higher the probability of avatars meeting each other. This metric can be useful for modeling mobility: when generating waypoints on maps, the model can limit the avatar to visit only a small subset of waypoints. As Figure 5.9 shows, for WoW and SL, the normalized number of distinctive areas is low. Most (about 95%) of the avatars visit less than 5% of the visited areas; only a few persons visit more than 10% of the visited areas. In average, each avatar visits about 0.4 to 1% of the areas in the WoW dataset; and in SL, each avatar visits about 1.2 to 2% of the areas.

For the GPS dataset, most (about 95%) of the avatars visit less than 0.5% percent of the visited areas. We attribute the significantly lower values for the GPS data to the fact that the real world cities are much bigger than the virtual world cities: the GPS dataset cover a map about $30\,km \times 30\,km$, while the largest virtual cities in the WoW and SL is smaller than $2\,km \times 2\,km$. For the empirical distributions: the normalized number of distinct visited areas for Ironforge can be fitted best by the Weibull distribution (scale $a = 2.34$, shape $b = 1.99$), and the best fit for the GPS dataset is the Weibull distribution ($a = 0.12$, $b = 2$). However, the SL traces can be better modeled using the LogNormal distribution.

5.3.7 Personal Preference in Area Visitation (C6)

In SL, some avatars like to visit the same group of persons [221]; and real-world citizens have strong preferences for different areas [24]. We study the personal preferences of virtual and real- world in this section. For each of the area the avatars visited, we count

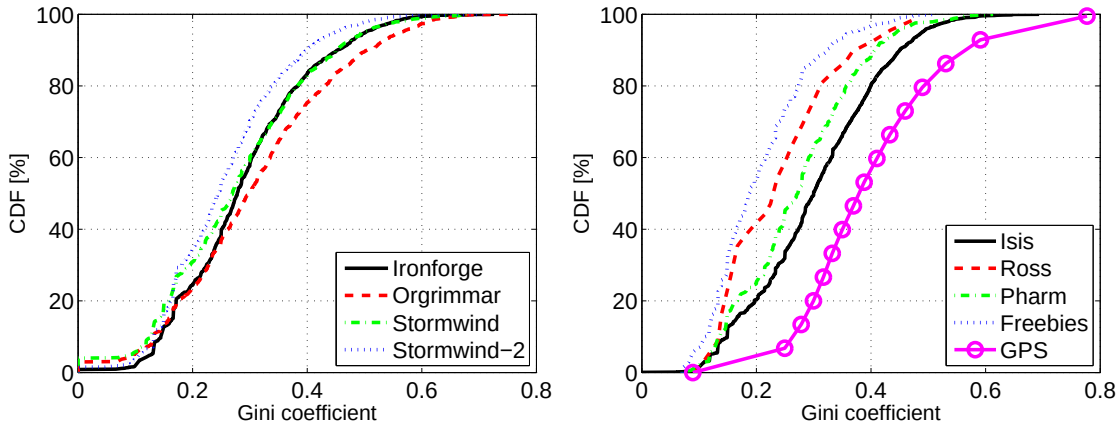


Figure 5.10: Gini coefficient of personal preference weight (left) the WoW dataset, and (right) the SL dataset and GPS dataset.

the number of time the avatar visited that area as *personal preference weight*. Then for each person, we calculate the Gini coefficient (also called Gini index) of the personal preference weight. It is used to quantify the inequality of personal preference (a value of 1 means very unequal, whereas 0 means perfectly equal).

Figure 5.10 shows the Gini coefficient distribution of each person for WoW, SL, and GPS. For this figure, we remove the persons that visit less than 5 areas (the result is similar without removal). In general, the two virtual world datasets have similar Gini coefficient distributions: most (80% to 95%) of the Gini coefficients are lower than 0.4. For the GPS dataset, about 40% are higher than 0.4. The probability distribution functions of the Gini coefficients for all datasets are bell-shape curves, can be modeled using the Weibull distribution.

The Gini coefficients of the personal weights in GPS dataset is higher than in the two virtual world datasets. This may suggest that the personal preference for areas is stronger in real-world environments than in virtual worlds, and has higher predictability in real-world human mobility than in virtual-world avatar mobility. As possible explanations, we point to the higher rate of movement, to the less restrictive of movement, and to other lower penalties for movement (legal restrictions, cost, etc.) in virtual vs real-world mobility.

5.4 SAMOVAR: An NVE Mobility Model

In this section we introduce SAMOVAR, the Statistical Area-based MObility model for VirtuAl enviRonments. SAMOVAR models microscopic, individual mobility in virtual environments, but can be used to generate macroscopic, population-wide mobility traces. The core of our model is a generative process, in the sense that the model incorporates

the notion of time and predicts what sequence of movements would be taken by each individual, so that movement traces are generated.

SAMOVAR consists of three parts, the *Characteristics modeling*, the *Map generation*, and the *Walking*. The characteristics modeling models each characteristic of an NVE trace using empirical modeling, the procedure is described in Section 5.3.1. The *Map generation* procedure generates a map with waypoints and paths between waypoints. The *Walking* part of SAMOVAR determines how avatars walk between waypoints.

5.4.1 Map Generation in SAMOVAR

In SAMOVAR, a map consists of waypoints and paths. Avatars only travel along the generated paths. The generation of SAMOVAR mimics the traffic network path of virtual world that some places of interest are well connected, in order to trave from a less popular zone to a popular zone far away, it is common to go to a nearby popular zone first and then using the transportation service of that popular zone to go to zones far away. The map is generated in SAMOVAR in six steps:

1. Partition the map into $10\,m \times 10\,m$ areas. Randomly select n area distributed across the map. The center point of each selected area is used as a waypoint.

2. Assign a *Popularity weight* (p) according the LogNormal distribution to each waypoint obtained in last step, then classify each waypoint to one of m levels according to p. Each waypoint is classified by a level, the level of a waypoint is determined by p. We use logarithmic binning to determine the level of a waypoint, that is, each waypoint level contains a fixed multiple of the number of waypoints included in the previous level.

3. Connect the heaviest waypoint with each other.

4. Connect each waypoint with its closest waypoint of higher level.

5. Connect same-level waypoints with each other, if they are connected to the same higher level waypoint.

6. Connect waypoints with each other if their distance is lower than a threshold value r.

We illustrate the map operation in SAMOVAR in Figure 5.11. Each waypoint has a level between 1 (highest) and 3 (lowest). To travel from waypoint A to waypoint E, an avatar would first go from waypoint A to waypoint B, then to C, etc. Waypoints F and G subordinate to different waypoints but are directly connected, because their distance is smaller than r. The generation of map of SAMOVAR is inspired by HTM [96]. Different

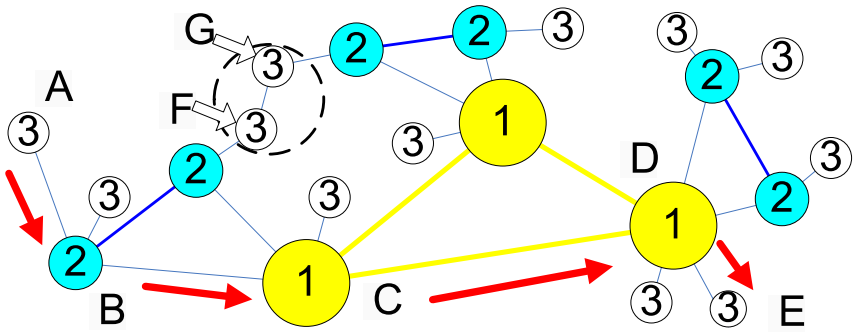


Figure 5.11: A generated map.

from HTM, which assigns a same weight for waypoints of the same level, SAMOVAR assigns LogNormal distributed weighs to waypoints. SAMOVAR connects waypoints with each other if their distance is lower than r while HTM does not. The traffic network of SAMOVAR is more realistic.

5.4.2 Walking Paths Generation in SAMOVAR

Walking paths in SAMOVAR are generated via simulation. The generation process includes the *path generation* and the *path traveling* processes. The generation process assigns to each avatar a limited number of waypoints to visit during the simulation. And each avatar will have different visitation frequencies to different waypoints. The traveling process determines how an avatar will travel to the assigned waypoints. We describe *path generation* and *path traveling*, in turn.

Generation process

The path generation process is based on an observation of the NVEs mobility traces that many avatars only visit a small amount of areas of the whole city (see C5 in Section 5.3.6). Besides, different avatars like to visit different areas of a virtual-world city: auction houses, profession trainers etc. The generation process is described as follows:

1. Assign to each avatar the number of waypoints this avatar can visit, k, sampled from a LogNormal distribution.

2. Assign to each person a start waypoint. We explore two ways to assign the start waypoint for each avatar, *SAMOVAR-U* and *SAMOVAR-W*. *SAMOVAR-U* assigns the start waypoint randomly while *SAMOVAR-W* assigns the start waypoint according to the weight of waypoint established by the map generation part of SAMOVAR. Higher weight waypoints have a higher number of avatars.

Characteristic	Distribution (Parameters)	D value	KS AD
Pause duration Δt	LogNormal (1.82, 1.57)	0.12	0.322 0.613
Velocity v	LogNormal (1.82, 0.67)	0.09	0.353 0.570
Popularity weight p	LogNormal (1.65, 1.30)	0.14	0.310 0.672
Number of visited areas k	LogNormal (2.27, 1.02)	0.06	0.416 0.688

Table 5.2: Fitting results for the `Ironforge` trace.

3. For each avatar, iteratively add the waypoints neighboring the already assigned waypoints, until the number of assigned waypoints reaches k (step 1). The complete set of waypoints assigned to an avatar is the *visit set* of that avatar.

4. For each avatar, assign a personal preference w for the waypoints in the *visit set* of the avatar, as a per-waypoint *personal weight*. The personal preference weight is sampled from Zipf distribution ($\theta = 1$). The personal weights are randomly assigned to the waypoints in *visit set*. The reason why we assign a personal weight to each waypoint lies that avatars do have different visitation frequencies to different areas (see Section 5.3.7). Currently we model the personal weight to follow a Zipf distribution ($\theta = 1$), we plan to investigate more on the personal preference modeling.

Path traveling process

Traveling occurs for an avatar only within the *personal waypoint graph*, that is, the subgraph of the map that spans only the waypoints in the *visit set* of the avatar, and includes all the paths between them (generated in map generation part). The path traveling process has the following steps:

1. Each avatar starts in the start waypoint assigned in *generation process* at time $t = 0$.

2. When not traveling or pausing, a person will change location by first selecting a waypoint to visit, from the waypoints in the visit set and according to the personal weight of each eligible waypoint. Then, the avatar travels using the shortest path in the *personal waypoint graph* with a speed v sampled from a LogNormal distribution.

3. After reaching the selected waypoint, each person pauses for Δt time units, which is drawn from a LogNormal distribution.

5.5 Validation and Application to NVEs

In this section, we validate SAMOVAR (SAMOVAR-U and SAMOVAR-W) against NVE traces using four mobility characteristics: pause duration, velocity, area popular-

Characteristic	Distribution (Parameters)	D value	KS AD
Pause duration Δt	LogNormal (3.50, 1.22)	0.16	0.189 0.578
Velocity v	LogNormal (0.82, 0.74)	0.14	0.220 0.323
Popularity weight p	LogNormal (2.51, 1.47)	0.07	0.398 0.636
Number of visited areas k	LogNormal (1.78, 0.88)	0.10	0.364 0.715

Table 5.3: Fitting results for the `Freebies` trace.

Name	Ironforge	Freebies
Map size	$791\,m \times 528\,m$	$256\,m \times 256\,m$
Number of avatars	1,302	3153
Number of waypoints n	1,378	651
Number of levels m	9	9
Connection range r	$20\,m$	$20\,m$
Area of Interest range R	$100\,m$	$64\,m$

Table 5.4: Parameters for simulation.

ity, and distinct visited areas. We show that SAMOVAR can reproduce all four characteristics while random waypoint and HotSpot model fail. Then, to show the practicality of SAMOVAR, we use the mobility traces produced by SAMOVAR, RWP, and HotSpot to drive the simulation of NVEs architecture. The simulation results produced by SAMOVAR are close to the results obtained by using traces from two NVEs: WoW and SL, while the results obtained from RWP and HotSpot are significantly different. We conclude that *SAMOVAR is a valid human mobility model which can be used for NVE evaluations*.

5.5.1 Experimental setup

For this study, we use two NVE traces: `Ironforge` and `Freebies`. `Ironforge` is a one day trace we collect from the popular city `Ironforge` of WoW (see Section 5.2.1). In total, the `Ironforge` trace contains 1,302 avatars' movement trajectories. `Freebies` is a one day trace collected by [135] from the `Freebies` zone of SL, and contains movement information of 3,153 avatars. The fitting results for the two traces are listed in Table 5.2 and 5.3.

To compare with the `Ironforge` trace, the simulation is conducted in a $791\,m \times 528\,m$ map, this is the same map size as the Ironforge city. We set the number of waypoints of SAMOVAR to be 1,378, because there are 1,378 areas are visited in `Ironforge`. The comparison with `Freebies` traces is configured similarly. All the default simulation parameters are listed in Table 5.4.

We describe the mobility models that we compare with in turn. In the random waypoint (RWP) model, each avatar will randomly select a destination in the simulation area and goes to destination along the straight line connecting current waypoint and destination; upon arrival, the avatar will pause for Δt which is uniformly distributed between $[1, 60]$. The HotSpot models can be viewed as a weighted random waypoint model, in

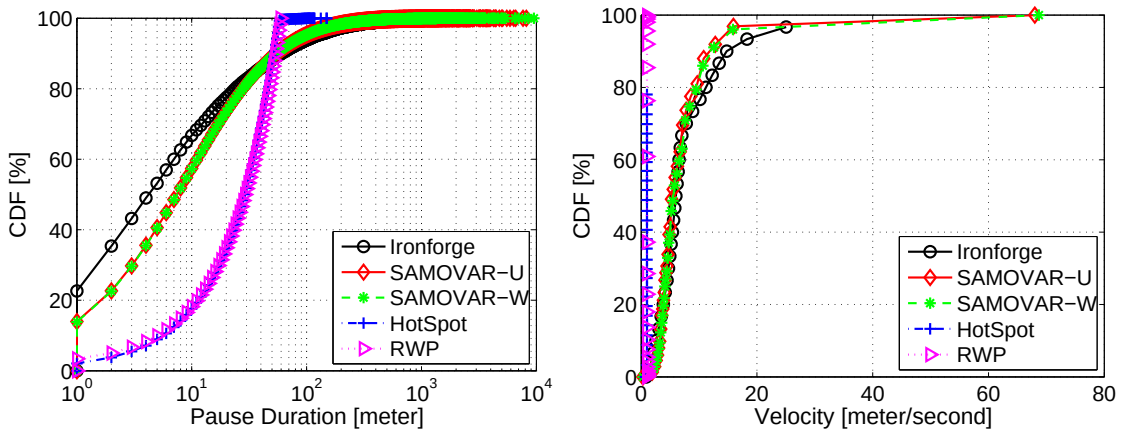


Figure 5.12: SAMOVAR-U: Pause duration (Left) and Velocity (Right).

which the probability to go to a waypoint is proportional to its weight, we assigned a *popularity weight* to each waypoint. The *popularity weight* is assigned using the same distribution as SAMOVAR. For RWP and HotSpot models, the velocities for traveling is $1\,m/s$.

5.5.2 Validation

For validation, we record all the traces generated by each mobility model, and extract four characteristics from the generated traces: pause duration, velocity, popularity, and number of distinct visited areas. After extracting the four characteristics, we compare the distribution of generated characteristics against the distribution of characteristics of `Ironforge`. For the three properties that are directly generated from empirical model, pause durations, velocity, number of distinct visited areas: the generated properties of SAMOVAR-U and SAMOVAR-W are very close to the original data; for pause durations, the distributions generated by SAMOVAR are slightly higher (10%) than the real trace; for area popularity, the distribution for both models is a bit lower (5% to 10%) than the values in the `Ironforge` trace. For the `Freebies` trace, we obtain similar results that the fitting to empirical data is much better than the alternatives.

Figure 5.12 (left) shows the pause duration for `Ironforge`, SAMOVAR-U, SAMOVAR-W, RWP and HotSpot models. The distribution of pause durations generated by SAMOVAR is a bit (5% to 10%) higher than `Ironforge` when the pause durations are lower than 20 seconds, after that SAMOVAR matches well with the trace. As Figure 5.12 (right) shows, the velocities predicted by SAMOVAR matches closely to real trace when the speed is lower than $8\,m/s$, after that the velocities predicted by SAMOVAR is slightly slower (about 5%) than real trace.

Figure 5.13 (left) shows the area popularity distributions for `Ironforge`,

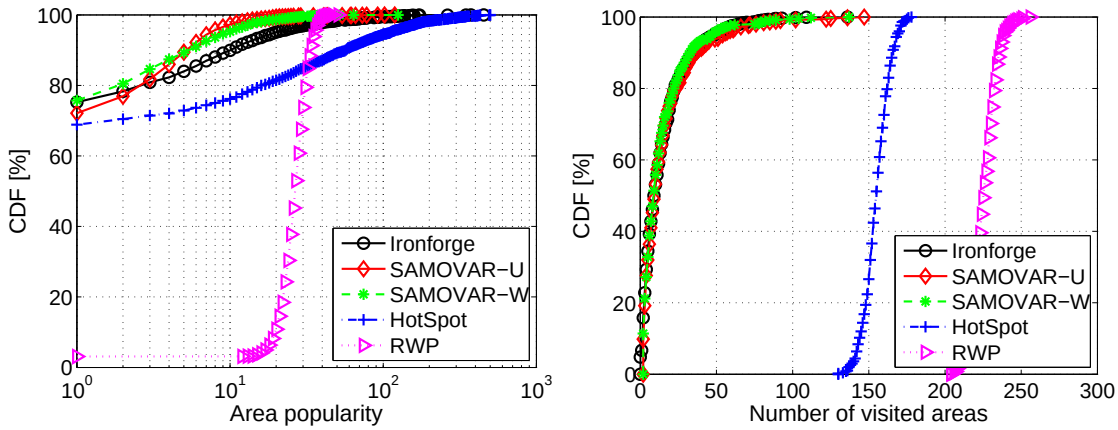


Figure 5.13: SAMOVAR-U: Popularity (Left) and Number of Visited Areas (Right).

SAMOVAR-U, SAMOVAR-W, RWP and HotSpot models. The area popularity distribution predicted by SAMOVAR is a bit (about 10%) higher than `Ironforge` trace when the area popularity is lower than 20. After that the area popularity predicted by SAMOVAR-U and SAMOVAR-W matches the trace well. Comparing to SAMOVAR-U, SAMOVAR-W has a longer tail, this will lead to more avatars concentrate in popular areas. The area popularity distribution predicted by SAMOVAR is much better than the results obtained using RWP and HotSpot model. Figure 5.13 (right) shows the distribution of number of distinct visited areas per avatar, the distribution curve of SAMOVAR is very close to the curve of `Ironforge`. Figure 5.12 and Figure 5.13 show that the map generation and walking procedure of SAMOVAR does not distort the characteristics which are modeled explicitly.

5.5.3 Application to NVEs

To show the practicality of SAMOVAR, we use the traces generated by SAMOVAR-U, SAMOVAR-W, RWP, and HotSpot models to drive the simulation of NVEs, and compare the results obtained using real traces: `Ironforge` from WoW and `Freebies` from SL. As SAMOVAR, RWP and HotSpot are mobility models which does not model session behaviors (when a player is online or not); to enable comparison with real traces, for each of the simulated avatar with id i, we pick an avatar with the same id from the real trace, and use the avatar's session behavior as the behavior of the simulated avatar. The SAMOVAR can be easily integrated with session behavior model such as [210], to enable capturing both the mobility patterns and session behaviors of avatars.

We adopt client/server (C/S) architecture as the NVE architecture, because it is the most commonly adopted architecture in industry, and it is also commonly used as a baseline for comparison with different architectures. In C/S architecture, a central server is

responsible for the simulated virtual world, informing clients about events via network communication. The server needs to inform all clients of any position change event within the clients' circular view of radius R (are of interest). We set the range R to be $100m$ for `Ironforge`, as it is the area of interest (AoI) range of WoW. To mimic the environment of SL, for `Freebies` trace, we set R to be $64m$ (the AoI range of SL) and the simulation area to be $256m \times 256m$ (the size of that zone). The movement trajectories in `Freebies` are sampled every 10 seconds. For each avatar in `Freebies` we interpolate the movement trajectory to drive the simulation. For each experiment, we count the number of messages sent by the server to clients and normalize the message count by dividing the value by the message count obtained using real traces.

Figure 5.14 (left) shows the normalized message count (NMC) with increasing number of players for `Ironforge`, RWP, HotSpot, SAMOVAR-U and SAMOVAR-W. The results obtained by using the RWP and the HotSpot models predict higher message counts than the real trace, especially when the number of avatars is large (120% more messages when there are 1,000 avatars). For SAMOVAR, the message count predicted by SAMOVAR-U and SAMOVAR-R is very close to `Ironforge`. The maximal gap between SAMOVAR-U and the real trace is only about 15%. As expected, SAMOVAR-U predicts a lower message count than SAMOVAR-W, and the message count predicted by SAMOVAR-U is closer to real trace than SAMOVAR-W.

For the `Freebies` trace, as Figure 5.14 (right) shows, for the HotSpot model, the predicted message counts are about 4 to 5 times higher than the real trace. The RWP model predict slightly lower messages count than the HotSpot model, but its predictions are highly inaccurate too (about 4 to 5 times higher). For SAMOVAR-W, when the number of avatars is lower than 600, the NMC are about 1.8 to 1.4, but NMC quickly drops to 1.1 when the number of avatars is 800. For SAMOVAR-U, the simulation results made by it are closer to real trace than the others. When the number of avatars is lower than 300, the NMC ranges from 1.5 to 1.15. When the number of avatars is larger, SAMOVAR-U matches the real trace well, especially when number of avatars is higher than 800, the simulation results are only 5% different from real traces.

5.6 Related Work

In this section, we compare our work with research on workload characterization and mobility model of NVEs.

Workload Characterization

Much of the prior work [39,44,74,121,173] has focused on network measurement, online population, and session behavior. Since the late-2000s, several studies [128, 135, 221]

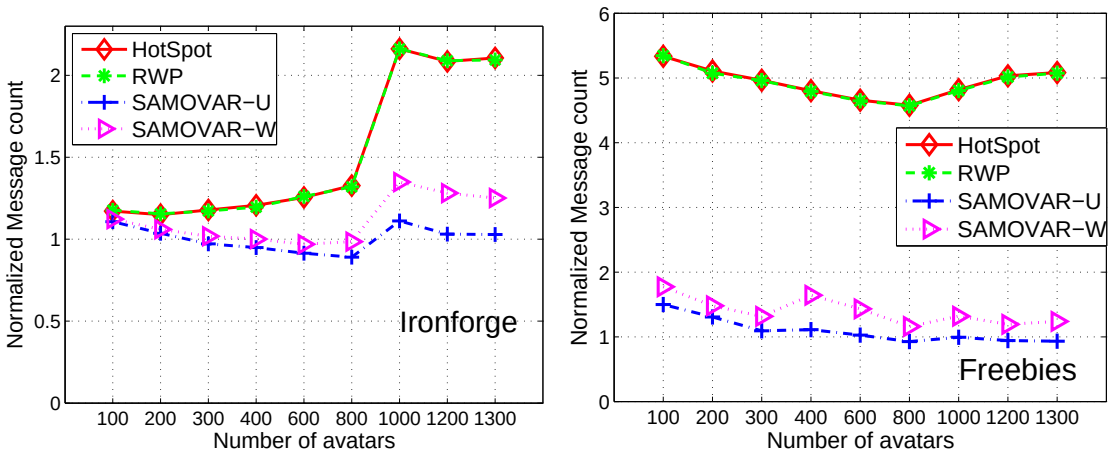


Figure 5.14: Simulation results for C/S architecture.

collect and analyze mobility traces of NVEs. We compare our work with these, in the following.

Closest to our work, Liang et al. [135] collect trace from SL, analyze the session behavior, contact patterns, and mobility patterns, in SL. For mobility patterns, they find that the number of visits to different cells of a region is skewed (C4), accumulated pause duration of avatars stay inside a cell is skewed (C2), and total travel distance of avatars is skewed (similar to C1). Our studies complement each other. We study the five characteristics (C1)–(C6) of two virtual world and two real world mobility traces, to find the similarity and difference of mobility between virtual- and real- world.

Pittman and GauthierDickey [174] analyze traces of two NVEs: WoW and Warhammer; they find that the popularity of different areas is skewed (C4), which they model using the Weibull distribution, and the durations each player stays in different zones vary. Varvello et al. [221] find that in SL, the popularity of zones is skewed (C4), and about half of the players form groups of good friends who meet frequently at the same locations (similar to C6). In contrast to [221], Miller and Crowcroft [149] find that, in their observation scenario of WoW battleground, most movement is individual rather than group-based. La and Michiardi [128] investigate (C4) and mobile communication related metrics such as inter-contact time, using traces collected from SL, and use the traces to evaluate the performance of wireless network protocols.

NVE Mobility Model

Mobility models for NVEs are rare [111, 135, 209]. The most commonly used mobility models used in NVEs are the random walk (RW), the random waypoint (RWP), and the HotSpot model. In the RW model, an avatar randomly chooses a direction and a speed to travel, each movement in RW occurs in predefined time interval or movement distance. In

the random waypoint model, each avatar randomly chooses a waypoint in the simulated map, and goes to the waypoint using a predefined speed. Different from RW, in RWP, if an avatar has reached a waypoint, the avatar will pause for a duration that is sampled from some distributions. The HotSpot model can be viewed as a weighted random waypoint model. In HotSpot model, a number of waypoints are randomly selected and each waypoint is assigned a weight, and the probability to go to a waypoint is proportion to its weight. Pittman and GauthierDickey [174] propose that the weight of waypoints follows a Weibull distribution. SAMOVAR models pause duration, velocity, area popularity, and distinct visited areas, while RW, RWP and HotSpots do not.

There are some workload models proposed for First Person Shooter (FPS) games. The Networked Game Mobility Model (NGMM) [214] is a variation of the HotSpot model which adds perturbations to movement paths and waypoints. Others [19, 220] use artificial intelligence (AI) players to generate workloads for FPS games. Using AI players to drive simulation is computationally intensive; thus, it is difficult to conduct large-scale experiments. Besides, the major user behaviors in NVEs are socializing, trading, finishing quests etc; instead of constantly moving and shooting as in FPS games. SAMOVAR does not model the mobility patterns in fighting scenarios, but in more real-life activities.

5.7 Summary

Understanding the characteristics of mobility is important for the design and tuning of NVEs, but has been hampered until now by the lack of datasets and of comparative studies. In this chapter, we collect and characterize some mobility traces in NVEs; and we have extended the comparison to also include networked real-world environment (NREs). Further, we developed a NVEs mobility model which can generate realistic mobility traces. We have collected detailed position information of virtual world mobility traces from World of Warcraft (WoW). Using our traces, the public Second Life (SL) traces, and two NRE traces collected by others and kindly shared with us, we have conducted a comprehensive, comparative study of mobility characteristics. Our study has shown evidence that long-tail distributions characterize well flight lengths, pause durations, and area popularity; that the invisible boundary of human movement also appears for NVEs, and that avatars do have preferences to different areas. We have also indicated several differences between NVE and NRE mobility characteristics: the flight lengths distributions have longer tail for NREs, the personal preference in area-visitation is more pronounced for NREs. We propose SAMOVAR, a Statistical Area-based MObility model for VirtuAl enviRonments. SAMOVAR models four mobility characteristics that are important to the performance of networked virtual environments: pause duration, velocity, area popularity, and distinct visited areas. Through simulation, we validate that SAMOVAR can produce all four characteristics. Further, comparing to results of simulation using real world traces,

SAMOVAR can produce very close results, while the results for Random Waypoint and HotSpot models are significantly different.

Chapter 6

Scaling NVEs through the Area-of-Simulation Mechanism and Architecture

Multi-Avatar Virtual Environments (MAVEs), such as Real-Time Strategy (RTS) [32] games, have a large market, with millions of users [217]. Contrary to the trend of Internet-based applications of allowing massive numbers of users to interact, the current generation of MAVEs design technology is not scalable. As a typical example, the StarCraft series limits the number of concurrent players in any gaming instance to 16; although hundreds of thousands of instances may run concurrently, they are essentially not communicating. This significant scalability limit stems from the difficulty of managing more than the hundreds of avatars that even this small number of players control in each game instance. Much previous work has focused on the scalability of Networked Virtual Environments (NVEs) [83, 137, 231], leading to mechanisms such as Event-Based Lockstep Simulation (EBLS) [172] [85] used in RTS games and military simulations, and the Area of Interest (AoI) [3] used in Single-Avatar Virtual Environments (SAVEs). However, as we show in this chapter, these mechanisms cannot be used to scale current MAVEs far beyond their current limits. To address the problem of scaling MAVEs, we propose the *Area of Simulation* scalability mechanism, we design an architecture around it, and we implement and deploy this architecture to demonstrate its scalability.

In MAVEs, users can have and control simultaneously multiple avatars which are their virtual world representations. We focus in this chapter on an important type of MAVEs: RTS games, such as Blizzard's StarCraft and Microsoft's Age of Empires series, are essentially Internet-based real-time world simulations in which players control avatars to gather resources, to construct buildings, to train combat avatars, to explore unknown territories, to trade, and to conquer.

Two main problems prevent MAVEs from scaling. First, the resource capabilities of

individual players may be exceeded: the bandwidth can become insufficient for transmitting messages, the computers of the players can become overloaded in trying to update the local copies of the game world status, etc. Second, MAVEs require strong consistency among the players for important areas of the virtual world: deciding which vehicle to move, where to build an important warehouse are decisions of precise location. Providing a consistent view to all players is challenging. When the scalability requirements are not met, the consequences for the game operators can be significant: players may quit en masse.

Two of the most commonly used scalability mechanisms: EBLS and AoI, do not work for MAVEs. EBLS [83, 85, 172], the predominant *event-based* operational model for RTS games, uses lockstep simulation [14] to ensure a globally consistent execution order of events. EBLS trades off computation for bandwidth, by transmitting only events and by having every player recompute the state from the received events. EBLS consumes lots of computational power, and it cannot scale to hundreds of players on commodity computers, as we have previously shown in our evaluation of RTS games [200]. AoI uses an *update-based* operational model in which clients do not perform simulation of game states, but receive state-updates for objects in close in-game proximity. AoI-based approaches can scale to hundreds of concurrent avatars for *single* AoI. In this chapter, we analyze a large number of game traces and show that single-AoI approaches are not suitable for RTS games, which exhibit *multiple*, often-changing AoIs.

We propose, in this chapter, the *Area of Simulation (AoS)* scalability mechanism, which combines and extends the EBLS and AoI mechanisms. The AoS mechanism allows different areas of the virtual world to employ different operation models, from *event-based* to *update-based*, depending on the recent interest shown by the player. The AoS mechanism is the first mechanism to combine the event-based and the update-based operational models for managing the areas that a player is interested in.

We further design a system architecture for MAVEs with as its main feature the support of multiple, dynamic AoIs managed using the AoS mechanism. This architecture also includes two message dissemination mechanisms to reduce bandwidth consumption. We demonstrate the viability of our architecture through realistic simulations and through real-world experiments with a prototype game. By implementing and deploying a working system, we show that for a prototype yet realistic game, our architecture enables an order of magnitude more users than the state-of-the-art while satisfying the overall requirements of MAVEs.

In summary, our main contribution is five-fold:

1. We show that most MAVEs users have multiple areas of interest, which can change often during gameplay. Thus, traditional approaches with a single AoI work poorly (Section 6.1);

2. We propose a new scalability mechanism for MAVEs, the Area of Simulation (Section 6.2).

3. We propose a system architecture for MAVEs based on Area of Simulation that can scale to hundreds of concurrent players with tens of thousands of avatars (Section 6.3).

4. We implement this architecture and evaluate the architecture in simulation (Section 6.4) and with a real-world prototype RTS game (Section 6.5).

5. We compare our work with a large body of related approaches, both quantitatively (Section 6.4 and 6.5) and qualitatively (Section 6.6).

6.1 Background

In this section, we discuss the characteristics and requirements of MAVEs in Section 6.1.1, and we present a MAVE model in Section 6.1.2, we identify and discuss, in turn, three main challenges in fulfilling the per-command and overall latency requirements of MAVEs (Section 6.1.3). We also show that the traditional AoI mechanism, which is widely used to scale NVEs, is not efficient for MAVEs (Section 6.1.4). Thus, a new scalability mechanism and an accompanying architecture are needed to scale MAVEs.

6.1.1 Characteristics and requirements of MAVEs

MAVEs such as RTS games have unique characteristics and requirements among NVEs [49]. Unlike other NVEs, such as First-Person Shooter (FPS) and Role-Playing Games (RPG), in which the player controls one avatar and may encounter at any time at most a few tens of other avatars (often not human-controlled characters), in RTS games the players often need to control many tens or even hundreds of avatars and in-game buildings, etc.

The control in RTS games combines long-term strategic decisions, including macro-management of resources such as buildings and large groups of avatars; short-term strategic decisions, including management of small groups of avatars; and quick tactical decisions, including micro-management of individual units. Usually, players expect the latencies not to exceed several seconds until the commands they issue are executed, or even less [49, 151]. Moreover, even if individual commands take long to be executed, the overall responsiveness of the game should not be compromised: players expect to see their game visuals updated at a rate of over 24 frames per second [91].

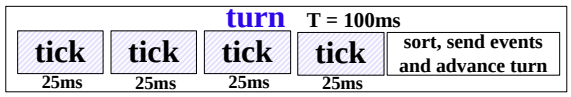


Figure 6.1: Turn and tick.

6.1.2 A System Model for MAVEs

In MAVEs, each user can have multiple avatars and in-game buildings. Each avatar has a pre-defined speed and range of vision. Typically each MAVE user has a base to produce/train the user's avatars.

Many RTS games, such as StarCraft, Age of Empires, 0 A.D, OpenTTD, and Zero-k, are EBLS-based systems [85, 172], for which events can be triggered not only by user input (commands) but also by the (discrete) passage of time. Events are spatial and temporal, that is, they have a well-defined area and duration of effect. In this chapter, we use the terms "command" and "event" interchangeably.

In EBLS, the virtual world simulation is temporally divided into multiple simulation *turns*. Each turn has a pre-defined real world length T ms; after T ms, a turn is cutoff and a new turn is started. A *turn* is further divided into multiple simulation *ticks*. During each tick, the virtual world will perform simulation logic and render virtual-world objects' updates. In Figure 6.1, a turn's duration is set to $100\,ms$, and the turn is divided into 4 ticks, with each tick's duration equal to $25\,ms$. During a turn, each client will send events to a server. At the end of a turn, the clients send turn advance messages to the server. Upon receiving the turn advance messages, the server will sort all the events received and send the sorted events to all clients, for execution. According to [172] and our own experience, the time spend for sorting and sending (non-blocking) events is negligible. A tick can be rendered using one or multiple frames; in this chapter, a tick is equivalent to one frame and we use the terms "frame" and "tick" interchangeably.

In this section, we identify and discuss, in turn, three main challenges in fulfilling the per-command and overall latency requirements of MAVEs (Section 6.1.3). We also show that the traditional AoI mechanism, which is widely used to scale NVEs, is not efficient for MAVEs (Section 6.1.4). Thus, a new scalability mechanism and an accompanying architecture are needed to scale MAVEs.

6.1.3 Challenges in fulfilling MAVEs latency requirements

Resource challenge: Scaling MAVEs under tight latency requirements can be limited by the lack of sufficient computational and networking resources. Although the average upload-bandwidth required by RTS games is 2-8 KB/s for 8 players, it increases quadratically with the number of players [49], and for 100 players it can easily exceed 1 MB/s. We have also shown in our previous evaluation of RTS games [200] that, as the number of players increases, the computational resources required to update the game world can

exceed the local computing power of modern commodity computers.

Game-design scalability challenge: We have also shown [200] that to deliver good gameplay experience when the number of players increases, a proportional increase in the size of the virtual world needs to occur, making the simulation of the virtual world even more computationally demanding than in today's commercial games.

Consistency challenge: Current and future RTS games require good consistency among players, especially for important areas of the game map (e.g, places of interest). It has been noted [49] that RTS games do not require location consistency as accurate as for FPS and RPG games, where the accuracy may make the difference between a player dying or living in the virtual world. However, avatar micro-management, which has recently become very popular due to the release of games such as StarCraft and to the growth of global competition networks [151], requires game-state consistency on-par with FPS and RPG games among the players simultaneously moving avatars in tight areas. For example, a trooper may be saved from disappearing by moving it in time just outside the fire range of an opponent's tank.

6.1.4 Presence of Areas of Interest in MAVEs

In this section, we show that *the AoI mechanism cannot support MAVEs well*. The AoI mechanism, adopted by many NVEs, exploits the interest shown by users to specific avatars or map areas, to reduce the traffic needed for progressing to the next simulation tick. However, previous approaches use only a *single* AoI per user, the location of which is defined as the area surrounding the virtual world location around the user's *single* avatar[1]. To study the potential use of AoI in MAVEs, we analyze the real use of StarCraft II (SC), one of the most popular RTS games, as a representative MAVE. Through the analysis of about 6,000 logs of SC matches, we show that most MAVEs users have each not a single but *multiple* AoIs in game, and that players switch quickly among their set of AoIs.

We collect 5,796 replays of SC from `sc2rep.net`, a popular repository of community-rated game replays. The replays have been created and uploaded to the website, for review by other players, by over 1,000 users. The replays are played between July 2010 and November 2010, and the average duration of the replays is about 13 minutes. We use the publicly available tool `SC2Gear` to extract from each replay the complete set of timestamped, location-aware commands.

The size of SC maps ranges from 64×64 to $256 \times 256\, tiles$. The speed of the fastest moving unit is about $7.5\, tiles/s$ and the broadest range of vision of in-game units is $14\, tiles$. The actual size of the map area that is viewable on-screen depends on the aspect

[1] Some SAVE games such as World of Warcraft, allow players to level-up multiple avatars, but the avatars cannot be controlled simultaneously in the same game instance

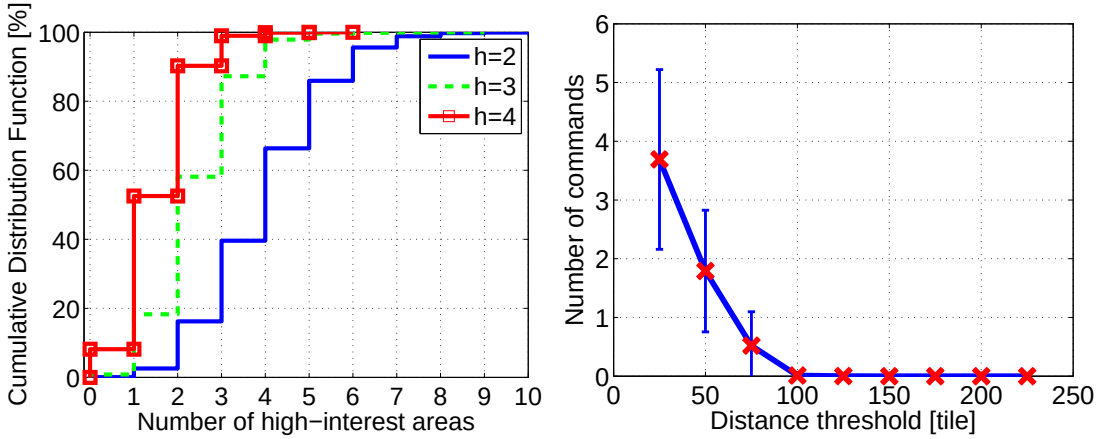


Figure 6.2: The presence of multiple, changing AoI in MAVEs: number of high-interest areas per user (left) and dynamics of interest (right).

ratio of the user's monitor, but we conservatively estimate that each screen can display map areas of about $25 \times 25\,tiles$. Thus, we divide for each replay the map into areas of $25 \times 25\,tiles$ and count the commands issued in each area. Because we are looking for areas of much higher than average interest, which correspond to the intuition behind AoI, we define a *high-interest area* as the area for which the number of issued commands is h times higher than the average number of commands issued per area.

Most users have multiple AoIs. Figure 6.2 (left) depicts the distribution of the number of high-interest areas per player when setting $h = 2, 3, 4$. As the figure shows, for $h = 2$, only less than 5% of the players have one high-interest areas and about 90% of the players have more than two high-interest areas. The maximal number of high-interest areas of a player is 16. For $h = 3$, only 10% of the players have one high-interest areas and about 80% of the players have more than two high-interest areas. For $h = 4$, over 40% of the users have 2 or more high-interest areas. Overall, we conclude that most of the players have 2–6 high-interest areas. This can be explained by observing that advanced players employ a mix of macro- and micro-management (see Section 6.1.1) in different areas of the game map.

Users switch among their AoIs in the virtual world, often with high frequencies. We look at the distance between the virtual world location of commands issued over short periods of time. For each replay, we split the duration in a series of 10-second time periods, and analyze the commands issued in each period. We consider a distance threshold x, in turn, from 25 (the screen size) to 225 (the maximum map size minus the screen size), in increments of 25. For each distance threshold and each period, we count the number of commands issued further than the threshold from the location of the first command in the period; such a command would require an AoI switch. Figure 6.2 (right) depicts the mean command-counts for various distance thresholds; the error-bar depicts the standard error. A point (x, y) on the figure should be read as "users issued, on average, y commands

whose distance from the first command is larger than x over each 10-second period". Values $y \geq 1$ indicates that it is likely that users need an AoI switch *every* 10-second period. The results indicate that players often issue commands that switch the current screen (the current focus area), effectively switching their AoI.

This new phenomenon, of the presence of multiple, frequently changing AoIs per MAVE user, is an important motivation for the mechanism and system architecture we will introduce later. If we adopt the traditional AoI approach, which maintains only single AoI per player, choosing the size of AoI as the size of the screen will lead to significant AoI switching management overhead, and late delivery of states. Alternatively, the size of the single-AoI area could be very large, to cover all the possible areas of interest, thus leading to inefficient resource usage. We further show the inefficiency of the single-AoI approach via simulation in Section 6.4.2.

6.2 The Area-of-Simulation Mechanism

In this section, we introduce a new scalability mechanism, the Area of Simulation (AoS). The key characteristic of our mechanism is the combined use of the EBLS and AoI, to efficiently maintain the areas of interest of each player.

The AoS mechanism adopts a distributed server architecture in which the virtual world (map) is divided into non-overlapping sub-maps. Each sub-map is simulated by one server and can also be simulated by clients. The AoS mechanism contains three parts: the partitioning of the virtual world into different types of areas, the mapping of areas, and the simulation of areas. We describe these three parts in the following, in turn.

6.2.1 Partitioning the Virtual World

From the user's point of view, the virtual world is partitioned into a number of areas. These areas can be areas of interest (AoI) or areas of non-interest (AoN). For each area of interest, depending on the operational model adopted, an area can be either a simulation area (SA) or an update area (UA).

For each SA, the game client receives events of that area, and then performs the simulation of that area (akin to EBLS-based operation). For virtual world objects and avatars in SAs, users have the most up-to-date and precise information. Currently, each sub-map can be operated as an SA. For each UA, the game client receives messages about the state-updates of that area, and updates the state of that area accordingly (akin to AoI-based operation). The visible area of a player's avatar can be a UA. For each player, a UA can overlap with the other UAs but not any SA. Users may receive different frequencies and precisions of the state-updates. Thus, for UAs, the user may have less up-to-date and lower-fidelity game-states than for SAs. For AoNs, the user will not receive any messages.

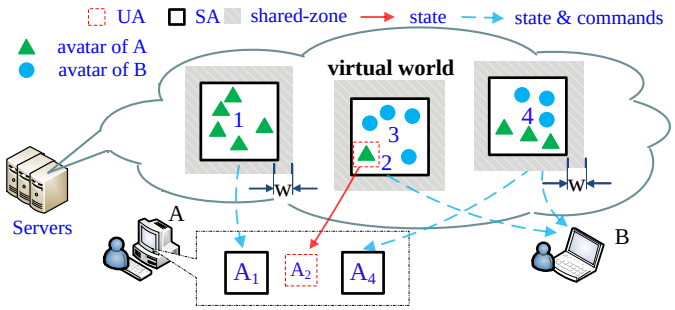


Figure 6.3: A game map and overlapping simulation areas (SA) and update areas (UA). w is the width of the shared-zone.

6.2.2 Mapping of In-game Areas to Real-World Resources

The *mapping* of an area, to either SA, UA, or AoN, depends on the user's interest, application logic, and resource availability. When the game client decides, based on measured or predicted interest, that the user should receive the most up-to-date information of an area, it classifies it as an SA. Similarly, areas of little or no interest are classified as UA and AoN, respectively. The lack of sufficient resources to manage SA or UA may force the game client to re-classify an area to a lower class.

An example of the AoS is illustrated in Figure 6.3. The virtual world contains sub-maps 1, 3, 4, and the other submaps. Player A has many avatars located in sub-maps 1 and 4, which are therefore classified as player A's SAs (A_1 and A_4). Player A also has an avatar exploring sub-map 3, the area visible to that avatar is a UA of user A (A_2). UA A_2 of user A is a special situation: although user A is highly interested in this area, user A already has two other SAs and, due to a lack of computational power, cannot afford to fully simulate another SA; instead, player A will use excess bandwidth to get updates from the server responsible for area 2. We will further describe the management of areas in Section 6.3.2.

6.2.3 Simulation of In-game Areas

Because the virtual world is partitioned, events that can affect multiple areas raise a *data replication* problem for the AoS mechanism. Processing such a shared-event may require access to state information from all the areas the event affects. To provide users with the illusion of a seamless un-partitioned virtual world, the AoS adopts a *shared-zone* technique to manage data replication of areas. Areas that can be affected by shared-events maintain shared-zones that overlap with other areas. As Figure 6.3 shows, the shared-zones are the gray areas around each SA. We choose the width w of the shared-zone to be larger than the maximum effect range (MER) of the shared event. The MER can be pre-determined by application designers according to game logic. In all MAVEs we have surveyed, the MER is small relative to the on-screen view (the maximal vision

Algorithm 1 Operation of the AoS mechanism.

1: **while** not end-of-game **do**
2: increment game tick
3: receive *last turn's* events issued in SAs
4: receive *last turn's* events issued in shared-zones surrounding SAs
5: receive *last turn's* states from shared-zones
6: receive *summary* of state updates of UAs
7: update game states according to DVEs logic by applying events to states of SAs
8: interpolate game states, display current states, and receive user's commands
9: advance simulation turn and send turn advance message, if all the events of a turn have been received;

range of avatars in StarCraft is 14 tiles, while the screen width is 25 tiles). Each server will exchange with the others the states of shared-zones it manages for processing shared-events.

The state of the virtual world may be changed with the passage of time, and by different events and by distributed clients multiple times, even inside a simulation tick. Thus, knowing which is the correct state after the update has been made is the problem we are facing. The CAP theorem [29] states that Consistency, Availability, and Partition-tolerance cannot be simultaneously guaranteed in a distributed system. In AoS, different parts (partitions) of the virtual world are hosted in the Internet where *temporary partitions* of the network caused by latency or message loss are bound to happen. During the period when the network is partitioned, we can either cancel the players' operations and thus decrease availability, or process the players' commands but with the risk of inconsistency. Following some previous work [18, 19, 147] which treat consistency as a non-first requirement, we believe that the availability of DVEs (players receive responses for their operations in time) is more important than the consistency of DVEs. Thus, we do not ensure strong consistency as EBLS does. In AoS, we choose to partition the virtual world for scalability but at the cost of some inconsistency. In other words, we *trade off consistency for scalability* by allowing some game states to become inconsistent. By adopting this approach, we allow by design that the states of some parts of the partitioned virtual world may be different than the states that would have resulted from the same sequence of player commands executed in an un-partitioned virtual world.

The simulation operation of the AoS mechanism is described in Algorithm 1. For each simulation tick, each *client* receives the *commands* from the servers of SAs (line 3); additionally, the client receives all the commands issued in the shared-zones of the SAs (line 4). Each client also receives the *states* of the shared-zones (line 5), which enables processing the events that require information from the neighboring areas. For UAs, each client receives a summary of state-updates from the server managing each UA (line 6); depending on the resource availability of the server and on the game logic, the client

Table 6.1: Summary of mechanisms used in the AoS architecture.

Problem	Mechanism	Novelty
Change of users' interest	dynamic area management (Section 6.3.2)	new
Bandwidth consumption of UAs	forwarding pool and level of detail (Section 6.3.3)	adapted
Bandwidth consumption of shared-zones	delta encoding (Section 6.3.4)	re-used

may receive updates of various details and with various frequencies. After receiving all the needed information, the client will perform simulation to update its local view of the virtual world (line 7–8). During each tick, the *server* needs to read events from clients, read events and states of shared-zones from the other servers, sort the events, perform the simulation, and send the events and states to the clients.

For the simulation operation, as we trade off consistency for scalability, some inconsistency may happen compared to the un-partitioned simulation. We rely on the application developer to use additional techniques to mitigate this drawback [18, 42]. For example, two avatars from different areas may collide unexpectedly. Application-specific logic, such as using distributed collision detection [42], allowing the bounding boxes used for collision detection to be slightly bigger than the actual size of the avatars, can be used. In this chapter, we simply disable collision detection following the practice of World of Warcraft (WoW) [147]. In Section 6.4.3, we will show that the AoS mechanism does not introduce much inconsistency: most of the time, the inconsistency is unnoticeable.

6.3 The Area-of-Simulation based System Architecture

In this section, we integrate the AoS mechanism into an MAVE architecture. Table 6.1 summarizes the problems and the mechanisms (Section 6.3.2 to 6.3.4) adopted to solve them. Last we discuss the implications and limitations of our mechanisms in Section 6.3.5.

6.3.1 Architecture Overview

To operate the entire virtual world, our architecture comprises three types of logical nodes: the registration server, the clients, and the area servers. The registration server is responsible for the registration of the other types of logical nodes and to reply to queries to locate an area server. We describe the functions of the clients and of the area servers in the following.

Each client is responsible for managing the virtual world for a player. Clients can connect to multiple area servers. If a client has an SA which is managed by an area server, the client is a Simulation Area Client (SAC) for that area server. Similarly, if a client has a UA which is managed by an area server, the client is an Update Area Client (UAC) for that area server.

Each area server is responsible for managing an area and for communicating with clients. Each area server is an SAC for itself. It is responsible for receiving the commands issued by players in its area, for forwarding the commands in its area and the shared-zones' states to all its SACs, for forwarding its states to all its UACs, and for forwarding the selected state of its shared-zones and commands to neighboring area servers. An area server is neighboring to another area server only if the areas they manage are spatial neighbors in the virtual world.

6.3.2 Dynamic Area-Management Mechanism

Users may change their interest at run time. We propose a *dynamic area management* mechanism to adapt to the change of users' interest. The mechanism allows each user to have up to n SAs concurrently, where n can be predefined by the application developers according to application logic. To select the SAs and UAs for each player, our mechanism relies on a *dynamic and automatic ranking* of areas, using the *level-of-attention* (interest) shown recently by the player. The top-ranked n sub-map candidates are marked to become SAs, and the avatars' visible areas which are outside the selected SAs are marked to be UAs. In this chapter, the *level-of-attention* of a sub-map v is calculated as:

$$v = w_1 \times \sum_i (\frac{s_i}{s_t}) + w_2 \times \frac{t_a}{t}$$

where, w_1 and w_2 are the weights whose sum is 1; s_i is the score (assigned by application developer) of the player owning an avatar with id i located in one area, and s_t is the total score of the avatars owned by the user; t_a is the accumulated time that the player has seen the area rendered on-screen. This mechanism is executed every t seconds (i.e., $t = 30$). w_1 is the relative weight of spatial metric while w_2 is the relative weight of temporal metric. MAVE designers can tune these weights according to their designs, for example, by setting w_2 higher, the MAVEs can respond to players recent activities faster. The score s_i can be assigned by the application designer according to game logic. For example, in Age of Empires, a swordsman needs to be trained using a certain amount of in-game resources (e.g., 50 units of food, 20 units of gold), hence the score of a swordsman can be assigned as 70 (50+20). To avoid the frequent changing of the level-of-attention ranking, a sub-map is an SA candidate, only if its level-of-attention is higher than a threshold th (i.e., $th = 0.1$). In our system there are two types of players, human and artificial-intelligence (AI) players. For AI players, the time factor is not taken into account ($t_a = t$).

There could be many ways to calculate the level-of-attention values by using spatial, temporal, social, and machine-performance metrics. For example, spatial metrics can

include the location of the player's base, or the number of avatars present in the area, or the total amount of in-game resources invested by the user in that area. Temporal metrics can include the number of recently issued commands, or interaction history. Social metrics can include a summation of the interest shown by in-game allies. Machine-performance metrics can include a dynamic assessment of the computing and network capabilities of the player's machine. For example, a resource monitor can be integrated into each player's machine, once a machine cannot maintain the minimal simulation speed of a sub-map, the level-of-attention of the sub-map can be calculated as zero, which leads to the demotion of the sub-map to a UA. We leave further exploration of the calculation of level-of-attention as future work.

The dynamic ranking of areas allows for the creation and destruction of areas for each player. For example, in RTS games it is customary to build temporary bases with tens of mobile and immobile avatars; such temporary bases can lead to a temporary SA being created. As areas may be promoted to SA status, or demoted to UA or AoN status by the area management module, the MAVEs operator can provision and allocate resources adapted to the player's needs, thus maintaining the quality of service.

For a client, upon losing interest in an SA, the area becomes first UA(s); further neglect leads to the conversion into an AoN, which makes the area server stop communicating with the client. Conversely, if the level-of-attention ranking of an area increases, a new SA needs to be created from an existing UA. First, the area server pauses the simulation procedure of the sub-map contains that area, and serializes the data for that sub-map. Then, the area server sends to the client all the needed state information and pending commands from the current simulation turn. Finally, the area server resumes the simulation and the gaming procedure is as usual. During this procedure, the area server needs to inform the neighboring area servers that the simulation of this sub-map is paused and the other neighbor area servers will not need to wait for the states sent by this sub-map. We plan to incorporate the well-known mechanism of live-migration [48] to make the UA-to-SA transition without any pause.

When UA-to-SA or SA-to-UA transitions occur, depending on the current counts and limits concerning each area type, other areas may be demoted to or promoted from the status of SA and/or UA. Cascading occurrence of transitions of UA-to-SA or SA-to-UA is unlikely to happen. However, to avoid such situations, AoS imposes a UA-to-SA rate on each server of sub-map. For each sub-map i, at most k_i (i.e., 100) clients can become SAC of a sub-map within a t (i.e., 30 seconds) time window. System designers can configure t and k_i according to the computation power of servers, which is orthogonal to our work.

6.3.3 UA State Dissemination Mechanisms

During the process of simulation, an area server needs to send state-updates to all connected clients, with high frequency. If an area is popular, the bandwidth consumption may exceed the capacity of the server managing this area. To alleviate this situation, we build a *forwarding pool (FP)* mechanism, in which the area server makes use of the idle upload bandwidth of the SACs. By making use of a unique property of the AoS mechanism, that all the SACs of an area have the same data as the area server, our FP can use some of the resource-rich SACs to help disseminating the states. In our current design, all the SACs (including the area server) of that area run a round-robin algorithm to select, in turn, an SAC to send state-updates to one UAC.

By using the FP mechanism, the upload-bandwidth consumption of servers can be greatly reduced. However, if there is no SAC or the aggregate upload bandwidth of the SACs cannot meet the demand of all the UACs, some form of state-reduction technique is needed, leading to less up-to-date states. We design a state-reduction mechanism, *level of detail (LoD)*, which effectively reduces network consumption by sending state-updates of different avatars at different frequencies, instead of a single fixed frequency. Each avatar i is assigned a score s_i which is determined using the same method in Section 6.3.2. The higher s_i is, the higher f_i is. Firstly, the avatars are sorted according to their scores in decreasing order, top $p\%$ of the avatars' state-updates will be sent every tick. Secondly, for the other $(100 - p)\%$ avatars, each avatar i has its own update frequency f_i; Thirdly, when a user issues a command to an avatar whose id is k, the state of the avatars that surrounds avatar k will be sent to the user at every tick. After t_{LoD} ticks, if there are no further commands from the user which affect avatar k, the update frequencies of avatars around avatar k are restored to their normal update frequencies. p, f_i, and t_{LoD} can be assigned by application designers according to game logic. To obtain the optimal parameter setting of the LoD mechanism, we recommend that for complex games, a combination between game designer expertise and experimental tuning is needed. For all the avatars whose state updates are not sent every simulation tick, our mechanism relies on techniques such as dead-reckoning [18, 19] to interpolate/extrapolate the avatars' positions. The LoD mechanism promises to reduce the network consumption significantly, without reducing the accuracy of information about the avatars that the player is paying attention to.

6.3.4 Shared-zone State Dissemination Mechanism

The area servers of the AoS mechanism need to send states from their shared-zones to all their SACs. As we will show later in Section 6.4.2, the states can consume more than 20% of the server upload bandwidth. Thus, we use a *delta-encoding* technique [126] to reduce the bandwidth consumption. Delta-encoding technique sends the difference of data

instead of sending original data to client. When sending the shared-zones' state to clients, the area server will first get the difference of data, and then transfer only the difference of data to clients. If the states of shared-zones only changed slightly since the last state transfer, this technique can significantly reduce the bandwidth needed for sending states of shared-zones.

6.3.5 Implications and Limitations

The AoS mechanism gives DVE designers the ability to tune the system by configuring the trade-off between the high-fidelity, compute-intensive SAs and the relatively low-fidelity, network-intensive UAs. In this way, the AoS mechanism addresses for MAVEs the resource challenges in Section 6.3.5 and the consistency challenges in Section 6.3.5. We also discuss some limitations to the AoS mechanism in Section 6.3.5.

Resource Challenges

The AoS mechanism has good scalability, because, as only a few areas catch the interest of each player (see Section 6.1.4), each client simulates only a few, high-interest areas. In contrast to the EBLS mechanism, the AoS mechanism does not simulate the entire virtual world. Different from the traditional AoI mechanism, the AoS mechanism reduces the network consumption by transferring commands besides using state-updates.

Consistency Challenges

Comparing to the update-based DVEs such as World of Warcraft, which have sequential consistency server-side and eventual consistency client-side, the AoS mechanism provides sequential consistency for SAs, and eventual consistency for UAs and shared-zones both on server- and client-side. The AoS mechanism can satisfy the users' consistency requirement that the areas that players are interested in have high consistency guarantee.

Limitations

The AoS mechanism adds some complexity into the design of MAVE servers. A DVE which is designed for single-server architecture may need to be re-designed to adapt to the distributed-server architecture of the AoS mechanism. The DVE designers may need to conduct experiments, to determine the area management parameter (n) to achieve optimal scalability for a specific DVE. However, as we will show in Section 6.4, even by setting $n = 1$, the AoS mechanism is more scalable than the other models.

The AoS mechanism ensures eventual (instead of sequential) consistency in the shared-zones areas. This may dissatisfy users who frequently control their avatars in

Table 6.2: Overview of experiments in Section 6.4.

Experiment	Evaluation target
Comparison with alternatives (Section 6.4.2)	Whether AoS scales under different scenarios
Comparison of Area Management mechanisms (Section 6.4.2)	Whether the dynamic management mechanism works
Forwarding pool and level-of-detail (Section 6.4.2)	The message reduction of these mechanisms
Consistency evaluation (Section 6.4.3)	Measuring the consistency tradeoff of AoS

shared-zones (e.g., FPS players). There are two possible solutions. One is to use multiple levels of consistency control protocols for different actions/objects [127]. For example, a strong consistency control protocol (e.g., two-phase commit) can be used for death events. Another possible solution is to change the design of the DVE, by reducing or limiting the chance that users are controlling avatars in the shared-zones (e.g., the border of each sub-map can be designed to be non-interesting to players).

6.4 Simulation Results

In this section, we evaluate AoS and four alternatives experimentally in a simulated environment. We present results obtained in a real-world environment in Section 6.5. Overall, our results indicate that AoS *is more scalable than the alternatives*.

We describe the experimental setup in Section 6.4.1. In Section 6.4.2, we compare AoS against four alternatives. The results show that AoS can achieve much lower network consumption than the pure update-based model (e.g., AoI), due to using the idle CPU resources on the client's side, and AoS can achieve much lower CPU consumption than the pure event-based model (e.g., EBLS), due to simulating only parts of a virtual world. In Section 6.4.3, we show that AoS achieves scalability without sacrificing too much consistency: 99.5% of the drift distance [64] of avatars can converge within about 0.3 seconds (a limit that is acceptable even for advanced users, see Section 6.1.1). Table 6.2 summarizes the experiments conducted in this section.

6.4.1 Experimental Setup

The default experiment parameters are shown in Table 6.3. The simulation is running on a $1280 \times 512\, tile$ map, partitioned into 5×2 sub-maps of $256 \times 256\, tile$ each. The simulated virtual world map is 10 times larger than the largest game of StarCraft II. This map size is consistent with our goal to scale this exemplary game and with the game-design scalability challenge (Section 6.1.3). As determining the maximum number of

Table 6.3: Default experiment parameters.

Name	Meaning	Values
$w \times h$	virtual world size	1280×512 tiles
N	number of users	[10 to 400]
c	frequency of user's input	1 command per 10 ticks
K	number of avatars per user	50
n	maximum number of SAs per user	1
v	avatar speed	1 tile per tick
r	avatar vision range	10 tiles
w	width of shared-zone $w = r$	10 tiles
tps	number of ticks per second	40 ticks
nt	number of ticks per turn	2 ticks
sl	simulation length	10,000 ticks
w_1, w_2, s_i	level of attention parameters	$w_1 = w_2 = 0.5, s_i = 1$

SAs that each user's machine can support is orthogonal to our work, we assume that each user can have up to n SAs.

Each player is assigned a base, uniformly, randomly distributed across the virtual world, with 50 avatars distributed around it. Each player will set the sub-map where the base locates in as an SA, and keep the area as an SA until the end of the simulation. The simulator is configured to update with a frequency of 40 ticks per second. Each avatar's vision is a square, centered on the avatar, with a range $r = 10$ tiles. The movement speed of the avatars is 1 tile per tick. The vision range and movement speed is similar to the setup of StarCraft II.

We run the simulation for 10,000 ticks (we have run some experiments with 50,000 ticks, and the results are similar). Unless otherwise specified, all the simulations are conducted with 60 users (about 4 times larger than the maximum number of players in one game of StarCraft II, which is 16) in a simulated LAN environment with no latency. Following the design of the very popular RTS game Age of Empires [172], all the commands are scheduled to run 2 turns later.

Workload models: Modern MAVEs such as RTS games do not support more than 32 players in a game instance, so we are not able to obtain real-world workload traces with many users. Instead, based on our experience with popular RTS games [172] and the code of a modern open-source RTS game engine [1], we evaluate AoS against four different *workload models*. Each workload model is a combination of a *command model* and a *mobility model*. For the command model, each player is restricted to issue 1 command per 10 ticks, which is equivalent to 4 commands per second. This mimics player behavior during intense operations [172]. Each command will order a randomly selected avatar to go to a position according to a mobility model. The *mobility models* are Weighted random Walk (WW), Weighted random walk Inside sub-map (WI), Weighted random walk with Distance (WD), and SAMOVAR. In WW, we partition the virtual world into multiple non-

overlapping $16 \times 16\, tile$ areas, and randomly assign a weight w between 1 to 10 to each area. Each user is assigned i high-interest sub-maps, where i is sampled from the number of high-interest areas per user (with $h = 4$, see Figure 6.2 (left)). In WW, when a user commands an avatar to go to a new destination, the avatar selects a high-interest sub-map randomly. Then the avatar has a higher probability to go to a grid of the selected sub-map with higher weight, and it will go to a random position inside that grid. WI is similar to WW, but i is fixed to 1. In WD, the probability p to go to a grid is defined as $p = \frac{w}{d^2}$, where w is the weight (1 to 10) assigned to that grid, and d is the distance between the centroid of that grid and the player's base. For a player, this will make many avatars move in close proximity of the base, with only a few of the avatars going to some (valuable) spots away from the base. SAMOVAR is developed based on [198]. SAMOVAR acts similarly to WW, except that each user has a limited amount of grids to visit, and each user has different personal weights to visit those grids. Albeit we do not evaluate our system using real-users, the players change AoI more frequently in the four workloads than in StarCraft, so we believe that the scalability results achieved by AoS will be better for real-users than for the four workload models. Unless otherwise specified, WD is the default workload.

Metrics: We look at four metrics: the network bandwidth consumption (upload and download), the compute unit, and the drift distance which we define as follows. We count the number of messages sent/received as network bandwidth consumption. The compute unit is a reference value to estimate the CPU consumption. We calculate the compute unit, at each tick, as the number of avatars simulated at each client/server. We do not count the computation used for updating objects and for processing events (according to our measurement in the prototype implementation, the time required to perform the simulation of an avatar is about 200 times higher than the time required to process state-updates/events). The drift distance [64] is used to evaluate the difference of avatar positions between the partitioned (AoS) and the un-partitioned (EBLS) models. We issue the same commands at the exact simulation time in AoS and EBLS and compare the distances of each avatar's position obtained through AoS and EBLS. For each experiment, we report the 99.5 percentile of drift distance. Each experiment is repeated 10 times, and the metrics shown are the average values.

6.4.2 Scalability Evaluation: Proposed Mechanisms

In this section, we evaluate AoS under a variety of scenarios. Our main findings are that (i) AoS consumes at least 30% less bandwidth than all the other alternatives for all the workload models; (ii) AoS requires an order of magnitude less computation than EBLS, for the server; (iii) the improved AoS (that is, AoS for which we enable the mechanisms introduced in Section 6.3) can further reduce bandwidth consumption by up to 60%.

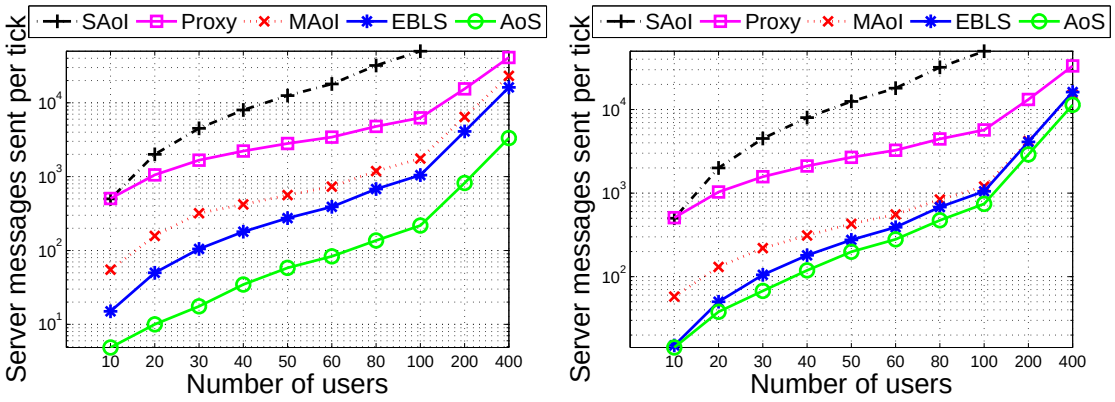


Figure 6.4: Network upload: (left) the WI workload; (right) the WD workload. (Logarithmic scale on vertical axes)

Comparison with alternatives

We evaluate the computation and network consumption of AoS against four alternatives: *single-AoI (SAoI)*, *multiple-AoI (MAoI)*, *proxy-server (Proxy)*, and *EBLS* explained in the following. *SAoI*, *MAoI*, and *Proxy* are pure update-based models, while *EBLS* is a pure event-based model. For each pure update-based model, a distributed server architecture is adopted, for which each server is responsible for simulating a sub-map of the virtual world. *SAoI* adopts a single, static area-of-interest approach, whose area is the whole map. In *MAoI*, each player can have multiple area-of-interest, and each area is the visible area around the player's avatars. MAoI represents an extension of current AoI techniques, but, unlike our AoS, lacks the areas with event-based updates (the SAs). MAoI can be also viewed as AoS without any SAs. *Proxy* [157] acts similarly to *MAoI*, but each server needs to send the states that it simulates to the other servers, for synchronization (the original Proxy uses one AoI per player).

Figure 6.4 shows the upload bandwidth consumption of the server for SAoI, MAoI, Proxy, EBLS, and AoS under WI and WD, in turn for various user counts. For ease of reading the figure, we truncate the results of SAoI when the number of users is larger than 100. SAoI consumes significantly more bandwidth than the other models; this result complements the analysis in Section 6.1.4 that a single static AoI does not work well for MAVEs. Proxy consumes the second-most bandwidth; compared to AoI, as Proxy needs to send additional messages to the other servers. It needs about 2 times and 1.5 times higher bandwidth than MAoI under WI and WD, respectively. AoS requires 30% up to 80% less upload bandwidth than EBLS, because the AoS servers only transfer messages that are relevant to the players instead of every message. Compared to MAoI, AoS consumes 40% up to 80% less bandwidth. This is because the AoS servers transfer commands besides state-updates to players, by making use of the players' idle CPU resources

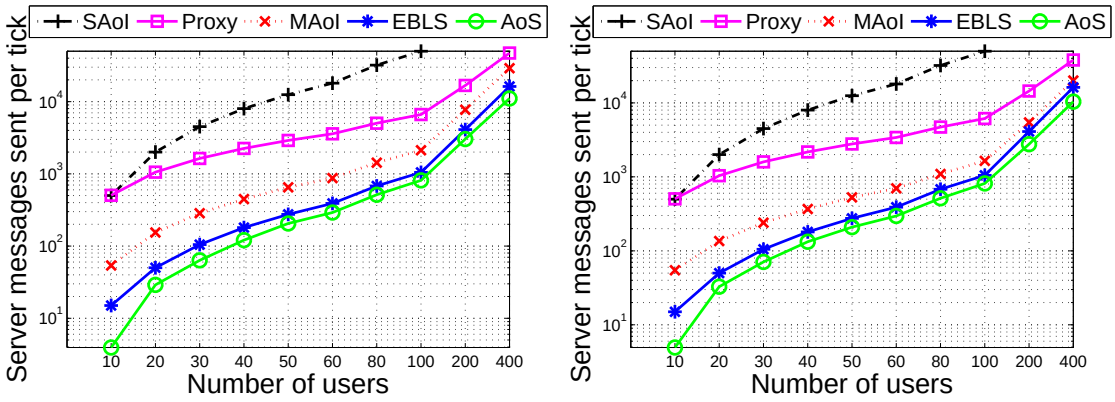


Figure 6.5: Network upload: (left) the WW workload; (right) the SAMOVAR workload. (Logarithmic scale on vertical axes)

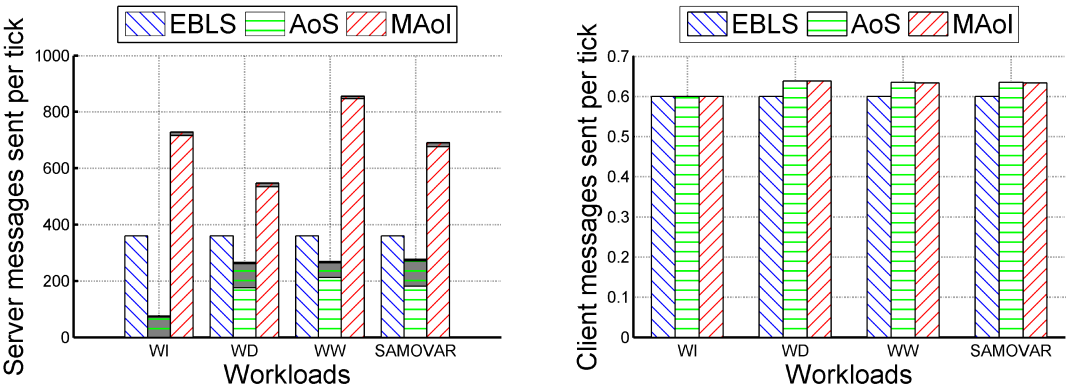


Figure 6.6: Network upload, 60 users: (left) server; (right) client. (For the servers' upload of AoS and AoI, each bar is divided into 3 parts, from top to bottom (dark, gray, and light): number of commands sent, number of shared-zones' states sent, number of state-updates sent. Servers of EBLS only send commands, thus the bars for EBLS are not partitioned.)

to perform simulation of the virtual world.

Figure 6.5 shows the upload bandwidth consumption for WW and SAMOVAR. The results are similar to the results for WI and WD: SAoI and Proxy consumes the most bandwidth, and AoS consumes the least bandwidth. The bandwidth consumption of AoS under WW and SAMOVAR is a bit more than that of under WI and WD, but AoS still consumes about 30% to 50% less than EBLS. As SAoI and Proxy consumes much more bandwidth than the other models, we only consider EBLS, AoS, and MAoI in the rest of our experiments.

Dissecting upload messages: Figure 6.6 (left) shows the upload bandwidth consumption of the servers for EBLS, AoS, and MAoI from left to right, grouped by four workloads with 60 users. The servers of EBLS only send commands, while for AoS and MAoI, com-

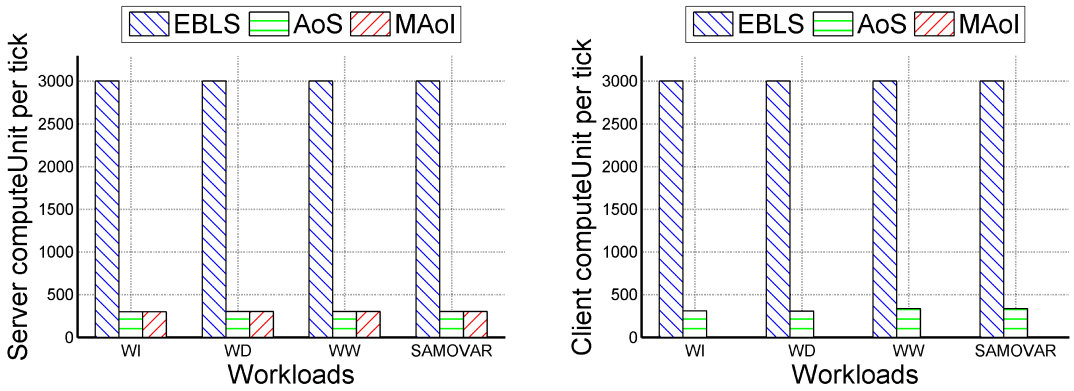


Figure 6.7: Compute Unit under 4 workloads, 60 users: (left) server; (right) client.

mands consume less than 5% of the network bandwidth, and the state-updates consume most of the bandwidth. For AoS, a large portion (more than 20%) of the bandwidth is used for sending the states of shared-zones. For MAoI, most ($\geq 95\%$) of its upload bandwidth is used for state-updates. Figure 6.6 (right) shows the upload of clients. The client upload is very low with less than 1 message per tick.

Computational overhead: Figure 6.7 shows the compute unit on the server-side (left) and client-side (right). On the server-side, AoS and MAoI consume the same amount of compute unit, and EBLS consumes about 10 times more. The compute unit depends only on the number of avatars simulated. For AoS and MAoI, each server only needs to simulate 10% of the avatars on average, while for EBLS, the server needs to simulate all the avatars. Thus, the compute unit of AoS and MAoI are only 10% that of EBLS. On the client-side, EBLS requires the same amount of CPU consumption on the client-side as that on the server-side. For AoS, each client needs to perform the simulation of a sub-map as the server of the sub-map. Thus, on average clients have the same amount of compute unit as that of servers. MAoI requires zero computations on the client-side, as the clients do not perform any computation.

Area management mechanisms and different numbers (n) of SAs

The previous experiments have shown that AoS with $n = 1$ is more scalable than the others. We show that by increasing n and adopting the dynamic area management mechanism proposed in Section 6.3.2, AoS can achieve much lower bandwidth consumption. We evaluate the impact of the area management mechanism, and the impact of the number of SAs per user by changing n from 1 to 4.

Figure 6.8 (left) shows the results of using a static mechanism which randomly and statically sets SAs for each user. In contrast, Figure 6.8 (right) shows the results of using the dynamic area management mechanism (described in Section 6.3.2). In general, the

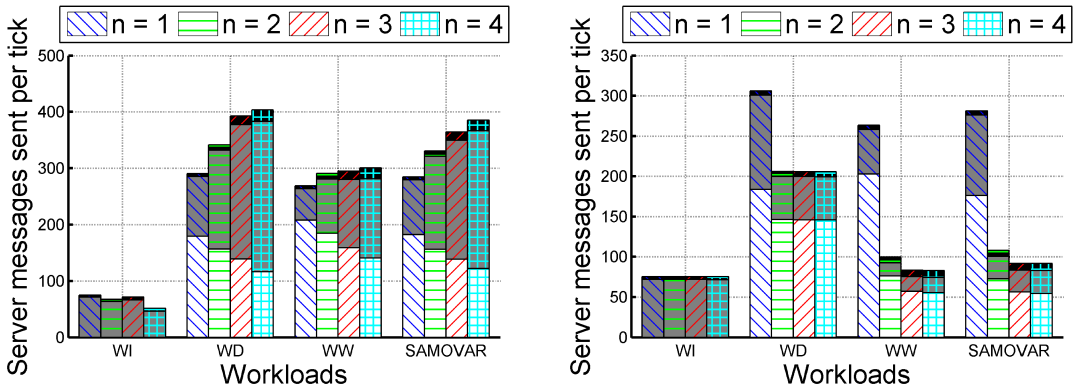


Figure 6.8: Server messages sent (left) without and (right) with dynamic management. (each bar is divided into 3 parts, from top to bottom (dark, gray, and light): number of commands sent, number of shared-zones' states sent, number of state-updates sent.)

number of state-updates sent for UAs (light bars in the figure) decreases with the increasing number (n) of SAs, while the number of commands sent (dark bars) increases.

As Figure 6.8 (left) shows, increasing n using the static mechanism slightly increases the bandwidth consumption for WD, WW, and SAMOVAR. This is because the number of shared-zone states that needs to be transferred increases with the increasing n, which decreases the bandwidth reduction through the use of more SAs.

As Figure 6.8 (right) shows, the dynamic mechanism achieves significant bandwidth reduction with increasing n. The amounts of shared-zones' states and state-updates sent by servers are both significantly reduced. For shared-zones' states, when a client has multiple neighboring SAs, the servers do not need to send the states of shared-zones of those neighboring SAs to the clients (the client itself has the master-copy of the states of shared-zones). The probability that a player will have multiple neighboring SAs is much higher when using the dynamic mechanism than when using the static mechanism, as the dynamic mechanism sets the top n sub-maps which contain more avatars of the player as SAs instead of randomly. For the state-updates, as most avatars are located in SAs for the dynamic mechanism, the servers send much fewer state-updates to clients than that of the static mechanism.

Increasing n using the dynamic mechanism does bring some overheads. The compute units used by servers do not increase, as each server only needs to simulate one sub-map regardless of n. For the clients, as Figure 6.9 (left) shows, the compute units increase with n. This is because with increasing n, each client has more SAs, which leads to increased simulation overheads. Figure 6.9 (right) shows the number of UA-to-SA transitions per sub-map. For WI, the number is zero, because the avatars of each player only move inside one sub-map, thus only one sub-map can be an SA for each player regardless of

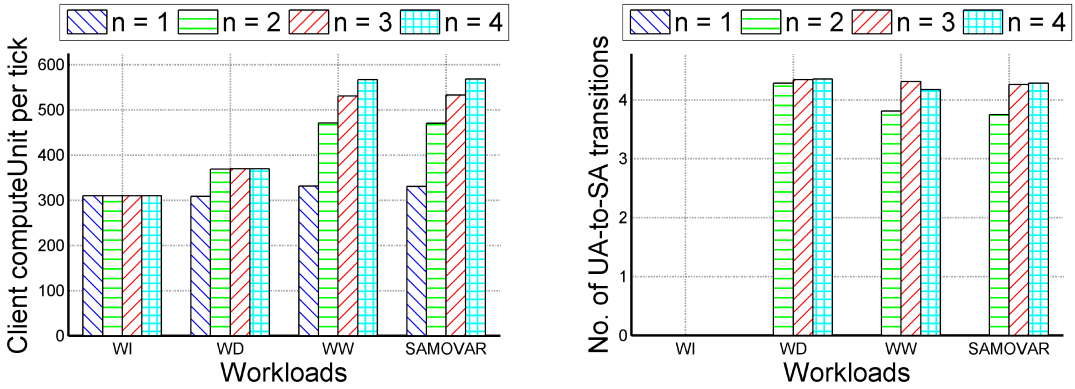


Figure 6.9: Dynamic area management mechanism with increasing n: (left) client compute unit and (right) number of UA-to-SA transitions

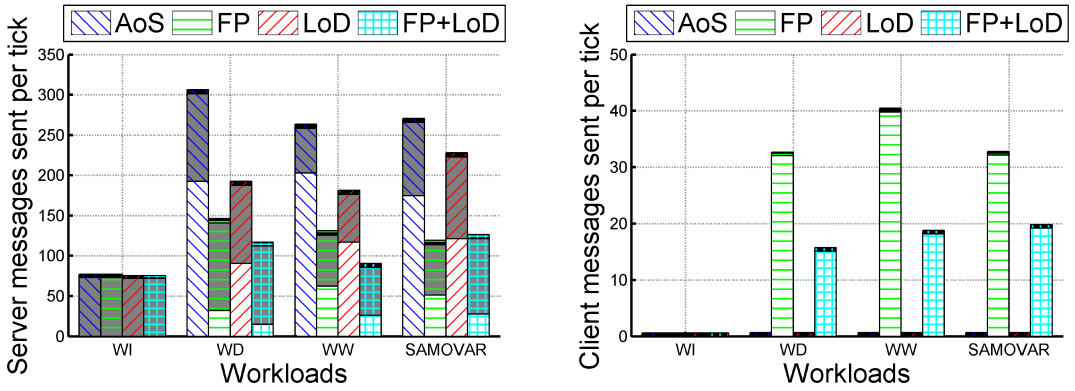


Figure 6.10: Network consumption for AoS when using state dissemination mechanisms to distribute states: (left) server; (right) client.

n. For the other workloads, on average, each player will experience about 4 UA-to-SA transitions per sub-map. As the UA-to-SA transitions may stall the game, which may dissatisfy players, we conduct experiments to evaluate the stall time of game-play using our prototype (see Section 6.5). The stall time includes the time taken to serialize the memory of a sub-map, to transfer the data to a client, and to create an SA based on the received data on the client side. The stall time is lower than 12 ms when the server and the client are within the same computer. The time used for transferring data over the Internet depends on the amount of data and network conditions. In our prototype, the amount of data of an SA is less than 0.1 MB, which is very low considering today's residential broadband bandwidth, so we expect the stall time will be low when the experiments are conducted on the Internet.

State dissemination mechanisms

To see whether the state disseminations we propose in Section 6.3.3 are efficient, we compare AoS (by default without any state dissemination mechanisms), with its variations that use state dissemination mechanisms: using forwarding pool (FP), using level of detail (LoD), and using both forwarding pool and level of detail (FP+LoD). When evaluating LoD and FP+LoD, $p = 10\%$ of avatars' state-updates are sent every tick while the others are sent every 40 ticks ($f_i = 0.025$), and $t_{LoD} = 150$.

Figure 6.10 (left) shows that the server upload bandwidth consumption can be significantly reduced by using these state dissemination mechanisms. Using FP can lead to about 50% to 60% lower upload bandwidth consumption under WD, WW and SAMOVAR, because FP makes use of the idle clients' upload bandwidth to transfer the states of UAs. Using LoD can lead to about 15% to 30% lower bandwidth consumption for the server, but at the cost that some avatars' state-updates are less frequently transferred (less accurate). By using FP and LoD together, the server upload can be further reduced by about 20% to 50%, compared to using FP only.

FP increases the clients' upload bandwidth consumption, as Figure 6.10 (right) shows, the clients' upload bandwidth consumption for FP and FP+LoD increases from less than 1 message per tick to about 30 to 40 for WD, WW, and SAMOVAR. Because LoD transfers fewer states, the clients' upload bandwidth consumption for FP+LoD is lower than that for FP. We conduct experiments on LoD with varying p, f_i and t_{LoD}. We find that higher p, higher f_i, and higher t_{LoD} lead to higher bandwidth consumption.

6.4.3 Consistency Evaluation

In Section 6.4.3 and 6.4.3, we evaluate the consistency tradeoff of AoS with a reference metric: the drift distance of avatars (measured in tiles). We show that *AoS introduces negligible inconsistency.* In Section 6.4.3, we evaluate the percentage of collisions between sub-maps.

The source of drift distance comes from the fact that when an avatar is crossing the border of a sub-map, instead of moving continuously, the avatar needs to first send a command to the area server of the neighboring sub-map where it will arrive; then the avatar will be created at the closest position connecting the current avatar's position when the issued command is executed at the neighbor sub-map. As the command can be delayed, the avatar's position may be slightly different in AoS, compared to EBLS. As the median value of the drift distance of all the experiments in this section is *zero*, we only show the 99.5 percentile of drift distance.

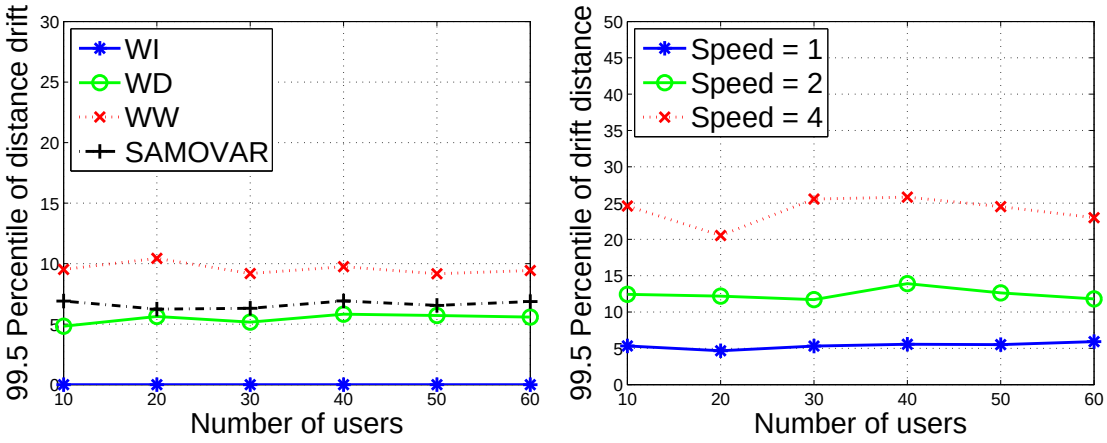


Figure 6.11: Drift distance of AoS with: (left) different workloads and (right) WD workload and different movement speeds.

Workloads and speed

Figure 6.11 (left) shows the results under four workloads. The drift distance is very small for each of these workloads. The drift distance for WI with varying number of users is zero, because in WI avatars do not cross any border. The drift distance for WW with varying number of users is about 10 tiles, as the movement speed of avatars is 1 tile per tick, this drift distance will eventually converge to zero within 10 ticks which is approximately 0.3 seconds. Figure 6.11 (right) shows the impact of movement speed: the higher the movement speed, the larger the drift distance is, but the drift distance can converge to zero within 0.2 seconds.

Partition

We measure the drift distance when partitioning the virtual world into 256×256, 128×128, to $64 \times 64 \, tiles$ sub-maps. The 99.5 percentile of drift distance is about 5 tiles for 256×256 sub-maps, it increases to be 20 tiles for 64×64 sub-maps. Finer grain of partition incurs higher drift distance, because finer partition will increase the probability of avatars crossing sub-map border.

Collisions

We evaluate the additional computational overheads needed to resolve the possible conflicting states by measuring the percentage of collisions between sub-maps. The percentage of collision between sub-maps is calculated by dividing the number of collisions between sub-maps by the total number of collisions. The percentages are low: for WI, WW, SAMOVAR, the percentages are below 0.5%, for WD, the percentage is below 4.5%. The

computational overhead needed to resolve in-precise collision detection depends on the method chosen to mitigate them, but given the low chance of collisions between sub-maps, the computational cost needed for resolving in-precise collision is low.

6.5 Real-world Experimental Results

To demonstrate the applicability of AoS, we implement the architecture described in the previous sections and deploy the working system in a *real-world* environment. We evaluate the working system with a prototype RTS game, which represents the large-scale multi-player extension of an open-source, single-player RTS game [89]. This prototype game features many common elements of RTS game: training avatars, constructing buildings, fog-of-war, battles, etc.

We conduct real-world experiments an order of magnitude larger load than the current state-of-the-art, up to 100 users (instead of 16 users in StarCraft II) and 5,000 avatars involved in a large virtual world. The results show strong evidence that our AoS-based system is scalable.

6.5.1 Experimental Setup

Implementation: our system implementation[2] has about 25,000 lines of C++ code, which add to the about 7,000 lines of C++ code of the original RTS game. The AoS mechanism needs about 3,000 lines of code, while the other 15,000 lines of code are used to implement the multi-user part for the original single-player game. Our network module follows a similar design as 0 A.D [1] and uses the reliable UDP library `ENet`[3]. We use the delta-encoding library `xdelta`[4] to encode the shared-zones' state to only transfer the difference of data, and `Zlib` for data compression. Without compression, the size of messages range from 40 bytes to 80 bytes. On average, the size of a command is 40 bytes, while the size of a state is 75. The size of messages lies within that of modern commercial RTS games [49].

Experimental environment: The experiments are conducted on the Amazon EC2 cloud. For our experiments, we use the "medium" instances of virtual machine (VM), each installed with Windows Server Datacenter edition 2008. Unless otherwise specified, we run 10 nodes (i.e., players) in each VM. All the other experimental configurations are the same as the default setup used in Section 6.4. Due to time and cost limitation, we use WI as the workload. As we focus on computing and networking resources, we disable the graphical output of the game.

[2] http://www.pds.ewi.tudelft.nl/~siqi/AoS.htm
[3] http://enet.bespin.org/
[4] http://xdelta.org/

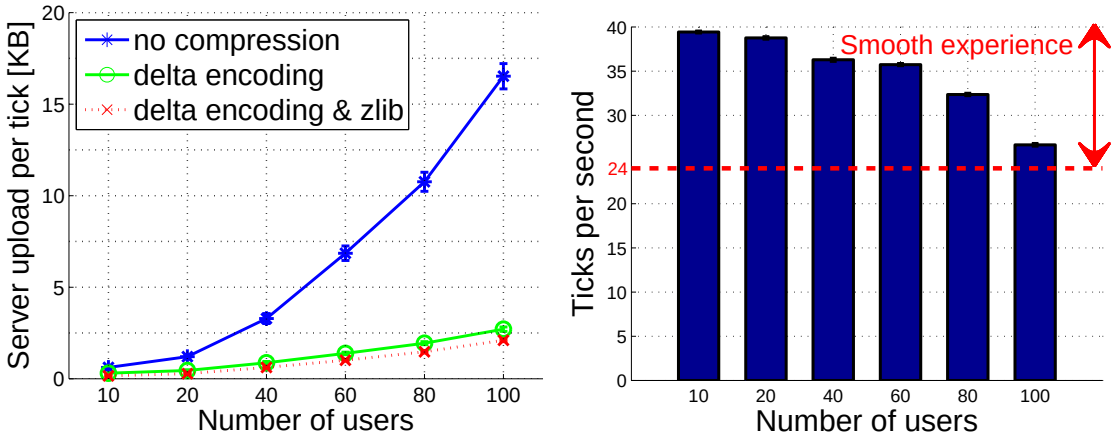


Figure 6.12: Upload bandwidth (left) and tick per second (right).

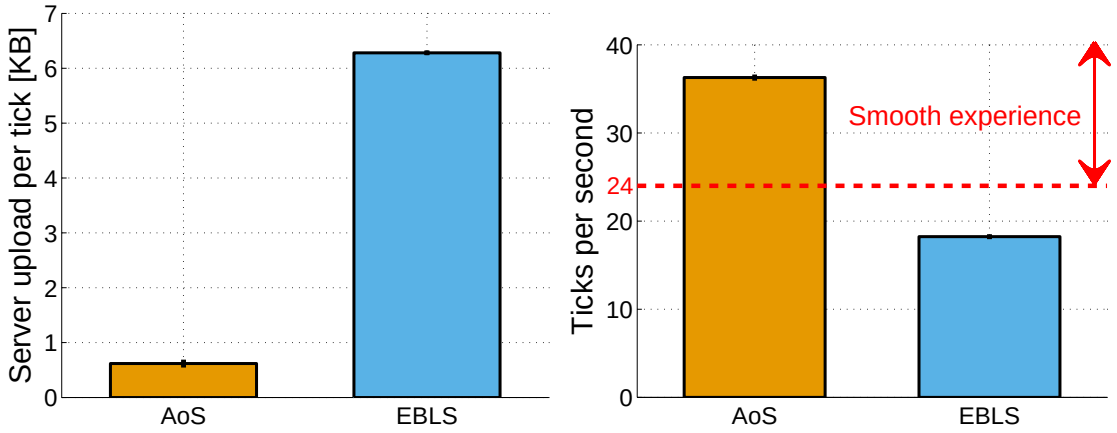


Figure 6.13: Comparison between AoS and EBLS: upload (left) and tick per second (right).

Performance Metrics: We measure and report the ticks per second of each experiment, the bandwidth consumption of the payloads of network messages sent by the server and number of messages sent. As the number of messages sent by servers matches well with the simulation results in Section 6.4, we only show the results for server upload bandwidth consumption and ticks per second (TPS). Each experiment is repeated 10 times, and the metrics shown are the average values.

6.5.2 Scalability Results

The bandwidth consumption of AoS scales lineally with the number of players. Figure 6.12 (left) depicts the upload bandwidth consumption of the server for an increasing number of users for different compression methods: AoS without compression, AoS

with delta encoding, AoS with delta encoding and zlib. Using delta encoding can reduce about 80% of the bandwidth consumption, by only transferring the differences between shared-zones' states. For WI, the shared-zones' states can use about 90% of the upload bandwidth. Using the delta-encoding mechanism we proposed in Section 6.3.4 can significantly reduce the amount of network bandwidth used. Using zlib can further reduce the upload bandwidth consumption by about 10%. Using delta-encoding and zlib, AoS consumes about 2.1 KB per tick, when the number of players is 100. As the simulation speed is about 26 ticks per second when the player count is 100, AoS consumes about 52 KB per second.

Figure 6.12 (right) shows TPS with varying number of players. With an increasing number of players, the simulation overhead increases which leads to TPS decreases. When the number of players is 100, the TPS is 26. All the TPS with different number of players are higher than the TPS required (24) to deliver smooth virtual experience, the user experience will not suffer.

To compare AoS with EBLS, we run 40-node experiments using AoS and EBLS. As EBLS requires more memory and CPU than AoS, we run 5 (instead of 10) nodes per VM when running the prototype using EBLS. Figure 6.13 (left) shows the network consumption for these two models. The server upload of AoS is about 13 times lower than EBLS; this value is slightly different from the message count result (there, AoS is about 15 times lower), because in the prototype a state-update uses more bytes than an event. Figure 6.13 (right) shows the TPS of the two models. Using AoS, the prototype can achieve a simulation speed of 36 TPS while EBLS can only achieve 18 TPS. The 18 TPS is below the minimal TPS (24) required to achieve smooth virtual world experience, so the user experience will suffer. *In summary, compared to EBLS, AoS can consume lower network bandwidth and achieve lower computation consumption while still fulfilling all the requirements of MAVEs.*

6.6 Related Work

There are two major operational models to update the NVEs in the clients' view: update-based and event-based. For the update-based model, the clients receive state-updates from the servers and then update their local view using the received information. For the event-based model, clients receive commands/events from a server and perform simulation, using the events to update the local game replicas. In this chapter, the NVE clients need to receive state-updates and may perform simulation, which is different from cloud gaming techniques such as [101], where the NVE clients only receive video streams from a server. In this section, we compare AoS with the research about scalability in Section 6.6.1 and consistency in Section 6.6.2.

6.6.1 Scalability Techniques

Much recent research explores scalable NVEs. We identify four main approaches: zone-based, object-based, server replication, and interest management.

Zone-based

The virtual world is partitioned spatially into multiple non-overlapping zones; each zone is assigned to a separate server [100, 124, 183]. Our *AoS*, SimMud [124], DSG [129], and MOPAR [233] partition the the virtual world statically. In contrast, VSO [100], Solipsis2 [79], Cell [61], and [140] partition the virtual world dynamically. As an example of zone-based NVEs, SimMud [124] partitions the game world into static zones and use an peer-to-peer multicast channel to send game updates to clients. Many previous studies focus on scalable messaging, and do not consider game-logic processing. For the studies that do, the operational model is mostly update-based.

Object-based

The virtual world objects are load-balanced across servers [139, 154, 224]. Each server is responsible for the simulation of a subset of objects (often called active objects), while the remaining ones (often called shadow objects), which are active in the other participating servers, are synchronized across servers. Proxy server [157] used for comparison in Section 6.4.2 belongs to this category.

Server replication

The virtual world states are fully-replicated at each server; clients connect usually to the closest server. The event-based model is usually adopted to reduce the network bandwidth. In this model, the events are transferred to some servers or broadcasted to all the servers, and the server performs the simulation based on events. EBLS is one of the most widely used server replication techniques adopted by NVEs such as [234] [52], and RTS games. As the states are fully-replicated, maintaining the state for a large virtual world is problematic for most servers.

Interest management (IM)

IM determines information that is interesting and should be received by players [231]. IM can be class-based or space-based. For class-based IM, users only receive specific types of information that are predefined per class, while space-based IM is based on proximity. AoI is a form of space-based IM in which a player only receives the information close to the location of the avatar(s) of the player. Usually, when an avatar moves, the AoI

moves with the avatar. An AoI can be a zone [124], a geometry area inside a zone [28], or intersects with multiple zones [233]. The shape of an AoI can be: tile-based [28], circular [3], and free format [19]. The size of an AoI can be static [100] or dynamic [117]. Donnybrook [19] propose an estimation of FPS players' interest based on distance between avatars, the aiming of the player's weapon, and interaction history. They classify avatars (based on interest) into two sets. Up-to-date states of the avatars in one set are received every frame while the states of the others are received every second. Different from previous work, we consider each player has multiple AoIs instead of one, and each AoI is operated using event-based or update-based model.

AoS is the first approach that combines zone-base partition, event-based and update-based models to support large-scale MAVEs, especially for RTS games.

6.6.2 Consistency Control

Pessimistic and *Optimistic* methods are two major classes of consistency control methods. *Pessimistic methods* anticipate inconsistency between data replicas when performing local actions. In contrast, *optimistic methods* assume no inconsistency exists and perform local actions instantaneously.

Pessimistic methods

The local-lag [145] mechanism delays the execution of operations to reduce the probability of the occurrence of inconsistency. It usually delay commands according to a system-level delay value. The local-lag mechanism works efficiently when the value is larger than the largest network latency. The two-phase-commit (2PC) method is a distributed transaction technique. It needs two round-trip time to finish a transaction which reduces the responsiveness of NVEs. Research such as [161] adopt the 2PC method to manage distributed data by synchronizing events that span multiple servers.

Optimistic methods

AoS can be classified as an optimistic method which allows that inconsistency exists in shared-zones. There are two major classes of optimistic methods: time warp (TW) and dead reckoning (DR). In TW, all replicas are allowed to execute update optimistically, and the method needs to record old states and roll back when inconsistency happens. Although there are improved version of TW such as trailing state synchronization [52] and [75], TW requires roll-backs, which may dissatisfy player; thus, we do not adopt this approach. DR, widely applied in distributed interactive simulation [104] and NVEs such as [19], is a technique to reduce the network consumption for position updates. DR can be used in AoS architecture to reduce the network consumption of UAs.

Other approaches exist. Lupei et al. [141] use software transaction memory technique (STM) to build a FPS game which runs in a multi-core machine. STM may have high overheads and transaction abort rate; thus, fine-level tuning of transactions and even re-design of game logic are needed. Krammer et al. [127] use multiple levels of consistency control for different objects. Tang et al. [216] use different update frequencies for different avatars to improve time-space consistency. Their work can be used in ours to resolve the inconsistencies.

6.7 Summary

Multi-Avatar Virtual Environments (MAVEs) such as RTS games entertain millions of people in small-scale, non-communicating game instances of only 8–16 players. To enable a future generation of MAVEs, in this paper we investigate a new mechanism and a system architecture built around it, which are scalable and have many desirable properties.

Our main contribution is five-fold. Firstly, we conduct the first empirical investigation into the presence of areas of interest in MAVEs. We find that, unlike the other virtual environments such as RPG and FPS games, in MAVEs users have multiple areas of high interest and that interest location changes quickly. Secondly, our AoS mechanism is novel in its use of update-based and event-based operation for areas of interest and provides a versatile scalability-consistency trade-off. For the latter, the AoS mechanism ensures that only areas of high interest are fully simulated, that areas of limited interest only receive infrequent updates, and that areas of no interest do not consume either computational or network resources. Thirdly, we propose an AoS-based system architecture for scalable MAVEs, which supports the dynamic management of multiple areas of interest and several more common, scalability-related techniques. Fourthly, we implement this architecture as a real-world system, which is able to run RTS games up to 100 users and 5,000 avatars in the same virtual world. Fifthly, we compare qualitatively and quantitatively our approach with various state-of-the-art approaches, and show strong evidence that AoS-based approaches offer superior performance and more flexibility for MAVEs.

Chapter 7

Scaling NVEs Efficiently through Cloud Scheduling

A growing number of applications are running in the cloud. Academia [16, 54, 77, 143, 158, 189, 226] and industry [227] are both increasingly using cloud resources as infrastructure to serve their users, due to the elastic, flexible, and pay-as-you-go features of Infrastructure-as-a-Service (IaaS) clouds. Cloud brokers need to lease resources from IaaS clouds cheaply, yet execute the users' jobs in time. To achieve this, cloud brokers must use scheduling policies that match diverse requirements. *Finding scheduling polices that can schedule diverse workloads with zero waiting time yet cheaply is the focus of this chapter.*

IaaS clouds offer their users various types of machine configurations: different amount of CPU cores, memory, and disk. It is non-trivial for a cloud broker to determine the combination of machine configurations for user demands. This situation is complicated by the current IaaS pricing models: machines configurations are not priced linearly with their performance. For example, an EC2 `large` instance can serve more web requests per core than the `small` instance, but their price per core is the same [192]. Moreover, for the same machine configuration, the clouds offer different billing options on-demand-, reserved-, and spot-instances, which are charged differently. Scheduling enough resources to meet user demands yet keep the cost low while adapting to workload changes remains challenging, despite recent research efforts [165, 192, 222].

In this chapter, we present a Cloud-based, online, Hybrid scheduling policy (*CoH*), which keeps the rental cost of cloud resources low by finding the best combination of machine configurations and billing options. At the core of this policy are its provisioning and allocation strategies. We formulate these strategies as Integer Programming Problems (IPP). As *CoH* needs to be executed online, the time to obtain a decision should be low. We limit the time to solve the IPP, and run simultaneously various heuristics. The *CoH* compares the result of IPP and heuristics, and picks the best one as its scheduling deci-

sion. Thus, a novel aspect of *CoH* is its portfolio-based scheduling strategy [103] adapted to IaaS clouds. Further, we devise, *CoH-R*, an extension of *CoH* to also makes use of reserved instances, which can lead to significant cost reduction compare to policies that use on-demand instances only.

The major contribution of this chapter is three-fold.

1. A novel online scheduling policy, *CoH*, which makes scheduling decision using a portfolio of IPP and heuristics-based approaches (Section 7.2).

2. A policy extended from *CoH*, *CoH-R*, which also makes use of reserved instances to reduce rental cost (Section 7.3).

3. An evaluation of our policies for two broad application domains, grid computing and online game hosting, using trace-based simulation (Section 7.4).

7.1 System Model

7.1.1 Workload and Resource Model

The workload model in this chapter is a set of independent jobs. The resource requirements and the runtime of each job are known when the job arrives in the system. Once started, jobs run to completion, so we do not consider task preemption or migration during execution.

Each job can be described by a tuple (r_i, a_i, d_i), where r_i is the resource requirement of job i, a_i is the arrival time of job i, and d_i is its departure time. We assume that a computer can host one or multiple jobs. This model is similar to the work of Stillwell et al [206]. This kind of jobs is common: a compute node can run multiple MapReduce tasks; an online game hoster may consolidate several game servers on the same machine; etc. The resource requirements of each job, r_i, could be a vector indicating multiple resource requirements (e.g., CPU and Memory), or a scalar value (e.g., CPU only). We focus on the CPU requirement. In practice, r_i can be obtained though profiling [108, 192] or can be provided by the user.

We model the operation and billing model of cloud providers based on the real case of Amazon EC2. We assume that clouds have infinite capacity. Each newly provisioned VM needs serval minutes to be booted [108, 165]. An VM is charged per hour; even a factional consumption of less than one hour is counted as one hour. An VM indexed by j, has capacity denoted by w_j and hourly cost c_j.

7.1.2 Scheduling model

In our scheduling model, all machines are provisioned exclusively from clouds. The cloud broker has pre-configured and stored in the cloud all the necessary VM images to run users' jobs. All the incoming jobs are enqueued into a queue. A system-level scheduler, running on a dedicated system, manages all the jobs and a pool of machines, and decides whether to provision new VM from clouds and/or to allocate jobs to VMs.

The scheduler is executed periodically (e.g., every 10 seconds). At each scheduling moment, the scheduler performs five tasks: (1) Predicting future incoming workloads; (2) Provisioning necessary VMs in advance, from clouds; (3) Allocating jobs to VMs, (4) Releasing idle VMs (which don't have job running on them) if its Billing Time Unit (BTU) is close to increase (e.g., 10 second before the leased hour). (5) If the wait time of un-allocated jobs is high, starting the necessary number of VMs. We design in the next section a scheduling policy, *CoH* to perform tasks (3) and (4). We further extend this policy in Section 7.3 to also use reserved cloud instances. As workload prediction is not the focus of this chapter, we assume that there exists a predictor that can achieve perfect prediction of future workload. Relatively good predictor [160] already exists for the type of workload we target in this chapter.

7.2 A Scheduling Policy using On-Demand Instances

This section describes *CoH*, a Cloud-based, online, Hybrid scheduling policy using on-demand instances. The strategy of *CoH* is presented in Section 7.2.1. *CoH* needs to take both provisioning and allocation decisions, that is, to find a combination of VMs, and a mapping between jobs and VMs. We formulate the above problem as an Integer Programming Problem (IPP) in Section 7.2.2 and then select various heuristics to assist *CoH* in Section 7.2.3.

7.2.1 Policy Overview

CoH actively provisions VMs before they are needed, and maps jobs to already provisioned VMs according to the best mapping it can find. *CoH* finds the combination of VMs and the mapping by solving an online scheduling problem through solving (partially) one IPP and by using several heuristics, independently and simultaneously. As an online scheduler, *CoH* needs to take scheduling decisions within limited amounts of time; thus, it limits the time used to solve IPP, and compares the result of the IPP and heuristics. *CoH* acts as a portfolio-based scheduler, in which multiple strategies are considered simultaneously at each scheduling moment. The strategy that has the best objective value (defined in formula (1)) is picked as the scheduling decision. Heuristics are needed be-

Table 7.1: Overview of notations in Section 7.2.

x_{ij_k}	whether job i is assigned to jth VM of type k, $x_{ik_j} \in \{0,1\}$
y_{j_k}	whether jth VM of type k is to-be-provisioned, $y_{k_j} \in \{0,1\}$
z_{ij}	whether job i is assigned to jth running VM, $z_{ij} \in \{0,1\}$
c_k	hourly cost of VM of type k
c_j	hourly cost of running VM j
w_j	capacity of running VM j
fc_k	full capacity of VM type k
r_i	resource consumption of job i
d_i	departure/end time of job i
s_j	The start time of VM j: the time when it started to boot
ld_j	latest departure time: the time that the final running job finish in VM j
ct	current time
M	number of newly arrived jobs, $\mathbb{M} = \{1, ..., M\}$
N	number of running VMs, $\mathbb{N} = \{1, ..., N\}$
K	number of types of VMs, $\mathbb{K} = \{1, ..., K\}$
$\lceil t \rceil$	math operation, divide time t by 3600 and get its ceil value.

cause the solution of IPP under limited time may be suboptimal or even infeasible (*CoH* may not find a feasible solution of the IPP in limited time).

7.2.2 Formalization of the Scheduling Problem

CoH needs to provision enough number of VMs to support all the incoming jobs, and to allocate all the jobs smartly such that the rental cost is minimized. An VM that has jobs running on it cannot be shut down, so it will still incur cost. Unnecessary cost will be incurred if long-running, low-resource requiring jobs are assigned to expensive VMs. We formulate the scheduling problem as follows. The goal of the scheduling problem, as defined in formula (1), is to minimize the cost while ensuring enough VMs for the incoming jobs. All the notations used in this section are listed in Table 7.1. An VM *to-be-provisioned* is identified by its identifier j and its type k, while a running VM is identified only by its identifier j.

Minimize

$$\sum_{k=1}^{K}\sum_{j=1}^{M}(y_{j_k} \times \lceil \max_{i\in\mathbb{M}}(d_i \times x_{ij_k}) - ct \rceil \times c_k) + R \tag{7.1}$$

$$R = \sum_{j=1}^{N}(\lceil (max\{\max_{i\in\mathbb{M}}(d_i \times z_{ij}), ld_j\} - s_j) \rceil \times c_j)$$

subject to

$$\sum_{i=1}^{M} z_{ij} \times r_i \leq w_j \qquad \forall j \in \mathbb{N} \tag{7.2}$$

$$\sum_{k=1}^{K}\sum_{i=1}^{M} x_{ij_k} \times r_i \leq fc_k \times y_{j_k} \qquad \forall j \in \mathbb{M} \tag{7.3}$$

$$\sum_{j=1}^{N} z_{ij} + \sum_{k=1}^{K} \sum_{j=1}^{M} x_{ij_k} = 1 \qquad \forall i \in \mathbb{M} \tag{7.4}$$

$$x_{ij_k} \leq y_{j_k} \qquad \forall i, j \in \mathbb{M}, \forall k \in \mathbb{K} \tag{7.5}$$

The cost of scheduling consists two parts: the cost of to-be-provisioned VMs and the cost of running VMs (defined by R). Each to-be-provisioned VM is charged between the current time (ct) and the latest departure time of its allocated jobs. The sum of the cost of running VMs, denoted by R, is defined similarly: each running VM is charged between the time it was started (s_j) and the latest departure time of its jobs (jobs that are running on VM and the jobs to-be-allocated to it).

This IPP is subject to a few constrains, which we describe in turn. Constraint (2) ensures that the allocated jobs in each VM cannot exceed the running VMs' capacity. Constraint (3) ensures that the allocated jobs in each *to-be-provisioned* VM can not exceed the VM's capacity. Constraint (4) ensures that each job is only allocated to one VM. Constraint (5) ensures that each job will not be allocated to a VM that will not be provisioned. The decision variables x_{ij_k} and y_{kj} are binary. If the result of this IPP is that $y_{kj} = 0, \quad \forall k \in \mathbb{K}, \quad \forall j \in \mathbb{M}$, there will be enough VM capacity left to allocate all the future jobs. Otherwise, more VMs are needed. If $x_{ij_k} = 1$, job i will be allocated to the to-be-provisioned VM with identifier j type k.

7.2.3 Scheduling heuristics

We explore for *CoH* a large class of scheduling heuristic algorithms. They work as follows. While there are un-allocated jobs, each algorithm performs a loop consisting of four steps. Firstly, the algorithm sorts all the un-allocated jobs using *job selection* criteria and sorts all the VMs using *VM selection* criteria. Secondly, the algorithm picks the first un-allocated job. Thirdly, it picks the first VM which should have enough capacity left for the job. And then allocate the job to the selected VM. If such an VM does not exist, a new VM is provisioned according to *VM type selection* criteria and the job will be allocated in the next loop.

This general class of heuristic algorithms uses three criteria: *job selection*, *VM selection*, and *VM type selection* criteria. All *job selection* and *VM selection* criteria used in this chapter are listed in Tables 7.2 and 7.3. For *VM type selection*, we use a Cost-Efficient heuristic, which always chooses the VM with the largest $capacity/cost$ value.

Most of the selection criteria we use in this chapter are simple, which allows them to be run online. We describe some of the criteria below. Latest arrival time (LA) sorts the VM according to the latest arrival time of jobs in each VM in decreasing order. Opposite to LT, Earliest arrival time (EA) sorts the VM by earliest arrival time of jobs in increasing order. Similar to LT, Latest departure time (LD) picks the VM which has the job that

Table 7.2: Job selection criteria.

Name	Description
FCFS	First-come-first-server
RR	round-robin
LJF	Largest job first
SJF	Smallest job first
LTJF	Longest run-Time job first
STJF	Shortest run-Time job first

Table 7.3: VM selection criteria.

Name	Description
Rnd	Random
LM	Largest capacity VM first
SM	Smallest capacity VM first
LA	Latest arrival time
EA	Earliest arrival time
LD	Latest departure time
ED	Earliest departure time
LAA	Latest average arrival time
EAA	Earliest average arrival Time
CFH	Close to Full Hour
FFH	Far from full hour

has the latest departure time; Earliest departure time (ED) does the opposite. The latest average arrival time (LAA) and earliest average arrival time (EAA) sorts VMs according to the average arrival time of their jobs in decreasing and increasing order, respectively. Close to full hour (CFH) makes use of the billing model of EC2; it always puts jobs on VM whose Billing Time Unit (BTU) is closest to be increased, while Far from Full Hour (FFH) is the opposite. In this chapter, the scheduling heuristic method specified by its job and VM selection criteria is uniquely identified as $\{$*job selection*$\}$-$\{$*VM selection*$\}$. For example, the FCFS-Rnd heuristic uses First-Come-First-Server (FCFS) for job selection, random (Rnd) for VM selection and the cost-efficient criteria for VM type selection.

7.3 A Scheduling Policy using Reserved and On-Demand Instances

Cloud providers allow their users to reserve VM instances, long-term, for reduced cost. For instance, Amazon offers *reserved* instance, which can be rented for 1-3 years for a lower price than their on-demand counter-parts. When using reserved instead of on-demand instance, for the same VM configuration, users can pay a higher upfront cost (UF_i) for a lower hourly cost (C_i). Currently, there are three types of reserved instances supported in EC2: lightly utilized, medium-utilized, and heavily utilized reserved instances. For the lightly utilized and medium-utilized instances, users need to pay an upfront cost and pay for each hour the VM is running. For the heavily utilized instances, users need to pay an upfront cost and pay for each hour during the reserved term even if the VM is not running. The hourly cost of Amazon EC2 instances are listed in Table 7.5. We present *CoH-R*, an extension of *CoH*, which uses reserved instances to reduce

the operational cost. We describe the strategy of *CoH-R* in Section 7.3.1, then describe the method used to determine the number and types of reserved instances to be used in Section 7.3.2.

7.3.1 Policy Overview

Assuming that it is given an arbitrary amount of reserved instances, *CoH-R* makes use of these reserved instances as follows: the heavily utilized instances are always on, while the medium and lightly utilized instances are shut-down when they do not have any jobs running on them before their Billing Time Unit (BTU) is about to increase (m seconds before the BTU increases). Whenever *CoH* plans to start a new VM of type k, *CoH-R* firstly looks at medium-utilized instances of type k, and starts one of them if any is off. If no medium-utilized instance of type k exists, *CoH-R* tries to use a lightly utilized instance of type k. As a last resort, *CoH-R* uses on-demand instance of type k.

Having too few reserved instances will not benefit much from the reduced price; while reserving too much may actually increase operational cost. We do not seek to find the optimal number of reserved instances, because obtaining the optimal solution requires exact workload information of the entire reservation period (e.g., one year). Even if we can know the workload of the upcoming time period, obtaining the optimal solution via solving an IPP that takes the exact workload as input is computationally infeasible.

7.3.2 Determining the reservation plan

CoH-R only requires the workload distribution instead of exact information of the number of VMs needed at each time interval. For simplicity of analysis, we assume VM start-up and shut-down time are instantaneous (In experiment, we set the start-up time as two minutes.). Assuming the number of VMs (resource demand) needed for the current time interval t is D_t, we can obtain the cost needed at each interval $B(D_t)$ as follows. If D_t is lower than the number of heavily-utilized instances (N_3), no other VMs will be needed, as the heavily utilized instances can provide enough computing resources. If D_t is high than N_3, but lower than the total number of heavily and medium-utilized instances ($D_t \leq N_3 + N_2$), *CoH-R* needs to provision $D_t - N_3$ medium-utilized instances. If D_t is higher than the sum of heavily and medium-utilized instances ($D_t > N_3 + N_2$) but lower than the total number of reserved instances, *CoH-R* needs to provision all the heavily and medium-utilized instances and $D_t - (N_3 + N_2)$ lightly utilized instances. Last, if D_t is higher than the sum of all reserved instances, *CoH-R* needs all the reserved instances and $D_t - (N_1 + N_2 + N_3)$ on-demand instances.

CoH-R obtains the number of reserved VMs needed, of each type, via finding the combination of number of reserved instances (N_i) that minimizes $\sum_{t=1}^{T} B(D_t) + \sum_{k \in \mathbb{K}} (UF_k \times$

N_k), where T is the number of intervals (e.g, hour) of a time period (e.g, year or month). Further, if the resource demand of each interval does not affect the other time intervals (in practice, most of the jobs' runtime is short, in the order of tens of minutes), the goal can be reformulated via only using the probability of VMs needed at each time interval as below, $(\sum_{i=0}^{M} Pr(D = i) \times B(i)) \times T + \sum_{k \in \mathbb{K}} (UF_k \times N_k)$, where $Pr(D = i)$ is the probability distribution of demand, and $\mathbb{K}$ is the set of reserved type, and M is the maximal number of VM needed.

We extend the above method which deal with one machine configuration only, to allow it to deal with multiple machine configurations. The goal is to find the number of reserved instances (N_{jk}) of machine configuration j and reserved type k, needed, to minimize the cost defined in formula (7.6).

$$(\sum_{i=0}^{M} Pr(D = i) \times B(i)) \times T + \sum_{j \in \mathbb{J}} \sum_{k \in \mathbb{K}} (UF_{jk} \times N_{jk}) \tag{7.6}$$

In formula (7.6), $\mathbb{J}$ is the set of machine configurations and UF_{jk} is the upfront cost of reserved instance of machine configuration j and reserved type k. The billing function $B(D)$ need to be changed to be the lowest cost to meet the demand D via finding the combination of reserved and on-demand instances to be used. $B(D)$ ={*Minimize* $\sum_{j \in \mathbb{J}} \sum_{k \in \mathbb{K}} (n_{jk} \times c_{jk}) + \sum_{j \in \mathbb{J}} (n_j^{od} \times c_j^o)$} , where n_{jk} and c_{jk} are the number and the cost of the reserved instance with machine configuration j and reserved type k, respectively. n_j^{od} and c_j^o are the number and the cost of the on-demand instances of machine configuration j, respectively. The capacity offered by n_{jk} reserved instances and n_j^{od} on-demand instances should be enough to satisfy demand D.

7.4 Experimental Results

In this section, we evaluate the performance of our proposed approaches using multiple real-world traces corresponding to two separate but popular domains: grid computing and online game hosting. Firstly, we compare *CoH* against various commonly used heuristics. Then, we evaluate *CoH-R*, which uses reserved instances to further reduce cost, and compare it to *CoH*. Our results indicate that our proposed approaches can lead to significant lower cost than heuristics.

7.4.1 Experimental setup

We conduct experiments using three real-world workloads *LCG*, *Grid5000*, and *Dotalicious* which are taken from public workload archives [73, 92, 107]. *LCG* and *Grid5000* contain information about the computing activities of two grids while *DotaLicious* con-

tains workload information of a game platform. We use the first year traces *Grid5000* and *Dotalicious*, and the full trace of LCG (13 days) as our input workloads. The common data we find in the above traces are, for each recorded job, its job id, the arrival time, and the departure time. The basic statistics of these workloads are listed in Table 7.4. Notably, the gaming server have similar runtime (CPU requirement) to Grid5000 jobs in the order of tens of minutes. Game servers are also computationally intensive, a result of having to perform virtual world physical simulation.

As not all the traces contain resource requirements for each job, we generate for each job resource requirements using 3 different methods: Heterogeneous, Constant-100, and Constant-10. For Heterogeneous workload, we generate each jobs's resource requirements as ten times a random number which is between 1 to 10. For Constant-100 method, each job's resource required is 100. For Constant-10 method, each job's resource required is 10. We only consider two instance types: `small` and `large`. We model a `small` EC2 instance's capacity as 100 and a `large` instance's capacity is 410. `Large` instance is more cost efficient than `small` instance. Their cost[1] are summarized in Table 7.5.

As running all the heuristics online is time consuming, we evaluate the heuristics by running simulation and pick the heuristics that have good performance as alternative method to compete with the solution obtained by solving IPP. We find that none of heuristics can perform always best, across all scenarios, and find that the job selection criteria does not have a significant impact on cost but VM selection criteria does have an important impact on cost. We pick FCFS-SM when the input workload is heterogeneous, and use FCFS-LD and FCFS-CFH when the workload is homogeneous (Constant-10 and Constant-100).

All the experiments are conducted using our own simulator and repeated at least 10 times. We set the acquisition time of an VM to two minutes and the scheduler is executed every 10 seconds. We use IBM CPLEX to solve the formulated IPP when the number of jobs to be scheduled is lower than 50 and set the time limited as two seconds. As our methods have proactively provision VMs for all the jobs, the wait time of each job is zero. We evaluate one metric, the rental cost. The rental cost is the price paid to cloud providers for all the rented computing resource. We focus on cost because it is a major barrier for cloud adoption.

For calculation of the utility of all the methods, we compare the lower-bound for cost against actually paid cost. The lower bound for cost is calculated by assuming that we have an ideal computer that it can vertically scale to the any of the desired capacity. The vertical scaling takes zero time and the VM is charged by its actual usage of resource which scales linearly with its capacity. So the optimal cost can be computed as $[\sum_{i=1}^{N} r_i \times (d_i - a_i)] \div w_k \times c_k, \quad \forall i \in \mathbb{N}$, where N is the total number of jobs, and w_k and c_k are the capacity and the cost of the most cost-efficient VM, respectively.

[1] http://aws.amazon.com/ec2/pricing/

Table 7.4: Overview of traces.

Trace	#jobs	average runtime [s]	duration	source
Grid5000	200,450	2728	May 2004 - May 2004	Grid workload archive [107]
LCG	188,041	8971	Nov 2005 - Dec 2005	Parallel workload archive [73]
DotaLicious	109,251	2231	Apr 2010 - Apr 2011	Game trace archive [92, 93]

Table 7.5: Overview of cost of EC2 instances: Small and Large.

	Small (hourly, upfront) [$]	Large (hourly, upfront) [$]
On demand	(0.065, 0)	(0.26, 0)
Lightly utilized	(0.039, 69)	(0.156, 276)
Medium utilized	(0.024, 160)	(0.096, 640)
Heavily utilized	(0.016, 195)	(0.064, 780)

7.4.2 Results

We first evaluate *CoH* against various heuristic methods. Figure 7.1 shows the average experiment results using *Grid5000* and *LCG* datasets, respectively. The error bars are the standard deviation. Figure 7.1 shows the lower bound for cost (LB), and results for FCFS-SM, FCFS-CFH, FCFS-LD, and *CoH*, from left to right; grouped by type of workloads. We find that *CoH* performs better than any of the heuristics. For the *Grid5000* dataset, *CoH* can obtain about 20% to 40% lower cost than any heuristic. For the *LCG* dataset, *CoH* can obtain 5% to 20% lower cost. This indicates that CoH can find better combinations of VMs, and better mapping between jobs and VMs.

The cost obtained through *CoH* is about 1.1 to 1.6 times higher than LB. The utilization of *CoH*, that is, the average use of leased VMs, ranged from 90% to 63%. We identify three reasons why *CoH* is higher than the lower bound (LB): Firstly, our scheduler is run online, thus not having all the necessary information. Secondly, the billing model of the cloud: a fractional consumption of a VM's capacity is charged as the fully busy VM. Thirdly, the boot-up time of VM is not negligible. One possible way to lower the gap between LB and *CoH* is to allow the jobs to wait for better scheduling opportunity, so that scheduler can pack more jobs in the same VM instead of starting a VM for each short job. This approach would be particularly effective during bursts in the workload.

We evaluate *CoH-R* using the *Dotalicious* and *Grid5000* datasets. The results are shown in Figure 7.2. We do not evaluate *LCG* dataset because it lasts for only 13 days (less than the minimal reservation period of EC2). We compare in Figure 7.2: FCFS-CFH, *CoH*, CoH-oneType, and *CoH-R*. CoH-oneType is a variation of *CoH-R* which only uses heavily-utilized instance. For the *Dotalicious* dataset, *CoH-R* and CoH-oneType obtain lower cost than *CoH*. *CoH-R* can obtain lower cost than CoH-oneType, because it takes advantage of the cost reduction and flexibility provided by different reserved types. The result obtained by *CoH-R* using the *Dotalicious* dataset is about 13% to 20% lower than *CoH* and about 30% to 60% lower than FCFS-CFH. For the *Grid5000* dataset, the performance of *CoH-R* obtain about 3% to 5% lower cost than *CoH*, but still about 20% to 50%

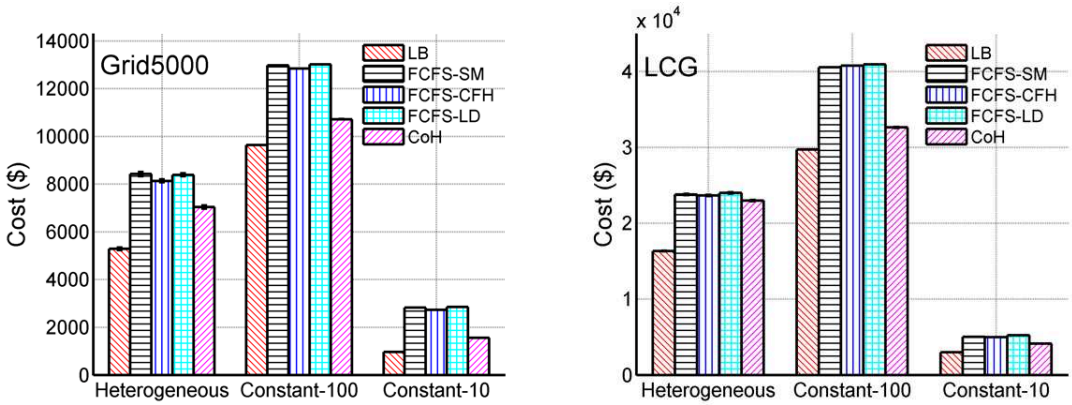


Figure 7.1: Cost of various scheduling methods: (left) Grid5000 and (right) LCG.

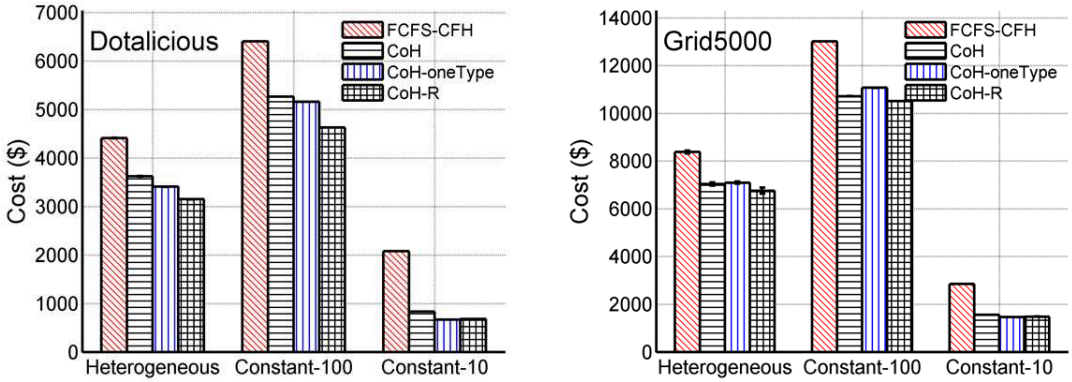


Figure 7.2: Effect of using reserved instances: (left) Dotalicious and (right) Grid5000.

lower cost than the heuristic. The reason why *CoH-R* only obtains a small improvement on *Grid5000* is because *Grid5000* contains busty workloads with short jobs, and some very long jobs. As *CoH-R* always schedules jobs to VMs as soon as the jobs arrive in the system, this cause some long jobs to run on on-demand instances instead of the cheaper reserved instances. In summary, *CoH-R* can obtain about 20% and up to 60% lower cost than the heuristic. *CoH-R can obtain significantly lower cost than heuristics which use on-demand instances only.*

7.5 Related Work

A significant body of work has already focused on cloud resource scheduling from a cloud provider's perspective [95,178,219,236]. In this context, the common goals are to reduce the storage/electricity cost and to improve platform utilization. In contrast, in this study

we schedule resources from a broker's perspective, with the goal to minimize the rental cost.

Previous studies have focused on provisioning and allocation of cloud resources, under various constraints. In contrast to these studies, which we describe in the following, we consider multiple instance types, billing models and heterogeneous workload. Closest to our work, Genaud and Gossa [82] evaluate provisioning heuristics for on-demand resources. Villegas et al. [222] conduct a performance-cost analysis of scheduling policies for IaaS Cloud. Deng et al. [59] develop a portfolio scheduler. Oprescu and Kielmann [169] schedule bag-of-tasks on clouds focusing on budgets and runtime. They formulate the provisioning problem as a Bounded Knapsack Problem and allocate jobs to VMs round-robin. Mao et al. [142] propose a linear program for provisioning, and allocate jobs randomly to VMs. Sharma et al. [192] use on-demand instances and use migration but only for homogeneous workloads.

Hong et al. [98] use a method to determine number of reserved instances of one reservation type. We show in our experiments that it is necessary to use multiple reserved instance types to reduce cost. Chaisiri et al. [38] propose an algorithm to determine the number and types of reserved VM to be used by solving a stochastic IPP to minimize expected cost. They limit the on-demand instances can be only provisioned in specific provision phase, while we proactively provision VM at any necessary time. Ostermann and Prodan [170], and Song et al. [204] use spot-instance to reduce cost. Their work complement ours.

7.6 Summary

It is challenging to select among machine configurations and billing options offered by clouds to fit user demand while reducing operational cost. In this chapter, we propose *CoH*, a Cloud-base, online, Hybrid scheduling policy which make uses of multiple machine configurations to plan enough capacity for users with less cost. We formulate the resource provisioning and the job allocation problems as Integer Programming Problems (IPP). To obtain the scheduling decision online, *CoH* limits the time of exploration for a solution and only obtains an intermediate IPP solution. *CoH* makes scheduling decision by picking the best among the solution of IPP and various heuristics; thus, *CoH* operates as a portfolio scheduler. Further, we propose *CoH-R*, a policy that makes use of both on-demand and reserved instances to reduce cost. Via simulation using real-world traces, we show that our approaches can lead to significant lower cost than heuristics while operating online.

Chapter 8

Making NVEs Robust through the Availability-on-Demand Mechanism

Increasing amounts of datacenter resources provide the infrastructure of ICT utilities at global scale [189, 213]. Datacenter users rent datacenter resources to provide diverse ICT utilities, from business-critical processes [43] and scientific computing [57], to social networking [15] and online gaming [160]. Due to the sheer scale of datacenters, resource failures are bounded to happen [36, 188]. When failures occur during critical service periods, such as during flashcrowds [8, 23], during periodic collection of results, or at the end of service operation (such as just before the outcome of an online game match), they are likely to lead to significant revenue loss or customer departure [112, 136]. Over the past decade, many high availability (HA) techniques have contended for masking resource failures [138, 185], but they can be costly and difficult to manage when applied indiscriminately. Moreover, datacenters and even public Infrastructure-as-a-Service clouds offer today to their users only limited management options for dynamically selecting and configuring HA techniques. In this chapter, we propose Availability-on-Demand (AoD), a mechanism for dynamic HA management comprised of an API to dynamically specify availability requirements and a configurable availability-aware scheduler.

Managing HA techniques effectively is non-trivial. First, many HA techniques exist, including recent virtualization-based techniques such as Active/Active (AA) and Active/Standby (AS) [138], which are increasingly adopted in large datacenters and commercial datacenter products [53, 223]. Second, the impact of resource failures on revenue is difficult to estimate. Anecdotal evidence [112, 136] indicates revenue loss: even the small, sub-second delays in generating the response to a customer query can lead to significantly fewer sales (1% for Amazon) and overall site traffic (up to 20% for Google). All HA techniques increase administrative costs and human resource needs [181], and may incur significant costs in redundant infrastructure. Thus, we ask in this chapter the research question *How and when to use HA techniques effectively inside the datacenter?*

We answer our main research question by designing and analyzing experimentally Availability on Demand (AoD), a HA-aware mechanism for dynamic datacenter resource management. Novel in this chapter, we consider for our mechanism the class of ICT services where the availability requirements, and thus the utility of using HA techniques, *can change over time*. In contrast to mission-critical applications, such as online-banking transactions, which require HA during their entire lifespan, datacenter-supported services such as business support, some types of scientific computing, and online games require HA only during limited periods of time. For example, a company may want to run its support services with HA only during working hours, an online service may increase its HA requirements during launch or after major updates, an online gaming service may require higher HA during the end of important matches (e.g., the final of the World Cup of e-Sports League of Legends), etc.

We further design our mechanism to provide support for the specification and management of HA in datacenters. We propose an easy-to-use API that allows datacenter users to specify dynamically the availability levels they need. Users can express their availability requirements over time, and for entire services or for parts of their service, e.g., only for the master component of a master-worker application. We also propose an availability-aware scheduler which tries to balance availability and the cost it incurs. We equip this scheduler with a scheduling policy which manage computing resources dynamically to meet user-specify availability requirements.

We evaluate our mechanism experimentally, through trace-based simulation. Using the API, we express dynamic availability requirements for a variety of workloads. We also conduct comprehensive, trace-driven, simulation-based experiments that compare the proposed scheduler with several alternative approaches. To give evidence on the versatility and efficiency of our mechanism, our experiments use long-term traces representative for two important and popular application domains, scientific computing and online gaming.

The main contribution of this chapter is twofold:

1. We propose a novel mechanism, Availability on Demand, which manages the dynamic HA-requirements of datacenter users (Section 8.2). The mechanism consists of an API for datacenter users to specify dynamic availability requirements, a scheduler that manages resources while trying to improve the availability of the system, and a policy to configure the scheduler.

2. We evaluate our mechanism experimentally, through trace-based simulation (Section 8.3). Our results indicate superior performance for our mechanism, in contrast to approaches which use HA techniques indiscriminately or naively. Moreover, comparing to an ideal approach which use perfect failure predictions, our approach can lead to 13% to 31% more cost, but with similar availability for critical parts of

applications.

8.1 System Model

The system model we consider in this chapter is common for datacenter studies and follows our work [58, 196], which it extends with a consideration of failures derived from [106, 215, 237]. We describe, in turn, the infrastructure, the workload, the operational, the failure, and the HA elements of the system model used in this chapter.

8.1.1 Infrastructure and Workload Model

We consider datacenters who provide Infrastructure-as-a-Service (IaaS) or managed-IaaS (Platform-as-a-Service like) cloud services. Datacenter users (*customers*) express their ICT services (*applications*) as workload units (*jobs*) that run on virtual machines (VMs) rented from the datacenter. VMs are hosted by the datacenter on homogeneous physical hosts (e.g., blade servers) owned by the datacenter.

Jobs can consist of one or multiple tasks, where a task can be a typical Linux process or a VM. For the IaaS model, customers submit each task in the form of a VM to the datacenter, and the datacenter will allocate the VM to some host. For example, users can control tasks that are running in VMs provided by Amazon's EC2. For managed-IaaS, customers submit their jobs to the datacenter, and let the datacenter allocates VMs and run the jobs. There are two types of tasks: primary and backup. Primary tasks are tasks that execute the application logic, while backup tasks are used to protect primary tasks to avoid interruption of ICT services.

We only consider tasks for which the CPU is the dominant resource, that is, the time to execute a task is inversely proportional to the performance of the processor it runs on (e.g., SPEC CPU). We allow dependencies between tasks, and specifically consider two traditional computational models: master-slave (MS) and bag-of-task (BoT). For MS-type dependency, if the master task fails, the whole job fails; if any of the slave-tasks fails, it will not affect the other tasks. For jobs with BoT-type dependency, individual task failures do not lead to the failure of the entire job. We do not consider MPI-type applications. For those applications, should any of the tasks fail, the whole job fails.

Match-based games, such as the Defense of the Ancients, are examples of bag-of-task jobs. Matches are independent from each other. MapReduce applications are a type of master-slave workload. The master-task of a MapReduce application monitors and controls the slave-tasks to perform some data-analysis. We only consider the availability of the jobs themselves, not the systems that jobs rely on. If those systems fail, the jobs fail too. For example, for a master-slave application such as MapReduce, if the Hadoop

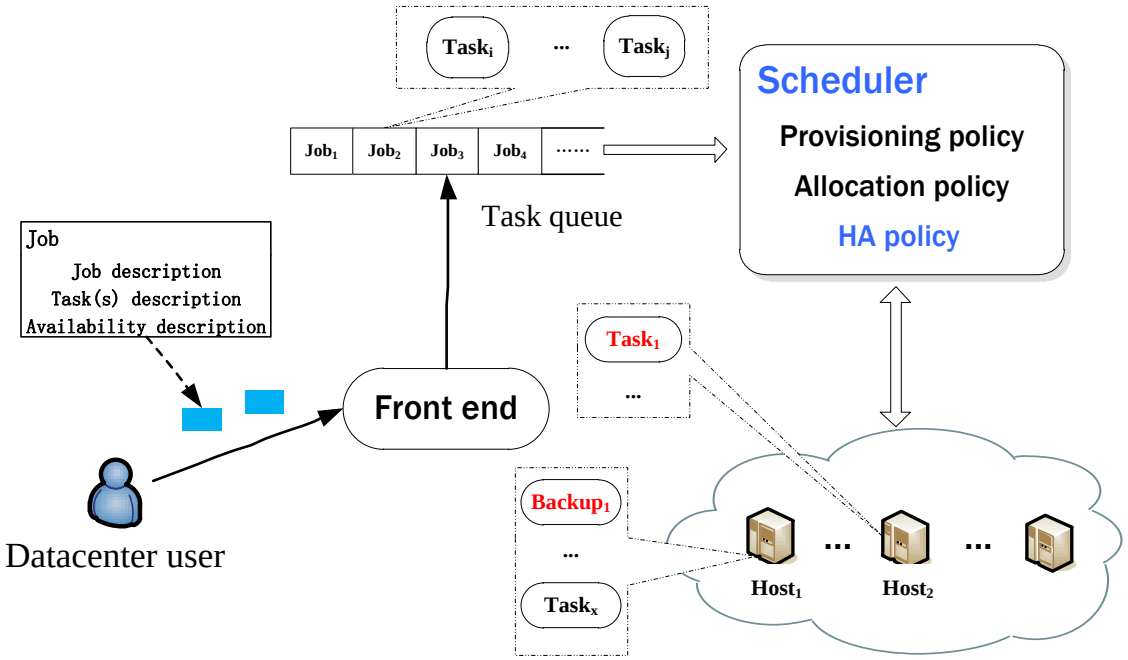


Figure 8.1: Schematic Plot of the Operational Model.

Distributed File System fails, the MapReduce application fails, even when equipped with our mechanism (introduced in Section 8.2).

8.1.2 Operational Model

The operational model is depicted in Figure 8.1. Users submit their jobs to the frontend of a datacenter. Each job contains the necessary information needed to execute the job, and the availability requirement of the job. All incoming jobs are enqueued into a system-level queue. A system-level scheduler, running on a separate physical host, manages all the jobs, a pool of physical machines, and a pool of virtual machines. The scheduler decides whether to boot up physical and virtual machines, and whether to allocate tasks to hosts.

The scheduler use an HA policy to manage backup tasks which is used to protect normal tasks. As is depicted in Figure 8.1, $backup_1$ is running in host_1 to protect $task_1$ which is running in host_2. The scheduler uses a provisioning policy for booting up hosts from the datacenter, and an allocation policy to select jobs or tasks and to allocate them to hosts.

Provisioning Policy

For each task, the provisioning policy will try to place the task to the first host which has enough capacity. If the task is a backup task, the host should not contain both the primary and the backup tasks of the same task. If the task cannot be placed on any of the running hosts, a new host will be booted.

Allocation Policy

The allocation policies will first select a job, according to the First-Come-First-Serve (FCFS) policy, and then allocate all of the tasks of that job to some hosts. An allocation of a job is successful only if all of its tasks can be allocated to hosts. For each task t, the allocation policy will place the task on the eligible host with the least number of idle CPUs. A host is eligible if it has enough capacity and it is not executing the primary task of t (if t is a backup task).

8.1.3 Failure Model

We assume that physical hosts fail according to the *fail-stop* model: once a host fails, all the VMs hosted on the physical host stop and fail. Failures adhere to the model proposed in [106]: as in traditional failure models, once a failure happens to a physical host, the physical host will be down for a while, then resume normal operation. Similarly to [215], when a failure happens and cannot be masked by the HA technique of the datacenter (see Section 8.1.4), the failing tasks that ran on the host are resubmitted to the system-level queue and start from their beginnings.

We do not address other error models [11]. As failure-detection is not the focus of this chapter, we also assume that there exists a failure detection mechanism which can detect the fail-stop failures timely with perfect accuracy.

8.1.4 High Availability Model

We consider in this chapter one main HA model and its practical technique that can be used at the VM level of the datacenter: the Active/Active (AA) technique.

The AA technique masks single failure occurring to individual VMs (and their service), by using a *backup VM* is running in parallel with a *primary VM*, so the two VMs operate as active replicas of each other. If a failure happens to one of active replicas, the other active replica takes over. If both active replicas fail, the service fails. There are many ways to achieve the AA technique, including synchronous methods such as lockstep [223], which execute the exact instruction and data at each step; asynchronous methods such as Xen Remus [53], which replicates its state asynchronously to the backup

active replica; and hybrid methods such as COLO [65], which synchronizes the replicas only their outputs differ significantly.

Dynamically adding or removing AA replicas for a primary VM is already enabled by current virtualization techniques [53, 114, 223]. Dynamically adding an AA replica can be achieved by the following procedure. First, the virtual machine monitor initializes live migration (e.g. as in [114]). Instead of terminating the primary VM at the end of the migration, the replicated VM will stay synchronized with the target VM using mechanisms such as [53, 223].

Other HA models exist. Among them, we have considered but not explored in this chapter the Active/Standby (AS) model, which recovers a failed VM from a booted standby VM. In contrast to AA, AS ensures slower recover speed, but at the cost of only a standby, rather than active, resource.

8.2 Availability On Demand

In this section, we propose the Availability on Demand (AoD) mechanism for the specification and management of HA in datacenters. The main requirement for our AoD mechanism is to support services for which individual service components (tasks) can have time-varying availability requirements. The mechanism includes an easy-to-use API to specify HA requirements and an HA-aware scheduler.

Our key innovation is the support for dynamic HA requirements, which promises to provide high availability with low cost (use of computational resources). Traditional approaches do not support the dynamic specification of HA for each service component, and maintain replicas for each service and for the entire duration of the service. In contrast, the AoD API enables the dynamic specification of requirements per service component, and the AoD scheduler uses replicas only for *selected* services (tasks) and only *temporarily, when needed*.

We describe, in turn, the API by which the users can specify their dynamic HA requirements (Section 8.2.1), and, in detail, the availability-aware scheduler (Section 8.2.2) and its policy (Section 8.2.3). Last, we discuss the implications and limitations of our approach (Section 8.2.4).

8.2.1 A Customer API for Specifying Availability Requirements

We propose an API for customers to specify the dynamic availability requirements of their applications, and per job or task. Our API is easy-to-use, in that it allows users to specify their requirements through a single, three-parameter API call. To achieve this, we consider in this chapter two levels of availability, *high* and *normal*; normal availability does not provide any HA model, whereas high availability is supported through the AA

technique. The API provides a single function, the same for both per-job and per-task specifications:

`SetAvailability(id, availability, time period)`

with the parameters: "id", through which users can specify the unique id of the job or task which requires different levels of availability; the "availability" field, to specify the availability level, normal (`NA`) or high (`HA`). Users can specify the period for which the availability requirements expressed in the API call should be valid, by using the "time period" field. By default, the specified availability requirement apply to the entire life cycle of the tasks; the "time period" field is then set to `all`. In this chapter, we use the terms "critical period" and "high availability period" interchangeably to describe the period which requires high availability.

The AoD API, albeit simple, is expressive. First, it supports many types of availability changes, including three main models we consider in this study:

- *Bursty*: most of the time, the availability requirement of the task is normal, but can raise at any moment to high.
- *Periodical*: the availability requirement of the task changes over time, alternating between normal and high availability periods.
- *Steady*: the availability requirements of each task is set to normal or high and does not change over time.

Second, it offers support for a variety of application domains, including the following examples:

- For MS applications (see Section 8.1.1), which are common in scientific computing, the master component is more important than the slave-tasks. Users wishing to provide HA for these applications could specify this such kind of requirement by making a single, task-level API call: `SetAvailability(MasterId, HA, all)`. (The calls of `SetAvailability(WorkerId, NA, all)` represent the default, so they are not required.)
- For online gaming applications, many of which are BoT applications (see Section 8.1.1), the availability requirement may be higher between 9PM to 1AM (after dinner to late-night play). The users can specify this requirement through a single, job-level API call: `SetAvailability(gamingAppId, HA, 9PM→1AM)`.

In this chapter, we focus on the simplicity of the API to make it easy to understand and to be sufficient to meet our initial requirement. The API can be further improved by adding more features. For example, the API can include an option to be used to specify the number of backups, as more backups can ensure better availability. As another example,

Algorithm 2 AoD scheduler, main execution cycle.

```
1: while not end of scheduling do
2:     Managing backup tasks; //Section 8.2.3
3:     Removing backup tasks;
4:     Allocating backup tasks; //Section 8.2.3
5:     Enqueuing tasks for scheduling;
6:     Provisioning VMs;
7:     Allocating tasks to hosts;
8:     Turning off idle hosts;
```

the API can be extended to allow customers to specify their desired availability target, and then our system will give the customers recommendations, for example based on expected cost.

8.2.2 AoD Scheduler

In this section, we propose the AoD scheduler—a datacenter-level scheduler that is HA-aware and tries, through the novel HA policy we will introduce in Section 8.2.3, to support the requirements specified by datacenter users through the AoD API. The AoD scheduler is configurable, in the sense that each policy used by the scheduler can be selected by the user from a library of available policies. We assume that availability requirements are provided by calls to the API at the moment when the jobs are submitted to the datacenter.

The function of the scheduler is to manage the process of booting up or turning off physical hosts, of starting or stopping VMs, and of allocating tasks, while taking into account HA requirements and enforcing them through the AA technique (see Section 8.1.4). In this chapter, we only use AA backup tasks, which use AA technique to create backup for the primary task. For briefly, we refer to AA backup tasks as *backup tasks*.

The AoD scheduler consists of a main execution cycle, executed often (e.g., every second). The main steps of the schedulers are depicted in Algorithm 2. They are:

1. *Managing backup tasks* The scheduler creates backup tasks for the running tasks (detailed in Section 8.2.3).

2. *Removing backup tasks* The scheduler removes the backup tasks that are not longer needed, for example because their high availability period has just ended.

3. *Allocating backup tasks* The scheduler allocates backup tasks to physical hosts (detailed in Section 8.2.3). It is possible that some backup tasks cannot be allocated due to lack of computing resources. Those tasks will be put into the system queue and be processed later.

4. *Enqueuing tasks for scheduling* Newly arrived normal tasks, failed tasks, and backup tasks are submitted to the system queue for scheduling.

5. *Provisioning necessary computing resources* by turning on (booting up) enough hosts (see Section 8.1.2 for the provisioning policy).

6. *Allocating tasks to hosts* by creating a VM for each task of a job (see Section 8.1.2 for the allocation policy).

7. *Turning off idle hosts* to save operational cost. A host will be turned off if it has been idle for k minutes (e.g., 2 minutes).

8.2.3 An AoD High Availability Policy

HA policies used in this chapter determine the behaviors of the scheduler about how backup tasks should be created, executed and terminated. We propose an HA policy to manage backup tasks: AoD based on user-specified availability *R*equirements (*AoD+R*). The AoD+R policy creates backup tasks based on the availability requirements provided by the customers (described in Section 8.2.3). All the backup tasks created will be allocated to hosts to be executed using an allocation approach described in Section 8.2.3. The AoD+R policy terminates a backup task if HA is not longer needed for its primary task.

Management of Backup Tasks

A distinctive feature of the AoD+R policy is the management of backup tasks, which for our AoD mechanism are not running all the time and for all tasks, but temporarily and only for selected tasks. The AoD+R policy relies on the availability requirements provided by customers. Each time the policy is invoked, it works as follows. For each task t in the running task set (T_R), if t needs HA for a certain period of time, an AA backup replica (t_{aa}) is generated for t and added to the set of AA backups (T_{aa}). A backup task for a master-task of a MS-type job will run during the entire lifespan of the master-task, whereas backup tasks (t_{aa}) for non-master tasks (e.g., slave-tasks) will only run until the end of the HA periods; at the end of this period, t_{aa} is marked for removal by being moved into the removal set (T_K) which will be removed by the scheduler.

Allocation of Backup Tasks

In this section, we describe how the AoD+R policy allocates backup tasks present in the backup task set T_{aa} which is created in step 1 of Algorithm 2. The scheduler takes into account task characteristics of backup tasks (different runtime, different resource

Algorithm 3 AoD allocation heuristic.

Input: $t_{aa} \in T_{aa}$ all the AA backup tasks.
H_{on} on-line hosts.
c_t the resource consumption of task t.
c_h the remaining resource capacity of host h.
h_t the host where task t locates.

```
1: calculate {G_{t_aa}} for each t_aa ∈ T_aa;
2: sort {G_{t_aa}} in decreasing order;
3: for t_aa ∈ T_aa do
4:    for host h ∈ H_on do
5:       if c_{t_aa} ≤ c_h and h_t ≠ h then
6:          allocate t_aa to h;
7:          h_{t_aa} = h;
8:    if t_aa cannot be allocated then
9:       T'_aa = T'_aa ∪ {t_aa};
```

consumptions, etc.) and tries to maximize the availability gain (an availability-aware utility metric, defined in the following) achieved by allocating tasks to different hosts.

The goal of the allocation is to find a subset of T_{aa}, and to allocate them to hosts H, so that the availability gain is maximal. We denote by $G_{t_{aa}}$ the availability gain of backup task t_{aa}, where $G_{t_{aa}} = E_{t_{aa}} \times I_{t_{aa}}$, with $E_{t_{aa}}$ being the already executed time of t_{aa}'s primary task t, and $I_{t_{aa}}$ being the relative importance of the primary task t. The intuition behind $E_{t_{aa}}$ is that more gain is ascribed to the tasks that have been executed the longest. If the job has an MS-type dependency, and t is a master task (see Section 8.1.1), we model $I_{t_{aa}}$ as the total resource consumption of the job containing t—intuitively, if the master task t fails, the whole job fails. For all other tasks of all other job types, if t requires HA at the time when the allocation algorithm is invoked, $I_{t_{aa}} = 1$, otherwise 0.

The goal of maximal availability gain can be formulated as maximizing $\sum_{t_{aa} \in T_{aa}} G_{t_{aa}}$, subject to two constraints which are formulated as follows.

Resource Constraint: The amount of resources allocated to tasks in a host h cannot exceed the remaining resource capacity of h.

$$\sum_{h_{t_{aa}}=h} c_{t_{aa}} \leq c_h \quad t_{aa} \in T_{aa}, h \in H \tag{8.1}$$

Anti-colocation Constraint: The AA replica t_{aa} of task t cannot be placed in the same host as t.

$$h_{t_{aa}} \neq h_t \quad t_{aa} \in T_{aa}, t \in T_R \tag{8.2}$$

where h_t indicates which host the task t locates, $h_{t_{aa}}$ denotes where t_{aa} will be lo-

cated, and T_R is the running task set. The formulated problem is an integer programming problem (IPP). As most IPP are NP-hard, we do not seek to obtain the optimal solution for the above IPP we defined. We propose a heuristic algorithm to obtain a feasible, online allocation of tasks to hosts. The heuristic algorithm is depicted in Algorithm 3. First, it will obtain the availability gain for each task (line 1). Second, it will sort all the tasks in T_{aa} according to their gain $\{G_{t_{aa}}\}$ in decreasing order. Third, for each backup task t_{aa}, the algorithm will try to allocate the task to the first host h which has enough capacity and does not run the primary task t of t_{aa} (lines 3-9). For the tasks that cannot be allocated, they will be organized as T'_{aa} (lines 10-12). The T'_{aa} will be inserted into the system queue and be processed using the provisioning and allocation method described in Section 8.1.2.

8.2.4 Implications and Limitations of the AoD Mechanism

There are several ways to improve the AoD mechanism. One of the possible extensions is to use both the AS and AA models (see Section 8.1.4). Using standby backup tasks can reduce the resource consumption incurred by active backups, while keeping the downtime of applications low (but longer than for active backups). Another possible extension is to use both customer-specified availability requirements and failure predictions, to further reduce the cost of offering availability.

There are several practical limitations to our work in this chapter. First, although the overhead of AA replicas can be very small [65,223], in practice the AA technique may not work efficiently for multi-core VMs [223], and may not be efficient for memory intensive VMs [65]; in both cases, workload interference leads to decreased performance. An approach to solve this problem in practice is to run benchmarks statically or dynamically, to determine whether it is efficient to use AA techniques. Second, the AoD+R policy does not work efficiently for MPI-like applications. Checkpointing may be a better solution for those applications, but also faces many open challenges [27, 45].

8.3 Experimental Results

In this section, we evaluate our AoD scheduler equip with the *AoD+R* policy (see Section 8.2.3), and compare them with four alternatives. We use for this realistic trace-based simulation, using as input long-term, real-world traces that represent scientific computing and online gaming. *Our results indicate that the AoD+R policy can achieve high availability with low cost, in comparison to policies that use AA techniques randomly and an AoD+R policy variation. Moreover, compared to an ideal policy which use perfect failure predictions to manage backup dynamically, the* AoD+R *policy can consume 13% to 31% higher cost but with similar availability for critical parts of applications.*

We describe, in turn, the setup of our experiments (Section 8.3.1), the alternative approaches (Section 8.3.2) and the metrics used for comparison (Section 8.3.3), and the main results for the usability (Section 8.3.4) and performance (Section 8.3.5) of the AoD mechanism. Overall, we evaluate 5 scheduling policies under different scenarios: 2 task dependency models (MS and BoT) and 3 availability requirement models (bursty, periodical, and steady), and with different parameters. Unless otherwise specified, the default task dependency model is MS and the default availability requirement model is bursty.

8.3.1 Experiment Setup

Infrastructure

The experiments shown in this section are conducted using an event-based simulator developed for this study. The simulator is based on CloudSim [35] and our work on cloud simulation [58,196]. We simulate a datacenter which consists of 1000 hosts, with 16 CPU cores each. These values are realistic for a medium cluster, but also in the range of the systems that provided the traces described in Table 8.1. Scaling these traces, for much larger or much smaller systems, is difficult for various theoretical reasons [78].

Workloads

To indicate the versatility of our AoD mechanism, the real-world workload traces used in this chapter represent two application domains, scientific computing and online gaming. Table 8.1 presents an overview of these traces. The KTH-SP2 trace comes from the Parallel Workload Archive (PWA) while the DAS2 trace comes from the Grid Workload Archive (GWA). The DLI trace contains the first year records of the DotaLicious trace from the Game Trace Archive (GTA). As the DLI trace does not specify the number of CPU cores used per job, so we assume that each job uses 1 CPU core.

We do not have real user-defined availability requirements. Instead, we use the following synthetic formulation. For the bursty model, a randomly continuous time period ($k\%$ of the task duration) is picked as a critical period which requires high availability, while the other period is set to be the normal period which requires normal availability. For the periodical model, the task runtime is partitioned into multiple half-an-hour periods; then, for each of the period, the first $k\%$ of the period requires high availability, while the remainder requires normal availability. For the bursty and periodical model, high availability requirement period(s) are generated for a task only if the task's runtime is longer than 10 minutes. For the steady model, $k\%$ of the tasks need high availability all the time, while the other tasks only need normal availability. In default, k is set to be 30. In this chapter, the workload model specified by its task dependency and availability model is uniquely identified as $\{$*task dependency*$\}$-$\{$*availability*$\}$. For example, MS-Bursty means

Table 8.1: Overview of real-world traces. "Sci.comp." and "Onl.Gam." stand for scientific computing and online gaming, respectively.

Trace Type	Trace name	#jobs	Avg. runtime [s]	Avg. CPU	Trace source
Sci.comp.	KTH-SP2	28,489	8876	7.7	PWA [73]
Sci.comp.	DAS2	219,618	530	10.3	GWA [107]
Onl.Gam.	DLI	109,250	2232	1	GTA [93]

MS task dependency and bursty availability model.

Failure Generation

When generating failures, the time and duration of a failure are determined according to [106]; and then randomly one or two hosts will fail. The inter-arrival time of failures are generated using a Weibull distribution ($\alpha = 9.7$, $\beta = 12.2$), and the duration of failures are generated using a LogNormal distribution ($\mu = 2$, $\delta = 0.26$). To determine which hosts fail, we assign different failure probabilities to different hosts [237]. The failure probabilities follows a Zipf distribution (exponent r range from 0 to 1); the hosts with a larger failure probability will experience more failures. In this chapter, we set $r = 1$, this leads to more failures happening to some hosts.

8.3.2 Alternative Policies for Comparison

We compare the AoD+R policy against four scheduling policies:

- *None*: This policy does not use any HA techniques.
- *Rnd*: This policy will use the AA technique to improve the availability of all the jobs: for each task it will have a $k\%$ (i.e., 30) probability to add an AA backup task which runs for the entire duration of the job.
- *AoD-I*: This policy is a variation of the AoD+R policy. The AoD-I policy does not distinguish between master-tasks and slave-task and treats them equally, that is, if the task (either a master-task or a slave-task) needs HA at the time when the mechanism is invoked, it assigns $I_{t_{aa}} = 1$, otherwise $I_{t_{aa}} = 0$
- *Pred*: This policy is used as a reference to measure the gap between the AoD+R and optimal scenarios. It assumes the existence of a predictor which tells about when and where each failure will happen, the policy will create backup tasks for the tasks located on hosts predicted to fail. The *Pred* policy also informs the allocation module to stop allocating tasks to hosts predicted to fail, for an amount of time (e.g., 10 minutes) or, if a downtime predictor exists, until the predicted end of the failure.

We explore in this chapter only the ideal case in which the location of failure is perfectly predicted and the accuracy of the moment when failure happens is within 10 minutes.

8.3.3 Metrics

Each experiment is repeated at least 20 times. The results reported in this section are average values. We consider the following metrics:

- *Number of critical failure events (CRITS)* The number of failure events during periods which require high availability. This metric indicates the ability of the system to protect applications during the periods that matter, that is, when the customers could be willing to pay extra for high-availability guarantees. The lower this metric, the better.

- *CPU hours* The total number of hours that the CPUs in the datacenter are used by the customer. This metric is useful to assess the efficiency of an availability approach; less is better.

- *Number of failure events (FAILS)* The number of failure events, including failures during periods require normal or high availability. The lower this value, the better, but this metric may be misleading, because failures during normal availability periods may not be important enough (for example, it may not be user-facing). Similar to CRITS and CPU hours, lower values mean better results.

For the three metrics, the CRITS metric emphasizes the importance to protect applications during critical periods. The CPU hours metric evaluates the cost-efficiency of a scientific computing and online gaming system, and the FAILS metric measures the availability of a system. In this chapter, we only evaluate the above three metrics, more metrics could be used to evaluate the effectiveness of our approach. However, it requires future work.

8.3.4 Expressiveness Results

We apply the availability API proposed by AoD (in Section 8.2.1), in practice, for specifying the availability requirements of each task in the input workloads, for various scenarios.

We generate the availability requirements of each task according to the availability models we described in Section 8.2.1 and the amounts we have described in Section 8.3.1.

Overall, we conclude that the API can express the diverse workloads used in this chapter: 2 application domains, 2 task dependency models, and 3 availability models.

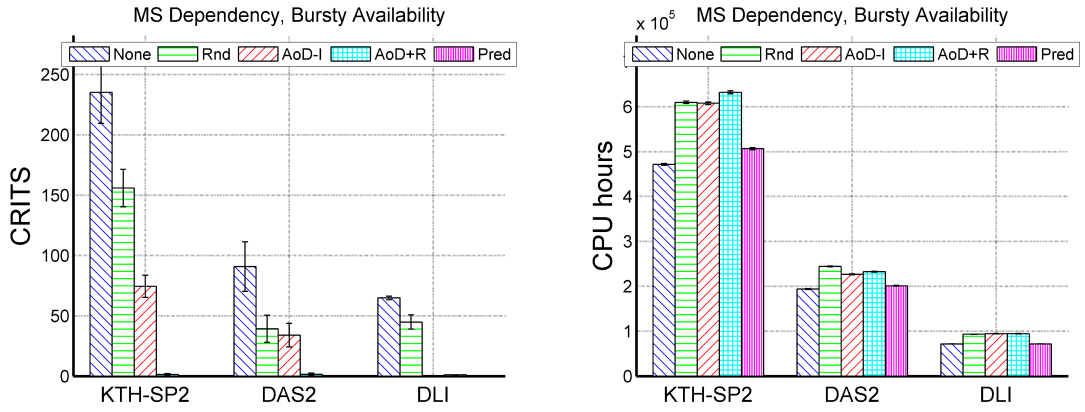


Figure 8.2: Results under the MS task dependency and bursty availability requirement model: (left) number of critical failure events (CRITS) and (right) CPU hours. (the non-visible bars represent zeros.)

8.3.5 Performance Results

In this section, we show the results under different task dependency and availability requirement models. The main findings are:

1. The AoD+R policy work well for the MS and BoT task dependency models.

2. The AoD+R policy consumes about the same CPU hours as the Rnd and the AoD-I policy, but has significantly lower CRITS. Moreover, the AoD+R policy can lead to less FAILS than the RnD policy and the AoD-I policy.

3. Comparing to the ideal policy: Pred, the AoD+R policy consumes 13% to 31% more CPU hours, but about the same CRITS.

MS task dependency with bursty availability requirement model (MS-Bursty)

Figure 8.2 (left) shows CRITS for the None, the Rnd, the AoD-I, the AoD+R, and the Pred policy, from left to right; grouped by traces. As is shown in the figure, the None and the Rnd policy have much higher CRITS than the other policies. The None policy has the highest CRITS, because it does not employ any HA techniques to protect tasks. The AoD-I has at least 50% lower CRITS than the Rnd policy, but the CRITS for the AoD-I under the KTH-SP2 and the DAS2 trace are non-negligible. The AoD+R, the Pred policy has the lowest CRITS. This shows that the AoD+R policy satisfies the design goal to protect applications at important occasions.

Figure 8.2 (right) shows the CPU hours metric for all the policies. As expected, the None policy consumes the least CPU hours, as it does not use any HA techniques which

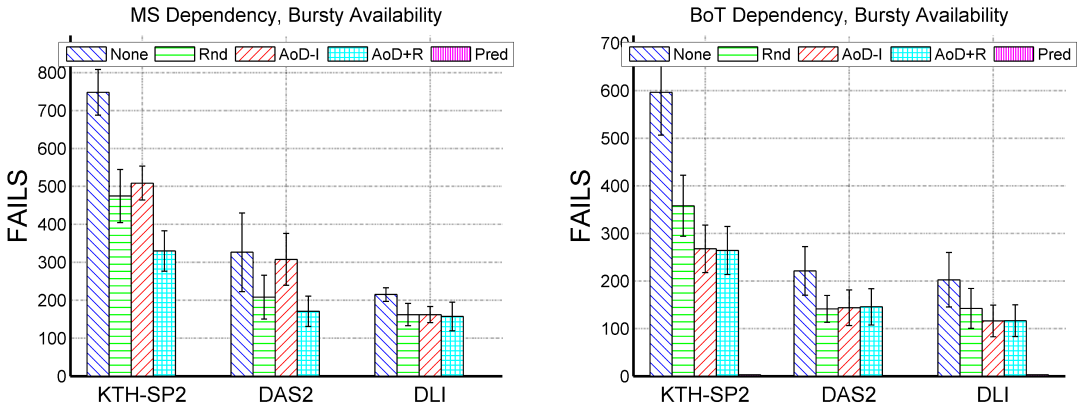


Figure 8.3: Number of failure events (FAILS) results under the : (left) MS-Bursty and (right) BoT-Bursty workload models. (the non-visible bars represent zeros.)

use additional computational resources to execute tasks. The Pred policy consumes the second lowest CPU hours. The AoD-R policy consume 15% to 30% more CPU hours than the Pred policy. In addition, the AoD-I policy consumes more or less the same amount of CPU hours as the Rnd policy. Moreover, the AoD-I policy consumes a bit less (about 5%) CPU hours than the AoD+R policy. This is because the AoD+R policy create AA backups for master tasks during their overall runtime instead of only during HA periods.

Figure 8.3 (left) shows the FAILS metric for all the policies under the MS task dependency and bursty availability model. The Pred policy has the least FAILS, as it predicts occurrences of failure and create AA backups to protect tasks located in physical hosts which will fail. The AoD+R policy has the second least FAILS, about 60% and 10% less FAILS than the Rnd policy under the KTH-SP2 and the DAS2 trace, respectively. The FAILS of the AoD-I policy are about 40% higher than AoD+R policy for the KTH-SP2 and the DAS2 traces. This because the AoD+R policy protect master tasks of jobs by creating AA backup tasks for all the master tasks, when failure happens to a master task, it will be protected as the master task has AA backup which is running in another physical host. The FAILS of the AoD-I and the AoD+R policy are identical for the DLI trace, because each job only contain one task in the DLI trace, thus the performance of the two policies are the same.

BoT task dependency with bursty availability requirement model (BoT-Bursty)

As is shown in Figure 8.4 (left), the AoD+R policy has about the same CRITS as the Pred policy, and significantly less CRITS than the None and the Rnd policy. Comparing to the MS-Bursty workload, the CRITS metric for the BoT-Bursty is less. This is because for the MS-Bursty workload, a failure of the master task will trigger failures of tasks of the same job. For the CPU hours metric, as is shown in Figure 8.4 (right), the Rnd, the

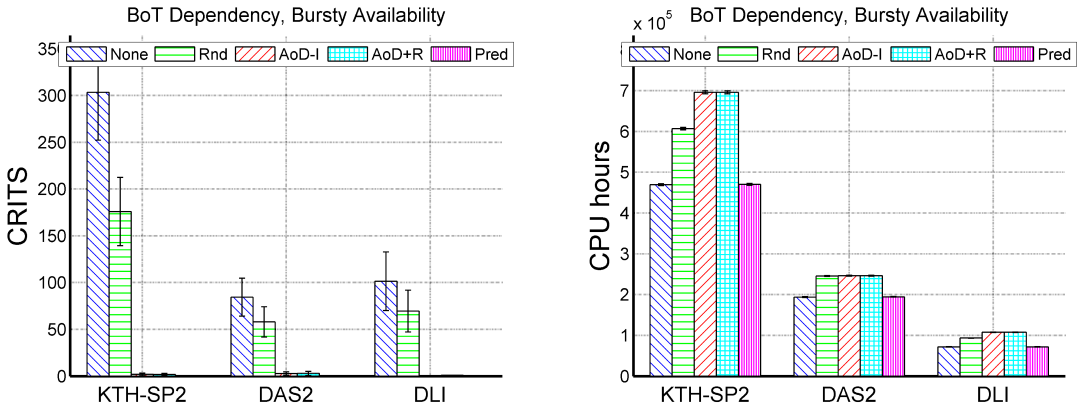


Figure 8.4: Results under the BoT task dependency and periodical availability requirement model: (left) number of critical failure events (CRITS) and (right) CPU hours.

AoD-I, the AoD+R policy consumes about the same CPU hours. And they consume 15% to 25% higher CPU hours than the Pred policy. The FAILS of the policies for the BoT-Bursty workload is lower than the FAILS for the MS-Bursty workload due to the reason we explain before. As is depicted in Figure 8.3 (right), the AoD+R and the AoD-I policy consume about the same FAILS as the Rnd policy.

For MS and BoT task dependency models with periodical and steady availability model: MS-Periodical, MS-Steady, BoT-Periodical and BoT-Steady, we obtain similar experimental results. The AoD+R policy has similar CRITS to the ideal policy: Pred. And the AoD+R policy consumes similar amounts of CPU hours to the Rnd policy, but leads to significantly lower CRITS. For tasks with MS dependency, the AoD+R policy can lead to significantly less FAILS than the Rnd and the AoD-I policy. In comparison with the Pred policy, we find that the AoD+R policy has more FAILS, but importantly similar CRITS and only 13% to 31% higher CPU hours.

Impact of changing the percentage of HA periods

We evaluate the AoD+R policy by varying the percentage of HA periods (k), from 10% to 50%. With increasing k, the duration of HA periods for each task increases. The results of this set of experiments are depicted in Figure 8.5. As is shown in Figure 8.5 (left), the CRITS metric stays low (≤ 8) for the AoD+R policy with increasing k. This indicates that the AoD+R policy can protect applications even for high percentages of HA periods. For the CPU hours metric, as is depicted in Figure 8.5 (right), the CPU hours consumed by the AoD+R policy increase linearly with k. This is because with the increasing k, AA backup tasks will run longer to protect primary tasks, which leads to increased, but only linearly, CPU hours. As Figure 8.6 (left) shows, the FAILS metric decreases linearly with increasing k. This is because when the duration of HA periods increase, AA backup tasks

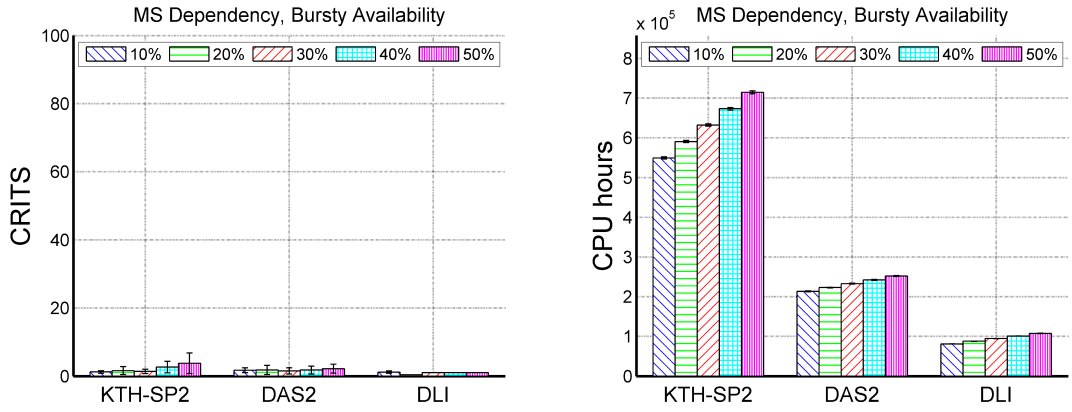


Figure 8.5: AoD+R with various percentage of HA periods: (left) number of critical failure events (CRITS) and (right) CPU hours.

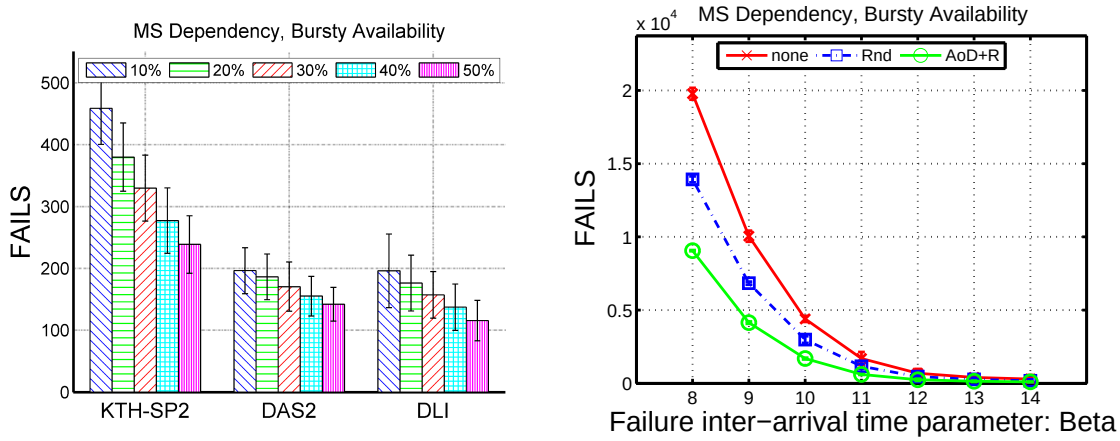


Figure 8.6: Number of failure events (FAILS) for AoD+R with: (left) various percentages of HA periods and (right) various frequencies of failures.

will run longer to protect their primary tasks longer.

Impact of changing the frequency of failures

We evaluate the impact of frequency of failures for the None, the Rnd, and the AoD+R policy by changing β which determines inter-arrival time (IAT) of failures. The smaller β is, the smaller the IAT is, which leads to in turn more failures. As is depicted in Figure 8.7 (left), the CRITS metric for the AoD+R policy remains low (≤ 5) for various values of β, whereas the None and the Rnd policy has very high CRITS metric for small values of β, and the metric decreases with increasing β (decreasing failure frequency). This suggests that the AoD+R policy can protect applications during important moments, regardless of the frequencies of failures. For the CPU hours metric, as Figure 8.7 (right)

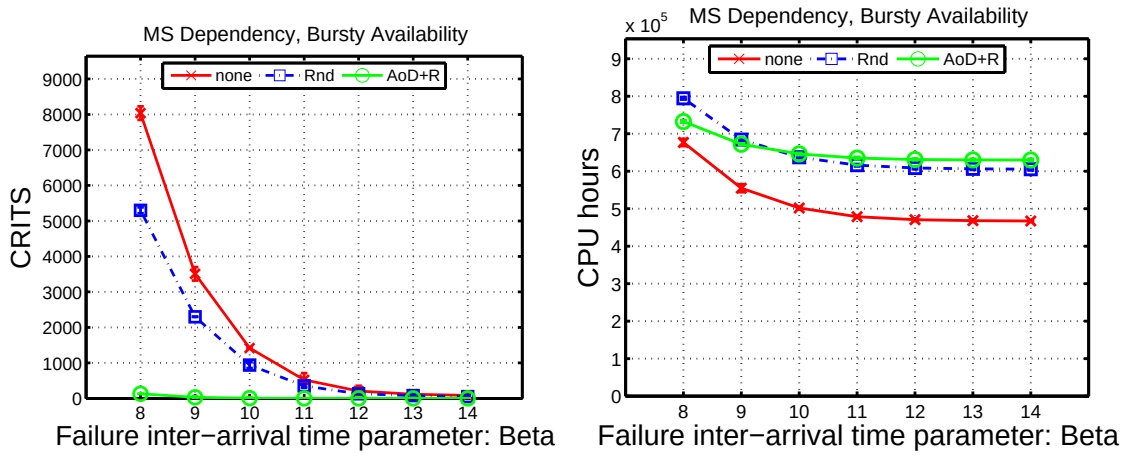


Figure 8.7: AoD+R with various frequencies of failures: (left) number of critical failure events (CRITS) and (right) CPU hours.

indicates, the None, the Rnd, and the AoD+R policies consumes slightly less CPU hours with reducing failure frequencies (increasing β). This is because with less failures, less tasks are needed to re-executed. For the FAILS metric, according to Figure 8.6 (right), the FAILS metric for the three policies decreases significantly with increasing β.

8.4 Related work

A large number of research efforts have been devoted to improve the efficiency [5, 41, 62, 94] and availability [46, 110] of distributed systems. Hardware-based techniques, which employ redundant power-facilities, cooling-facilities, switches, network links, and storages [47], have been proposed to improve the availability of distributed systems. In this section, we focus on comparing our work of this chapter with software-based high-availability techniques. Two of the most common software-based techniques are checkpointing and replication [36].

For checkpointing-based approaches: Researchers propose to use proactive and preventive checkpointing to improve the efficiency of HPC system [27]; to reduce the storage space and overhead of checkpointing [164]; to determine the checkpointing interval with the goal to reduce the job runtime and improve reliability [63]. For checkpointing-based approaches, the efficiency of the approaches heavily depend on the characteristics of failures and they may significantly slowdown job execution. In this chapter, we use replication-based techniques, we plan to integrate our approach with checkpointing.

Replication-based techniques can be classified into application-level and system-level replication. For application-level replication techniques, the application developer should provide customized code to create replication of the applications. Researchers propose

to use iterative redundancy which trade-offs accurateness for cost-efficiency in volunteer computing environment [30]; to use different fault-tolerance techniques for different parts of applications [238]. Different from them [30, 238], we use system-level replication techniques to improve the availability of systems.

For system-level replication techniques, the system where applications run can create replicas for parts of the system. For example, Dynamo [56], a highly available key-value store, automatically replicates data items in multiple locations to ensure the availability of the data store service. Different from the data-replication techniques used in Dynamo, we use a VM-level replication technique that creates backups for VMs. Researchers propose to change the location of AA backups when failure happens with the goal to optimize availability and application performance (latency) [115]; to use AA and AS to achieve dependable VMs allocation [232]; to use AS to provide HA with the goal to minimize the number of hosts used [20]. These approaches [20, 115, 232] statically allocate VMs to hosts and only change the location of backup VMs when failure happens. In contrast, our method dynamically manage backup VMs by taking into account the time-varying availability requirements and failure predictions.

Some work use checkpointing and replication techniques both. Chtepen et al. [45] use the two techniques both to improve resource utilization when running bag-of-task in Grid. Elliott et al. [69] use the techniques both to provide fault-tolerance for MPI applications. Different from them, we manage the backups dynamically instead of statically.

8.5 Summary

Datacenters are hosting the ICT services that serve our daily life. Failures, which are bounded to happen in datacenters, can disrupt the availability of ICT services. Although many high availability (HA) techniques have already been developed to mask failures, dynamically selecting and configuring HA techniques for applications are still daunting for datacenter practitioners and researchers.

In this chapter, we propose, Availability on Demand (AoD) a mechanism consisting of an API that allows datacenter users to specify availability requirements which can change over time, and a scheduler which provides HA to applications based on user-specified requirements by dynamically managing computing resources for applications. We equip our HA mechanism with a scheduling policy AoD+R which responds to the dynamic availability requirements expressed by datacenter users with efficient management of resources. By evaluating our proposed approach through realistic, trace-based simulation, we show that the AoD+R policy can protect applications during important occasions, while the baseline policies cannot. Moreover, compared to an ideal policy which uses perfect predictions, the AoD+R policy consume 13% to 31% more resources but with similar availability for critical parts of applications.

Chapter 9

Conclusion

Networked Virtual Environments (NVEs) have hundreds of millions of users with a global market of tens of billions of Euros. In recent years, cloud computing has emerged as a new computing paradigm which can offer computing resources to NVEs on-demand. Due to the flexibility, availability, and pay-as-you-go features of clouds, there is an increasing number of NVEs that are hosted in clouds. Beside the traditional challenges inherited from non-cloud-based NVEs, for example, scalability and consistency, cloud-based NVEs introduce new challenges and opportunities such as cost-efficient resource management. In this thesis, we have developed a benchmarking system for NVEs, we have collected and analyzed the traces of the workloads of several NVEs, we have designed and implemented methods to massivize NVEs, and we have evaluated them with realistic simulations and real-world environments. In this chapter, we present the main contributions of this thesis in Section 9.1 and we suggest directions for future work in Section 9.2.

9.1 Main Contributions

Based on the research reported in this thesis, we list the main contributions of the thesis below.

1. We have introduced RTSenv, a benchmarking system with a focus on Real Time Strategy (RTS) games that is useful for both NVE researchers and developers. Our system can control and measure many RTS-specific aspects, and enables a variety of experimental scenarios, from performance evaluations to taking game design decisions. RTSenv can operate in several types of computing environments, from single desktop computers to wide-area multi-clusters and commercial clouds, and leverages reactive fault tolerance techniques to perform robust, multi-machine, multi-instance RTS game experiments.

2. We have proposed a method for studying the characteristics of Online Meta-Gaming Networks (OMGNs), and we have applied this method to observe and analyze XFire, which is a popular OMGN that serves about 20 million users playing over 1,500 games. We have found that OMGN players spend collectively in-game over 100 years hourly, that a significant fraction of the players have played over 10,000 in-game hours, that OMGN members are routinely engaged in the creation and consumption of game-related media such as screenshots and videos, and that OMGN members have on average over 60 friends.

3. We have shown how a general formalism can be used to extract social relationships from the interactions that occur between networked-game players. Based on five types of interactions, we have investigated implicit social structures of four popular Multiplayer Online Battle Arena (MOBA), RTS games, and Massively Multiplayer Online First Person Shooter (MMOFPS) games. Moveover, we have provided hints on improving gaming experience through two socially aware services.

4. We have collected detailed virtual world mobility traces from World of Warcraft. Using these traces, the public Second Life traces, and two real-world mobility traces collected by others and kindly shared with us, we have conducted a comprehensive, comparative study of the mobility characteristics of NVEs and real-world. Our study has shown evidence that long-tail distributions characterize well flight lengths, pause durations, and area popularity, that avatars move within limited sets of areas inside a virtual world city, and that avatars do have preferences to different areas. We have also indicated several differences between NVE and real-world mobility characteristics: the flight length distributions have longer tails for real-world, and the personal preference for different areas is more pronounced for real-world. Based on these empirical observations, we have proposed a mobility model, SAMOVAR, to generate realistic mobility traces of avatars in NVEs.

5. We have conducted the first empirical investigation of the presence of areas of interest in Multi-Avatar Virtual Environments (MAVEs). We have found that, unlike the other virtual environments such as RPG and FPS games, in MAVEs users have multiple areas of high interest and that interest location changes quickly. Based on this observation, we have proposed a novel scalable mechanism, Area of Simulation (AoS), which uses update-based and event-based operation for areas of interest and provides a scalability-consistency trade-off. We have proposed an AoS-based system architecture for scalable MAVEs, which supports the dynamic management of multiple areas of interest and several more common, scalability-related techniques. We have compared qualitatively and quantitatively our approach with various state-of-the-art approaches via simulations and real-world experiments, and have shown

strong evidence that AoS-based approaches offer superior performance and more flexibility for MAVEs.

6. We have proposed CoH, a Cloud-base, online, Hybrid scheduling policy which makes use of multiple machine configurations to plan enough capacity for NVE systems with less cost. We have formulated the resource provisioning and allocation problem as an Integer Programming Problem (IPP). To obtain the scheduling decision online, CoH limits the time of exploration for a solution and only obtains a suboptimal IPP solution. CoH makes scheduling decisions by picking the best among solutions of the IPP and various heuristics; thus, CoH operates as a portfolio scheduler. Further, we have proposed CoH-R, a policy that makes use of both on-demand and reserved instances to reduce the cost. Via simulation using real-world traces, we have shown that our approaches can lead to significantly lower cost than heuristics while operating online.

7. We have proposed Availability-on-demand (AoD), a mechanism consisting of an API that allows NVE operators to specify availability requirements which can change over time, and a scheduler which dynamically manages computing resources for NVEs. The mechanism operates at the level of individual service instance, thus enabling fine-grained control of availability, for example during sudden requirement changes and periodic operations. Through realistic, trace-based simulations, we have shown that the AoD mechanism meet our design goal to protect NVEs with low cost. The AoD approach provides for NVEs higher availability than the approaches which use high-availability techniques randomly, and consumes the same number of CPU hours. Moreover, comparing to an ideal approach which has perfect predictions about failures, the AoD approach achieves similar availability for critical parts of NVEs with 13% to 31% more CPU usages.

9.2 Suggestions for Future Work

There are a few future directions that are worth exploring:

1. We suggest building a better automatic benchmarking system by extending the RTSenv developed in this thesis. NVE researchers can extend RTSenv by allowing it to control and measure Role Playing Game (RPG) specific metrics. Moreover, we suggest integrating RTSenv with a cloud gaming framework to automatically test and compare the performance of various cloud gaming system designs.

2. We suggest further analyzing the XFire datasets. While the richness of the datasets allows NVE researchers to foresee many avenues for analysis, we suggest in par-

ticular correlating our findings with a similar, albeit much less detailed, study of Steam datasets that we have already collected.

3. The field of online social networks research applied to online games is rich and could lead to important improvements in gameplay, with direct impact to networked-resource consumption and quality of experience. We suggest expanding the methodology which is used to analyze implicit relationships of online social networks for online games by taking into account temporal aspects such as loose (dense) interactions between players over long (short) periods of time.

4. Movement is the most common action in NVEs. The mobility of avatars in NVEs exhibits such properties as hotspots, long-tail pause duration, and invisible personal visitation boundary. We suggesting developing a cloud-based automatic load-balancing method which takes into account the observed mobility characteristics and the billing model of clouds to support large-scale NVEs with low operational cost.

5. MAVEs such as RTS games are one of the most popular genres of online games. We suggest investigating automatic tuning and balancing of the Area of Simulation (AoS) systems, and integrating into the AoS architecture the ability to increase the capacity of MAVE systems on-demand.

6. Clouds offer multiple billing options: on-demand, reserved, spot, etc. For the on-demand billing model, users are charged with an hourly cost. For the reserved billing model, users need to pay an upfront cost but with lower hourly cost. For the spot billing model, users can bid to pay with lower hourly cost than the reserved billing model without any upfront cost. We suggest using the spot billing model to further reduce the operational cost of hosting NVEs.

7. Ensuring high availability is crucial to the proliferation of NVEs. We suggest extending the availability-on-demand mechanism with the Active-Standby technique which recovers a failed VM from a booted standby VM to improve the availability of NVE systems with low resource usage.

Bibliography

[1] 0 A.D. team. 0 A.D.— A free, open-source game of ancient warfare. http://wildfiregames.com/0ad/, 2014.

[2] A.-H. G. Abulrub, A. Attridge, and M. A. Williams. Virtual Reality in Engineering Education: The Future of Creative Learning. *Int. J. Engineering and Technology*, 6(4):4–11, 2011.

[3] D. Ahmed and S. Shirmohammadi. Zoning Issues and Area of Interest Management in Massively Multiplayer Online Games. In *Handbook of Multimedia for Digital Entertainment and Arts*, pages 175–195. 2009.

[4] D. Ahmed and S. Shirmohammadi. Improving online gaming experience using location awareness and interaction details. *Multimedia Tools Appl.*, 61(1), 2012.

[5] J. Ahn, C. Kim, J. Han, Y. Choi, and J. Huh. Dynamic virtual machine scheduling in clouds for architectural shared resources. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2012.

[6] Amazon Inc. AWS Case Study: Cmune. http://aws.amazon.com/solutions/case-studies/cmune/, 2014.

[7] C. Ang. Interaction networks and patterns of guild community in massively multiplayer online games. *Social Netw. Analys. Mining*, 1(4):341–353, 2011.

[8] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. E. Long. Managing Flash Crowds on the Internet. In *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2003.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, UCBekerly, 2009.

[10] ASCI. Distributed ASCI Supercomputer-Version 4, 2010.

[11] P.-L. Aublin, S. B. Mokhtar, and V. Quéma. RBFT: Redundant Byzantine Fault Tolerance. In *International Conference on Distributed Computing Systems (ICDCS)*, 2013.

[12] M. Baladi, H. Vitali, G. Fadel, J. Summers, and A. Duchowski. A taxonomy for the design and evaluation of networked virtual environments: its application to collaborative design. *Int. J. Interactive Design and Manufacturing*, 2(1):17–32, 2008.

[13] M. Balint, V. Posea, A. Dimitriu, and A. Iosup. An analysis of social gaming networks in online and face to face bridge communities. In *International Workshop on Large-scale System and Application Performance (LSAP)*, 2011.

[14] N. E. Baughman and B. N. Levine. Cheat-proof Playout for Centralized and Distributed Online Games. In *Conference on Computer Communications (INFOCOM)*, 2001.

[15] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a Needle in Haystack: Facebook's Photo Storage. In *USENIX Conference on Operating systems design and implementation (OSDI)*, 2010.

[16] O. A. Ben-Yehuda, A. Schuster, A. Sharov, M. Silberstein, and A. Iosup. ExPERT: Pareto-Efficient Task Replication on Grids and a Cloud. In *International Parallel & Distributed Processing Symposium (IPDPS)*, 2012.

[17] Y. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developer Conference*, 2001.

[18] Y. W. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, 2001.

[19] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, X. Zhuang, and A. Bharambe et al. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2008.

[20] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz. Guaranteeing High Availability Goals for Virtual Machine Placement. In *International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[21] J. Blackburn, R. Simha, C. Long, X. Zuo, N. Kourtellis, J. Skvoretz, and A. Iamnitchi. Cheaters in a gaming social network. *SIGMETRICS Performance Evaluation Review*, 39(3):101–103, 2011.

[22] Blizzard Inc. StarCraft II: Wings of Liberty Becomes Biggest PC Game Launch of the Year, 2010.

[23] P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *ACM symposium on Cloud computing (SoCC)*, 2010.

[24] W. Bohte and K. Maat. Deriving and validating trip purposes and travel modes for multi-day GPS-based travel surveys. *Transportation Research Part C: Emerging Technologies*, 17(3):285–297, 2009.

[25] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca. Network Analysis in the Social Sciences. *Science*, 323(5916):892–895, 2009.

[26] J. Botev, M. Esch, H. Schloss, I. Scholtes, and P. Sturm. HyperVerse: simulation and testbed reconciled. *Int. J. Adv. Media Commu.*, 4(2):167–181, 2010.

[27] M.-S. Bouguerra, A. Gainaru, L. A. Bautista-Gomez, F. Cappello, S. Matsuoka, and N. Maruyama. Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

[28] J.-S. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In *Workshop on Network and Systems Support for Games (NetGames)*, 2006.

[29] E. Brewer. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer*, 45(2):23–29, 2012.

[30] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic. Smart Redundancy for Distributed Computation. In *International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[31] M. Buro. RTS Games as Test-Bed for Real-Time AI Research. In *Joint Conference on Information Science*, 2003.

[32] M. Buro and D. Churchill. Real-Time Strategy Game Competitions. *AI Magazine*, 33(3):106–108, 2012.

[33] M. Burton. *Dungeons and Desktops: The History of Computer Role-playing Games*. A K Peters Ltd, 2008.

[34] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst.*, 25(6):599–616, 2009.

[35] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw., Pract. Exper.*, 41(1):23–50, 2011.

[36] F. Cappello, A. Geist, B. Gropp, L. V. Kalé, B. Kramer, and M. Snir. Toward exascale resilience. *Int. J. High Perf. Comp. Appl.*, pages 374–388, 2009.

[37] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. B. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Internet Measurement Conference (IMC)*, 2007.

[38] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. *Transactions on Services Computing*, 5(2):164–177, 2012.

[39] C. Chambers, W.-C. Feng, S. Sahu, D. Saha, and D. Brandt. Characterizing online games. *IEEE/ACM Trans. Netw.*, 18(3):899–910, 2010.

[40] K.-T. Chen, P. Huang, and C.-L. Lei. Game traffic analysis: An MMORPG perspective. *Computer Networks*, 50(16):3002–3023, 2006.

[41] L. Chen and H. Shen. Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters. In *Conference on Computer Communications (INFOCOM)*, 2014.

[42] T. C. L. Chen and C. Verbrugge. A protocol for distributed collision detection. In *Workshop on Network and System Support for Games (NetGames)*, 2010.

[43] Y. Chen, S. Alspaugh, and R. H. Katz. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *PVLDB*, 5(12):1802–1813, 2012.

[44] S. Choy, B. Wong, G. Simon, and C. Rosenberg. The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency. In *Workshop on Network and System Support for Games (NetGames)*, 2012.

[45] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. D. Turck, P. Demeester, and P. A. Vanrolleghem. Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids. *IEEE Trans. Parallel Distrib. Syst.*, 20(2):180–190, 2009.

[46] W. Cirne and E. Frachtenberg. Web-Scale Job Scheduling. In *JSSPP*, 2012.

[47] Cisco Inc. Data Center High Availability Clusters. http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/HA_Clusters/HA_Clusters/HAOver_1.html, 2014.

[48] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Symposium on Networked Systems Design & Implementation (NSDI)*, 2005.

[49] M. Claypool. The effect of latency on user performance in Real-Time Strategy games. *Computer Networks*, 49(1):52–70, 2005.

[50] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[51] Costa, F. Rodrigues, G. Travieso, and V. Boas. Characterization of complex networks: A survey of measurements. *Adv. Phys.*, 56(1), 2007.

[52] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An Efficient Synchronization Mechanism for Mirrored Game Architectures. *Multimedia Tools Appl.*, 23(1):7–30, 2004.

[53] B. Cully, G. Lefebvre, D. T. Meyer, M. Feeley, N. C. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[54] M. D. de Assuncao, A. di Costanzo, and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2009.

[55] P. de Melo, V. Almeida, and A. Loureiro. Can complex network metrics predict the behavior of NBA teams? In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2008.

[56] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[57] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good. The cost of doing science on the cloud: the Montage example. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2008.

[58] K. Deng, J. Song, K. Ren, and A. Iosup. Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[59] K. Deng, R. Verboon, and A. Iosup. A Periodic Portfolio Scheduler for Scientific Computing in the Data Center. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2013.

[60] Y. Deng and R. W. H. Lau. On Delay Adjustment for Dynamic Load Balancing in Distributed Virtual Environments. *IEEE Trans. Vis. Comput. Graph.*, 18(4):529–537, 2012.

[61] Y. Deng and R. W. H. Lau. Dynamic Load Balancing in Distributed Virtual Environments Using Heat Diffusion. *ACM Trans. on Multimedia Comp., Commun. Appl.*, 10(2), 2014.

[62] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan. Gang migration of virtual machines using cluster-wide deduplication. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2013.

[63] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello. Optimization of cloud task processing with checkpoint-restart mechanism. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[64] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the Internet. *IEEE Netw.*, 13(4):6–15, 1999.

[65] Y. Dong, W. Ye, Y. Jiang, I. Pratt, S. Ma, J. Li, and H. Guan. COLO: COarse-grained LOck-stepping virtual machines for non-stop service. In *ACM symposium on Cloud computing (SoCC)*, 2013.

[66] N. Ducheneaut, N. Yee, E. Nickell, and R. J. Moore. The life and death of online gaming communities. In *SIGCHI Conference on Human Factors in Computing Systems*, 2007.

[67] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. University Press, Cambridge, 2010.

[68] EDGE Team. Rise of the Ancients: the unstoppable march of the MOBA. http://www.edge-online.com/news/rise-of-the-ancients-the-unstoppable-march-of-the-moba/, 2013.

[69] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. B. Ferreira, and C. Engelmann. Combining Partial Redundancy and Checkpointing for HPC. In *International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[70] M. Esch, W. T. Ooi, and I. Scholtes. Evaluation of the HyperVerse Avatar Management Scheme Based on the Analysis of Second Life Traces. In *International Conference on Parallel and Distributed Systems (ICPADS)*, 2009.

[71] ExileServers Inc. Minecraft hosting in the Cloud. https://exileservers.com/minecraft-in-the-cloud/.

[72] FaceBook Inc. FaceBook Statistics. http://www.facebook.com/press/info.php?statistics.

[73] D. Feitelson. Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/.

[74] W.-c. Feng, F. Chang, W.-c. Feng, and J. Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Trans. Netw.*, 13(3):488–500, 2005.

[75] S. Ferretti. A synchronization protocol for supporting peer-to-peer multiplayer online games in overlay networks. In *International Conference on Distributed Event-based Systems (DEBS)*, 2008.

[76] T. Fields. *Distributed Game Development: Harnessing Global Talent to Create Winning Games*. Focal Press, 2010.

[77] A. Folling and M. Hofmann. Improving Scheduling Performance Using a Q-Learning-Based Leasing Policy for Clouds. In *Euro-Par*. 2012.

[78] E. Frachtenberg and D. G. Feitelson. Pitfalls in Parallel Job Scheduling Evaluation. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2005.

[79] D. Frey, J. Royan, R. Piegay, A. M. Kermarrec, E. Anceaume, and F. Le Fessant. Solipsis: A decentralized architecture for virtual environments. In *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2008.

[80] T. Fritsch, B. Voigt, and J. H. Schiller. Distribution of online hardcore player behavior: (how hardcore are you?). In *Workshop on Network and System Support for Games (NetGames)*, 2006.

[81] R. Garfield. Metagames. Horsemen of the Apocalypse: Essays on Roleplaying, http://www.wizards.com/Magic/magazine/Article.aspx?x=mtg/daily/feature/96, 2000.

[82] S. Genaud and J. Gossa. Cost-Wait Trade-Offs in Client-Side Resource Provisioning with Elastic Clouds. In *International Conference on Cloud Computing (CLOUD)*, 2011.

[83] J. S. Gilmore and H. A. Engelbrecht. A Survey of State Persistency in Peer-to-Peer Massively Multiplayer Online Games. *IEEE Trans. Parallel Distrib. Syst.*, 2012.

[84] M. Girvan and M. E. J. Newman. Community Structure in Social and Biological Networks. *Proc. Natl. Acad. Sci., USA*, 99:7821–7826, 2002.

[85] Glenn Fiedler. What every programmer needs to know about game networking. http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/, 2010.

[86] A. D. Gloria, F. Bellotti, and R. Berta. Serious games for education and training. *Int. J. Serious Games*, 1(1), 2014.

[87] M. C. González, C. a. Hidalgo, and A.-L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–82, 2008.

[88] Google Inc. Angry Birds Soars Online with Google App Engine. https://cloud.google.com/files/Rovio.pdf, 2012.

[89] C. Granberg. *Programming an RTS Game With Direct3d*. Charles River Media, 2006.

[90] M. Granovetter. The Strength of Weak Ties: A Network Theory Revisited. *Sociol. Th.*, 1:201, 1983.

[91] J. Gregory. *Game Engine Architecture*. A K Peters, Ltd., 2009.

[92] Y. Guo and A. Iosup. The Game Trace Archive. In *Workshop on Network and System Support for Games (NetGames)*.

[93] Y. Guo, S. Shen, O. Visser, and A. Iosup. An Analysis of Online Match-Based Games. In *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2012.

[94] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, and S. M. Balle. Hpc-aware vm placement in infrastructure clouds. In *IEEE International Conference on Cloud Engineering (IC2E)*, 2013.

[95] M. Hadji and D. Zeghlache. Minimum Cost Maximum Flow Algorithm for Dynamic Resource Allocation in Clouds. In *International Conference on Cloud Computing (CLOUD)*, 2012.

[96] X. Han, Q. Hao, B. H. Wang, and T. Zhou. Origin of the scaling law in human mobility: Hierarchy of traffic systems. *Physical Review E*, 83, 2011.

[97] G. Henson. From Planes to Pets to People: The Growth and Breadth of Simulation Games. http://www.alteredgamer.com/pc-gaming/58403-from-planes-to-pets-to-people-the-growth-and-breadth\-of-simulation-games/, 2012.

[98] Y.-J. Hong, J. Xue, and M. Thottethodi. Selective commitment and selective margin: Techniques to minimize cost in an IaaS cloud. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2012.

[99] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: a scalable peer-to-peer network for virtual environments. *IEEE Netw.*, 20(4):22–31, 2006.

[100] S.-Y. Hu and K.-T. Chen. VSO: Self-organizing Spatial Publish Subscribe. In *Self-Adaptive and Self-Organizing Systems (SASO)*, 2011.

[101] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. GamingAnywhere: an open cloud gaming system. In *ACM Multimedia Systems Conference (MMSys)*, 2013.

[102] H. Huang. *Logistics Operations Management Simulation Practice Guide (In Chinese)*. Higher Education Press of China, 2010.

[103] B. A. Huberman. An Economics Approach to Hard Computational Problems. *Science*, 275(5296):51–54, 1997.

[104] IEEE. IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulations Applications. Entity Information and Interaction. *IEEE Std 1278-1993*.

[105] U. E. Inc. Planetary Annihilation.

[106] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. In *IEEE/ACM International Conference on Grid Computing (GRID)*, 2007.

[107] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema. The Grid Workloads Archive. *Future Generation Comp. Syst.*, 24(7):672–686, 2008.

[108] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(6):931–945, 2011.

[109] A. Iosup, R. van de Bovenkamp, S. Shen, A. L. Jia, and F. A. Kuipers. An Analysis of Implicit Social Networks in Multiplayer Online Games. *IEEE Internet Computing*, 2014.

[110] A. Israel and D. Raz. Cost aware fault recovery in clouds. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013.

[111] L. Itzel, F. Heger, G. Schiele, and C. Becker. The quest for meaningful mobility in massively multi-user virtual environments. In *Workshop on Network and System Support for Games (NetGames)*, 2011.

[112] B. Javadi, D. Kondo, A. Iosup, and D. H. J. Epema. The Failure Trace Archive: Enabling the comparison of failure measurements and models of distributed systems. *J. Parallel Distrib. Comput.*, 73(8):1208–1223, 2013.

[113] Jenna Pitcher. EA cancels Titanfall release in South Africa due to bad online performance. http://www.polygon.com/2014/3/7/5480654/ea-cancels-titanfall-release-in-south-africa, 2014.

[114] C. Jo, E. Gustafsson, J. Son, and B. Egger. Efficient live migration of virtual machines using shared storage. In *ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE)*, 2013.

[115] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Performance and availability aware regeneration for cloud based multitier applications. In *International Conference on Dependable Systems and Networks (DSN)*, 2010.

[116] J. Juul. *A Casual Revolution: Reinventing Video Games and Their Players*. MIT Press, 2009.

[117] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2003.

[118] A. Keränen, T. Kärkkäinen, and J. Ott. Simulating Mobility and DTNs with the ONE. *J. Commun.*, 5(2):92–105, 2010.

[119] J. Kienzle, C. Verbrugge, B. Kemme, A. Denault, and M. Hawker. Mammoth: a massively multiplayer game research framework. In *International Conference on Foundations of Digital Games, (FDG)*, 2009.

[120] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. R. Wixon. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *Conference on Human Factors in Computing Systems (CHI)*, 2008.

[121] J. Kinicki and M. Claypool. Traffic analysis of avatars in Second Life. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 69–74, 2008.

[122] B. Kirman and S. Lawson. Hardcore Classification: Identifying Play Styles in Social Games Using Network Analysis. In *International Conference Entertainment Computing (ICEC)*, pages 246–251, 2009.

[123] J. M. Kleinberg. The convergence of social and technological networks. *Commun. ACM*, 51(11):66–72, 2008.

[124] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Conference on Computer Communications (INFOCOM)*, 2004.

[125] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[126] Korn, D. and MacDonald, J. and Mogul, J. and Vo, K. The VCDIFF Generic Differencing and Compression Data Format, 2002.

[127] L. Krammer, G. Schiele, D. Koch, and C. Becker. Quality of Experience-Aware Event Synchronization for Distributed Virtual Worlds. In *International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

[128] C. La and P. Michiardi. Characterizing user mobility in Second Life. In *Workshop on Online Social Networks*, 2008.

[129] D. Lake, M. Bowman, and H. Liu. Distributed scene graph to enable thousands of interacting users in a virtual environment. In *Workshop on Network and System Support for Games (NetGames)*, 2010.

[130] K. Lee, Y. Kim, S. Chong, I. Rhee, and Y. Yi. Delay-capacity tradeoffs for mobile networks with Lévy walks and Lévy flights. In *Conference on Computer Communications (INFOCOM)*, 2011.

[131] Y.-T. Lee and K.-T. Chen. Is Server Consolidation Beneficial to MMORPG? A Case Study of World of Warcraft. In *International Conference on Cloud Computing (CLOUD)*, 2010.

[132] M. Lehn, T. Triebel, C. Leng, A. Buchmann, and W. Effelsberg. Performance Evaluation of Peer-to-Peer Gaming Overlays. In *International Conference on Peer-to-Peer Computing (P2P)*, 2010.

[133] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *International Conference on World Wide Web (WWW)*, 2008.

[134] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *International Conference on World Wide Web (WWW)*, 2008.

[135] H. Liang, R. N. Silva, W. T. Ooi, and M. Motani. Avatar mobility in user-created networked virtual worlds: measurements, analysis, and implications. *Multimedia Tools Appl.*, 45(1-3):163–190, 2009.

[136] G. Linden. Make Data Useful, 2006.

[137] H. Liu, M. Bowman, and F. Chang. Survey of state melding in virtual worlds. *ACM Comput. Surv.*, 44(4):21:1–21:25, 2012.

[138] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan. Leveraging virtualization to optimize high-availability system configurations. *IBM Syst. J.*, 47(4):591–604, 2008.

[139] F. Lu, S. Parkin, and G. Morgan. Load Balancing for Massively Multiplayer Online Games. In *Workshop on Network and System Support for Games (NetGames)*, 2006.

[140] J. C. S. Lui and M. F. Chan. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):193–211, 2002.

[141] D. Lupei, B. Simion, D. Pinto, M. Misler, M. Burcea, W. Krick, and C. Amza. Transactional memory support for scalable and transparent parallelization of multiplayer games. In *European Conference on Computer Systems (EuroSys)*, 2010.

[142] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *IEEE/ACM International Conference on Grid Computing (GRID)*, 2010.

[143] P. Marshall, K. Keahey, and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[144] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-Lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Multimedia*, 6(1):47–57, 2004.

[145] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Trans. Multimedia*, 6(1):47–57, 2004.

[146] G. McAllister and G. R. White. Evaluating User Experience in Games. chapter Video Game, pages 107–128. Springer Verlag, London, UK, 2010.

[147] J. McGee. The pros and cons of collision detection. http://wow.joystiq.com/2011/07/10/breakfast-topic-the-pros-and-cons-of-collision-detection/, 2011.

[148] J. McGonigal. *Reality is Broken: Why Games Make us Better and How They Can Change the World*. Jonathan Cape, 2011.

[149] J. L. Miller and J. Crowcroft. Avatar movement in World of Warcraft battlegrounds. In *Workshop on Network and System Support for Games (NetGames)*, 2009.

[150] J. L. Miller and J. Crowcroft. The Near-Term Feasibility of P2P MMOGs. In *Workshop on Network and System Support for Games*, 2010.

[151] P. Miller. Professional Gamers: A Day in the Life. PCWorld online article. http://bit.ly/f72HjT, 2011.

[152] A. Mirelman, B. L. Patritti, P. Bonato, and J. E. Deutsch. Effects of virtual reality training on gait biomechanics of individuals post-stroke. *Gait & Posture*, 31(4):433–437, 2010.

[153] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Internet Measurement Conference (IMC)*, 2007.

[154] P. Morillo, S. Rueda, J. M. Orduña, and J. Duato. A Latency-Aware Partitioning Method for Distributed Virtual Environment Systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(9):1215–1226, 2007.

[155] P. Morillo, S. Rueda, J. M. Orduña, and J. Duato. Ensuring the performance and scalability of peer-to-peer distributed virtual environments. *Future Generation Comp. Syst.*, 26(7):905–915, 2010.

[156] C. C. Moul and J. V. C. Nye. Did the Soviets collude? A statistical analysis of championship chess 1940-1978. *Journal of Economic Behavior & Organization*, 70(1-2):10–21, 2009.

[157] J. Müller, J. H. Metzen, A. Ploss, M. Schellmann, and S. Gorlatch. Rokkatan: scaling an RTS game design to the massively multiplayer realm. In *International Conference on Advances in Computer Entertainment Technology, (ACE)*, 2005.

[158] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen. Dynamic Provisioning of Virtual Organization Clusters. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2009.

[159] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. H. J. Epema, and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2008.

[160] V. Nae, A. Iosup, and R. Prodan. Dynamic Resource Provisioning in Massively Multiplayer Online Games. *IEEE Trans. Parallel Distrib. Syst.*, (3):380–395, 2011.

[161] M. T. Najaran, S.-Y. Hu, and N. C. Hutchinson. SPEX: Scalable Spatial Publish/Subscribe for Distributed Virtual Worlds Without Borders. In *ACM Multimedia Systems Conference (MMSys)*, 2014.

[162] A. Nazir, S. Raza, and C.-N. Chuah. Unveiling Facebook: a measurement study of social network based applications. In *Internet Measurement Conference (IMC)*, 2008.

[163] M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Rev.*, 45(2):167–256, 2003.

[164] B. Nicolae and F. Cappello. BlobCR: efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[165] B. Nicolae, F. Cappello, and G. Antoniu. Optimizing multi-deployment on clouds by means of self-adaptive prefetching. In *Euro-Par*, 2011.

[166] NIST/SEMATECH. e-Handbook of Statistical Methods. Online Book, 2003.

[167] OpenTTD team. OpenTTD. http://www.openttd.org/.

[168] OpenTTD team. TJIP OpenTTD Challenge. www.tjip.com/tjip-challenge.html.

[169] A. Oprescu and T. Kielmann. Bag-of-Tasks Scheduling under Budget Constraints. In *International Conference Cloud Computing Technology and Science (CloudCom)*, 2010.

[170] S. Ostermann and R. Prodan. Impact of variable priced cloud resources on scientific workflow scheduling. In *Euro-Par*, 2012.

[171] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller. The human-computer interaction handbook. chapter User-cente, pages 883–906. 2003.

[172] M. T. Paul Bettner. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. In *Game Developer Conference*, 2001.

[173] A. Petlund, P. l. Halvorsen, P. l. F. Hansen, T. Lindgren, R. Casais, and C. Griwodz. Network traffic from Anarchy Online: analysis, statistics and applications: a server-side traffic trace. In *ACM Multimedia Systems Conference (MMSys)*, 2012.

[174] D. Pittman and C. GauthierDickey. Characterizing virtual populations in massively multiplayer online role-playing games. *Advances in Multimedia Modeling*, 5916:87–97, 2010.

[175] V. Posea, M. Balint, A. Dimitriu, and A. Iosup. An analysis of social gaming networks in online and face to face bridge communities. In *RoEduNet*, LSAP11, pages 218–223, 2010.

[176] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In *International Conference on Peer-to-Peer Systems (IPTPS)*, 2005.

[177] R. Becker et al. Human mobility characterization from cellular network data. *Commun. ACM*, 56(1), 2013.

[178] S. Ren, Y. He, and F. Xu. Provably-Efficient Job Scheduling for Energy and Fairness in Geographically Distributed Data Centers. In *International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[179] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong. On the Levy-Walk Nature of Human Mobility. 2008.

[180] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle. Peer-to-Peer-Based Infrastructure Support for Massively Multiplayer Online Games. In *Consumer Communications and Networking Conference (CCNC)*, 2007.

[181] ROC team. The UC Berkeley/Stanford Recovery-Oriented Computing (ROC) Project. http://roc.cs.berkeley.edu/.

[182] A. Rollings and E. Adams. *Fundamentals of Game Design*. Prentice Hall, 2006.

[183] P. Rosedale and C. Ondrejka. Enabling Player-Created Online Worlds with Grid Computing and Streaming. Gamasutra Resource Guide, 2003.

[184] P. R. Rosenbaum. *Observational Studies*. Springer Verlag, 2nd. edition, 2002.

[185] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, 2010.

[186] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. In *International Symposium on Cluster Computing and the Grid*, 2007.

[187] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. P. Buchmann. pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games. In *International Conference on Peer-to-Peer Computing (P2P)*, 2008.

[188] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: a large-scale field study. In *International joint conference on Measurement and modeling of computer systems (SIGMETRICS)*, 2009.

[189] Schwiegelshohn, U., Badia, R. M., Bubak, M., et al. Perspectives on Grid Computing. *Future Generation Comp. Syst.*, 26(8):1104–1115, 2010.

[190] J. Scott. Social network analysis: developments, advances, and prospects. *Social Netw. Analys. Mining*, 1(1):21–26, 2011.

[191] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha. On demand platform for online games. *IBM Sys. J.*, 45:7–20, 2006.

[192] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A Cost-Aware Elasticity Provisioning System for the Cloud. In *International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[193] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in Warcraft III. In *Workshop on Network and System Support for Games*, pages 3–14, 2003.

[194] S. Shen, V. Beek, and A. Iosup. Towards Characterizing Business-Critical Workloads Hosted in Cloud Datacenters. In *CCGrid*, 2015, to appear.

[195] S. Shen, N. Brouwers, A. Iosup, and D. Epema. Characterization of Human Mobility of Networked Virtual Environments. In *NOSSDAV*, 2014.

[196] S. Shen, K. Deng, A. Iosup, and D. H. J. Epema. Scheduling Jobs in the Cloud Using On-Demand and Reserved Instances. In *Euro-Par*, 2013.

[197] S. Shen, S.-Y. Hu, A. Iosup, and D. Epema. Area of Simulation: Mechanism and Architecture for Multi-Avatar Virtual Environments. *Transactions on Multimedia Computing, Communications and Applications (TOMCCAP), under minor revision*.

[198] S. Shen and A. Iosup. Modeling Avatar Mobility of Networked Virtual Environments. In *MMVE*, 2014.

[199] S. Shen, A. Iosup, A. Israel, W. Cirne, D. Raz, and D. H. J. Epema. An Availability-on-Demand Mechanism for Datacenters. In *CCGrid*, 2015, to appear.

[200] S. Shen, O. Visser, and A. Iosup. RTSenv: An experimental environment for real-time strategy games. In *Workshop on Network and System Support for Games (NetGames)*, 2011.

[201] S. Shen, O. Visser, and A. Iosup. RTSenv: An Experimental Environment for Real-Time Strategy Games on Multi-Clusters. Tech.Rep. PDS-2011-002, TU Delft, 2011.

[202] K. J. Shim, K.-W. Hsu, and J. Srivastava. Modeling Player Performance in Massively Multiplayer Online Role-Playing Games: The Effects of Diversity in Mentoring Network. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2011.

[203] C. Song, T. Koren, P. Wang, A.-L. Barabási, and C. Song et al. Modelling the scaling properties of human mobility. *Nature Phys.*, 6(10):818823, Sept. 2010.

[204] Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In *Conference on Computer Communications (INFOCOM)*, 2012.

[205] Spil Games. 2013 State of Online Gaming Report. http://www.spilgames.com/press/\2013-state-online-gaming-report-released-spil-games/, 2013.

[206] M. Stillwell, F. Vivien, and H. Casanova. Dynamic fractional resource scheduling for HPC workloads. In *International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010.

[207] S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, 2001.

[208] D. Stutzbach, R. Rejaie, N. G. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Trans. Netw.*, 17(2):377–390, 2009.

[209] M. Suznjevic and M. Matijasevic. Player behavior and traffic characterization for MMORPGs: a survey. *Multimedia Syst.*, 19(3):199–220, 2012.

[210] M. Suznjevic, I. Stupar, and M. Matijasevic. A model and software architecture for MMORPG traffic generation based on player behavior. *Multimedia Systems*, 19(3):231–253, 2012.

[211] M. Szell and S. Thurner. Measuring social dynamics in a massive multiplayer online game. *Social Networks*, 32(4):313–329, 2010.

[212] H. Tabuchi. Sony Says Parts of PlayStation Network Will Be Back Online This Week. NYTimes article. http://www.nytimes.com/2011/05/02/technology/02sony.html?_r=0, 2011.

[213] D. Talia. Clouds for Scalable Big Data Analytics. *IEEE Computer*, 46(5), 2013.

[214] S. A. Tan, W. Lau, and A. Loh. Networked game mobility model for first-person-shooter games. In *Workshop on Network and System Support for Games (NetGames)*, 2005.

[215] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware, utility-based job scheduling on Blue, Gene/P systems. In *IEEE International Conference on Cluster Computing (CLUSTER)*, 2009.

[216] X. Tang and S. Zhou. Update Scheduling for Improving Consistency in Distributed Virtual Environments. *IEEE Trans. Parallel Distrib. Syst.*, 2010.

[217] The Entertainment Software Association. Essential Facts About the Computer and Video Game Industry: Sales, Demographics, and Usage Data, 2012.

[218] S. Tolic and H. Hlavacs. A testbed for p2p gaming using time warp. In *Transactions on Edutainment II*, volume 5660, pages 33–47. 2009.

[219] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Gener. Comput. Syst.*, 28(2), 2012.

[220] T. Triebel, M. Lehn, R. Rehner, B. Guthier, S. Kopf, and W. Effelsberg. Generation of synthetic workloads for multiplayer online gaming benchmarks. In *Workshop on Network and System Support for Games (NetGames)*, 2012.

[221] M. Varvello, S. Ferrari, E. Biersack, and C. Diot. Exploring Second Life. *IEEE/ACM Trans. Netw.*, 19(1):80–91, 2011.

[222] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup. An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[223] VMWare Inc. Protecting Mission-Critical Workloads with VMware Fault Tolerance. www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf.

[224] J. Waldo. Scaling in games & virtual worlds. *ACM Queue*, 51(8), 2008.

[225] W. E. Walker, J. Giddings, and S. Armstrong. Training and learning for crisis management using a virtual simulation/gaming environment. *Cognition, Technology & Work*, 13(3):163–173, 2011.

[226] D. Warneke and O. Kao. Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. *IEEE Trans. Parallel Distrib. Syst.*, 22(6):985–997, 2011.

[227] J. Webb. How the cloud helps Netflix. [Online] Available: http://radar.oreilly.com/2011/05/netflix-cloud.html, 2011.

[228] S. D. Webb, W. Lau, and S. Soh. NGS: an application layer network game simulator. In *Australasian conference on Interactive entertainment*, 2006.

[229] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *European Conference on Computer Systems (EuroSys)*, 2009.

[230] N. Wingfield. In E-Sports, Video Gamers Draw Real Crowds and Big Money. http://www.nytimes.com/2014/08/31/technology/esports-explosion-brings-opportunity-riches-for-video-gamers.html?_r=0, 2014.

[231] A. Yahyavi and B. Kemme. Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey. *ACM Comput. Surv.*, 46(1):9:1–9:51, 2013.

[232] H. Yanagisawa, T. Osogami, and R. Raymond. Dependable virtual machine allocation. In *Conference on Computer Communications (INFOCOM)*, 2013.

[233] A. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 99–104, 2005.

[234] L. Zhang and X. Tang. The Client Assignment Problem for Continuous Distributed Interactive Applications. In *International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[235] L. Zhang, J. W. Wade, D. Bian, A. Swanson, Z. Warren, and N. Sarkar. Data Fusion for Difficulty Adjustment in an Adaptive Virtual Reality Game System for Autism Intervention. In *HCI International*, 2014.

[236] T. Zhang, Z. Du, Y. Chen, X. Ji, and X. Wang. Typical Virtual Appliances: An optimized mechanism for virtual appliances provisioning and management. *Journal of Syst. and Soft.*, 84(3):377–387, 2011.

[237] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. Performance Implications of Failures in Large-Scale Cluster Scheduling. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 233–252, 2004.

[238] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King. FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications. In *International Symposium on Software Reliability Engineering (ISSRE)*, 2010.

Appendix A

Datasets

Understanding the workloads of NVEs is important. By analyzing workloads, the system designers can gain insights about the systems' and users' behaviors, and develop better systems to support the users. We have in total collected 14 datasets, and analyzed some of the datasets in Chapters 3, 5, and 6. The overview of the datasets are listed in Table A.1. The "GTA ID" column indicates the Game Trace Archive (GTA) trace ID of the dataset if it is published in GTA [92].

Table A.1: Datasets Overview.

No	Name	GTA ID	Analyzed in	Duration	Scale
	Multiplayer Online Battle Arena datasets				
1	DotAlicious matches	GTA-T1	Chapter 3 and [93, 109]	2010.4 - 2012.2	617,069 matches
2	DotAlicious replays			2012.2 - 2012.2	about 13,000 replays
3	Dota-League matches	GTA-T2	Chapter 3 and [93, 109]	2008.11- 2011.7	1,470,786 matches
4	Dota-League replays			2006.7 - 2007.1	about 80,000 replays
5	League of Legend			2011.11 - 2011.12	about 120,000 matches
	World of Tanks datasets				
6	WoT matches	GTA-T11	Chapter 3 and [109]	2010.8 - 2013.7	74,658 matches
7	WoT RU matches			2013.1 - 2013.8	about 0.6 millions matches
8	WoT clans			2011.3 - 2013.8	28365 clans
	StarCraft datasets				
9	SC2 replays		Chapter 6 and [197]	2010.7 - 2010.11	5,796 replays
10	SC2 matches	GTA-T10	Chapter 3 and [109]	2012.3 - 2013.8	85,532 matches
11	SC2 matches GGTracker			2012.10 - 2014.2	about 4.6 millions matches
13	**WoW mobility**	GTA-T12	Chapter 5 and [195]	2011.4 - 2011.9	4 virtual world cities
14	**VM stat**		[199]	2013.8-2013.9	1,750 VMs

Multiplayer Online Battle Arena datasets

Multiplayer Online Battle Arena (MOBA) is a game genre, in which each player controls an in-game representation (here, the hero or the champion), and equally-sized teams, generally of 3 or 5 players, have as objective the conquest of the opposite side's main building. This genre of games include many strategic elements, from the team operation to the management of resources and the creation of helper troops. In total, we have

collected traces from two games: Defense of the Ancients and League of Legend.

Defense of the Ancients (DotA) is a 5-against-5-player MOBA game. DotA is one of the most popular e-Sports, with many international DotA competitions and large audiences [230]. In DotA, social relationships, such as same-guild membership and even friendship, can improve the gameplay experience.

We have collected two matches datasets: *Dota-League matches* and *DotAlicious matches*, over multiple years for two DotA communities, Dota-League[1] and DotAlicious[2], respectively. Both communities identify each of their matches with a unique number in increasing order, and for each match dedicated information (such as the names of the players participating in the match and the duration of the match) is available on a corresponding webpage. We have crawled all the unique matches played within these communities via their openly accessible websites, by gradually increasing the identifier number from 1 to the total number of matches played; we obtained the latter from the main page of each website. Some of the webpages with a match identifier in the crawling range appeared to be broken. We have crawled each web page at least twice, at different times, to reduce the effect of possible temporary unavailability and traffic shaping of the website. The two replay datasets: *DotaLicious replays* and *Dota-League replays* are collected from Dota-League and DotAlicious, respectively. The replay dataset contains individual replay which record the players' actions during gameplay.

League of Legend (LoL) is one of the most popular MOBA game developed and published by `Riot Games`. The game play of LoL is similar to DotA. The *League of Legend* dataset contains about 120,000 matches from a fan organized website `www.leaguereplays.com`. This dataset is not analyzed yet.

World of Tanks datasets

World of Tanks (WoT)[3] is a free and popular Massively Multiplayer Online First Person Shooter (MMOFPS) Game developed by WarGaming[4]. There are a few fan organized websites that allow uses to upload their game matches results and share with each others. We collected two matches datasets: *WoT matches*and *WoT RU matches*. The data collection method is the same as the DotA datasets. The *WoT matches dataset* is collected from an European website `wotreplays.com`; this dataset is also analyzed in [109]. The *WoT RU matches* dataset is collected from a Russian website `wotreplays.ru`. Beside the matches datasets, we have collected *WoT clan dataset* which contains team information from about 30,000 European clans registered in the WarGaming's website.

StarCraft II datasets

[1] `www.dota-league.com`
[2] `www.dotalicious.com`
[3] `www.worldoftanks.com/`
[4] `Wargaming.net`

StarCraft II[5], the most popular Real Time Strategy games, is played by millions of players. We have collected one replay dataset *SC2 replays* and two matches datasets: *SC2 matches* and *SC2 matches GGTracker*. The data collection method is the same as the MOBA datasets. The *SC2 replays* dataset, collected from a fan organized website drop.sc, contains the actions of about 1,000 players issued during game play. The *SC2 matches* dataset is collected from drop.sc; the dataset is analyzed in [109]. Similarly, we have collected the *SC2 matches GGTracker* dataset from a fan organized website ggtracker.com/. We do not analyze this dataset yet.

WoW mobility dataset

World of WarCraft (WoW)[6] is one of the most popular Massively Multiplayer Online Role Playing Game. The virtual world of WoW resembles a medieval, albeit fantasy-based, real-world environment. The players of WoW need to be highly mobile, to be able to finish quests of the storyline, trade goods, and socialize with the other players. We have collected and analyzed the position information of about 30,000 players.

VM stat dataset

We have collected from a distributed datacenter hosting business-critical workloads two large-scale and long-term workload traces from about 1,800 virtual machines owned by an IT company Bitbrains[7]. We have analyzed in [194] using these traces both requested resources and actual resource usage, in terms of CPU, memory, and disk and network I/O.

[5] www.starcraft.com/
[6] www.warcraft.com/
[7] www.bitbrains.nl/

Summary

Networked Virtual Environments (NVEs) are virtual environments where physically distributed, Internet-connected users can interact and socialize with others. The most popular NVEs are online games, which have hundreds of millions of users and a global market of tens of billions Euros per year. Besides entertainments, NVE techniques are used in education, enterprise training, disaster-scenario analysis, etc. Because the number of NVE users is ever increasing, new NVEs which can host massive number of users are increasingly needed. To meet the demands of NVE users, NVE researchers and designers are continuously seeking novel ways to design NVEs. In recent years, due to the scalability, the flexibility, and the cost-efficiency of cloud computing technologies, clouds are gaining popularity as computing platforms for NVEs. It is challenging to manage large amounts of cloud computing resources to serve NVE users in a scalable, consistent, highly available, cost efficient, and interactive way. To massivize NVEs on clouds, in this thesis, we analyse the workloads of several NVEs, design and implement several NVE mechanisms, and evaluate them using realistic simulations or real-world experiments. Although we validate our approaches using exclusively online gaming data, we believe that the fundamental findings of this thesis can be applied to other fields of NVEs, because online games are often found to be among the most demanding kinds of NVEs.

In Chapter 2, we introduce RTSenv, a benchmarking system for NVEs with a focus on Real Time Strategy (RTS) games. RTSenv can operate in several types of computing environments, from desktop-computers, to wide-area multi-clusters and commercial clouds. It leverages reactive fault tolerance techniques to perform robust, multi-machine, multi-instance RTS game experiments. RTSenv can help NVE researchers and practitioners to gain better insight into the real-world performance of NVEs.

In Chapter 3, we propose a formalism to understand the implicit social networks of NVEs. The formalism consists of various ways to map interaction to social structure. By applying the formalism to real-world data collected from three different game genres, we analyse the implications of the formalism for in-game and gaming-related services, ranging from network and socially-aware matchmaking of players, to an investigation of social network robustness against player departure.

In Chapter 4, we collect a long-term trace from an online meta-gaming networks:

XFire. We present a high-level, marginal distribution- and time-based analysis of XFire: its global network, player activity, user-generated content, and social structure. We find that XFire is a slowly growing network whose players spend collectively in-game over 100 years, every hour. We quantify the "hardcore"-ness of XFire players, and find that a significant fraction of them have played over 10,000 in-game hours.

In Chapter 5, we collect mobility traces of virtual-world citizens from World of Warcraft (WoW), and compare these traces with mobility traces collected from Second Life. Furthermore, motivated by the existence of numerous studies and models of mobility for networked real-world environments (NRE), we systematically study the characteristics of two NVE and two NRE mobility traces. We find that the mobility of citizens in real-world and virtual-world share many properties but with some differences. Based on the findings of this study, we propose a mobility model which can be used to generate realistic mobility traces for virtual-world citizens.

In Chapter 6, we propose Area of Simulation (AoS), a scalability mechanism for multi-avatar virtual environments such as RTS games. The AoS mechanism combines and extends the mechanisms of Area of Interest (AoI) and Event-Based Lockstep Simulation (EBLS). Novel in the AoS mechanism, it uses both event-based and update-based operational models to manage not single, but multiple areas of interest. Moreover, the AoS mechanism synchronizes only selected areas of the virtual world instead of all the virtual world. We further design an AoS-based architecture, which uses the AoS and several scalability mechanisms simultaneously, dynamically trading-off consistency guarantees for scalability. We implement and deploy this architecture and we demonstrate that it can operate with an order of magnitude more avatars and a larger virtual world than common alternatives, yet without exceeding the resource capacity of computers of players.

In Chapter 7, we propose CoH, a Cloud-based, online, Hybrid scheduling policy which reduces operational cost of hosting NVEs by making use of both on-demand and reserved instances of clouds. When provisioning and allocating computing resources for NVEs, the CoH policy dynamically selects the best solution among the solutions of some heuristics, and then manages resources accordingly. We show, via simulation and using multiple real-world traces, that the hybrid scheduling policy can obtain significantly lower cost than typical heuristics-based policies.

In Chapter 8, we propose Availability-on-Demand (AoD), a mechanism providing high availability (HA) for NVE services when and only when HA is needed. The mechanism consists of an API that allows NVE operators to specify availability requirements which can dynamically change, and an availability-aware scheduler that dynamically manages computing resources based on user-specified requirements. Through trace-based simulation, we show that the AoD mechanism can protect important parts of NVEs dynamically.which also leads to low operational cost.

Samenvatting

Networked Virtual Environments (NVEs) zijn virtuele omgevingen waarin fysiek gedistribueerde gebruikers kunnen communiceren en socialiseren via internet. De meest populaire NVEs zijn online games, die honderden miljoenen gebruikers en een wereldmarkt van tientallen miljarden euro's per jaar hebben. Naast entertainmentdoeleinden worden NVE-technieken ook gebruikt in onderwijs, bedrijfstrainingen, rampenscenarioanalyse enz. Als het aantal NVE-gebruikers steeds toeneemt, zijn nieuwe soorten NVEs noodzakelijk om het enorme aantal gebruikers te kunnen bedienen. NVE onderzoekers en ontwerpers zoeken voortdurend naar nieuwe methoden voor het ontwerpen van NVEs, zodat die aan de eisen van NVE-gebruikers voldoen. In de afgelopen jaren is cloudinfrastructuur een steeds populairder wordende infrastructuur voor NVEs door de verbeteringen in de schaalbaarheid, flexibiliteit en kostenefficiëntie van cloud computing technologieën. Het is uitdagend om grote hoeveelheden cloud computing resources te beheren zodat NVE-gebruikers kunnen worden bediend op een schaalbare, consistente, hoogbeschikbare, kostenefficiënte en interactieve wijze. Om NVEs te schalen op clouds hebben wij in dit proefschrift verschillende NVE-workloads geanalyseerd, diverse NVE-mechanismen ontworpen, geïmplementeerd en geëvalueerd met behulp van realistische simulaties van real-world experimenten. Hoewel wij onze aanpak uitsluitend valideren met online gamingsgegevens, zijn wij van mening dat de fundamentele bevindingen in dit proefschrift op andere gebieden dan NVEs kunnen worden toegepast, omdat online games vaak beschouwd worden als een van de meest veeleisende NVEs.

In Hoofdstuk 2 introduceren wij RTSenv, een benchmarkingssysteem voor NVEs die zich vooral richt op Real Time Strategy (RTS) games. RTSenv kan opereren in verschillende soorten ICT-omgevingen; onder andere desktopcomputers, wide-area multiclusters en commerciële cloud infrastructuren. RTSenv maakt gebruikt van reactieve fouten-tolerante technieken die zorgen voor het uitvoeren van robuuste, multi-machine, multi-instance RTS-game experimenten. RTSenv kan NVE-onderzoekers en beoefenaars helpen om beter inzicht te krijgen in de prestaties van NVEs in de werkelijkheid.

In Hoofdstuk 3 stellen we een formalisme voor om de impliciete sociale netwerken van NVEs te begrijpen. Het formalisme bestaat uit verschillende manieren om interactie in verband te brengen met sociale structuren. Door het formalisme toe te passen

op real-world gegevens verzameld uit drie verschillende game genres, analyseren wij de uit het formalisme komende verbanden op in-game en gaminggerelateerde diensten, van netwerk- en sociale relaties van de spelers tot de robuustheid van het sociale netwerk tegen het vertrek van spelers.

In Hoofdstuk 4 hebben wij een langetermijntrace van een online meta-gaming netwerk, XFire, verzameld. We presenteren een op tijd gebaseerde analyse van XFire inclusief zijn wereldnetwerk, spelersactiviteit, user-generated content, en sociale structuur. Onze bevinding is dat XFire is een langzaam groeiend netwerk is waarin de spelers gezamenlijk 100 jaar besteden in elk uur. Wij kwantificeren de "*hardcore*"-heid van XFire-spelers, en vinden dat een aanzienlijk deel van hen meer dan 10000 speeluren hebben.

In Hoofdstuk 5 verzamelen wij de mobiliteitstraces van burgers in de virtuele wereld van World of Warcraft (WoW), en vergelijken die met die van Second Life. Bovendien, gemotiveerd door talrijke mobiliteits-studies en modellen van Networked Real-World Environments (NREs), bestuderen we systematisch de kenmerken van twee NVE mobiliteitstraces en twee NRE mobiliteitstraces. Wij concluderen dat de mobiliteit van burgers in de echte wereld en in de virtuele wereld zeer vergelijkbaar zijn, maar dat er toch verschillen bestaan. Op basis van de bevindingen van dit onderzoek stellen we een mobiliteitsmodel voor dat gebruikt kan worden om realistische mobiliteitstraces van burgers in virtuele wereld te genereren.

In Hoofdstuk 6 stellen we Area of Simulation (AOS) voor, een schaalbaarheidsmechanisme voor multi-avatar virtuele omgevingen zoals RTS games. Het AoS-mechanisme combineert en breidt de mechanismen van Area of Interest (AOI) en Event-Based Lockstep Simulation (EBLS) uit. Het AOS-mechanisme beheert niet alleen één, maar meerdere AOIs door het gebruik van event-based en update-based modellen. Verder synchroniseert het AOS mechanisme alleen geselecteerde delen van de wereld in plaats van de gehele virtuele wereld. Boodndien ontwerpen wij een AOS-gebaseerde architectuur, die AOS en diverse schaalbaarheidsmechanismen tegelijk gebruikt en consistentiegaranties dynamisch afweegt tegen schaalbaarheid. Wij implementeren deze architectuur en tonen aan dat die functioneert met een orde van grootte meer avatars en een grotere virtuele wereld dan de alternatieven, zonder de computercapaciteit van de spelers te overschrijden.

In Hoofdstuk 7 stellen wij voor een cloud-based, online, hybride (CoH) schedulingsalgorithme dat de operationele kosten van de NVEs vermindert door het gebruik van zowel on-demand als gereserveerde instances van clouds. Het CoH algoritme selecteert dynamisch de beste oplossing uit een aantal heuristieken tijdens de verstrekking en de allocatie van IT-middelen voor NVEs. Via simulatie en verschillende real-world traces tonen we aan dat het hybride schedulingsalgoritme aanzienlijk lagere kosten heeft dan de gewone heuristieken.

In Hoofdstuk 8 stellen we Availibility-on-Demand (AoD) voor, een mechanisme

dat High Availibility (HA) verstrekt voor NVE-diensten wanneer HA noodzakelijk is. Het mechanisme bestaat uit een API waarmee NVE-operators dynamische beschikbaarheidseisen kunnen specificeren, en een beschikbaarheid-bewuste scheduler die dynamisch computercapaciteit beheert op grond van door gebruikers het gespecificeerde eisen. Via simulatie met traces laten we zien dat het AoD-mechanisme op een dynamische wijze belangrijke delen van NVEs kan beschermen, hetgeen ook tot lage operationele kosten leidt.

Biography

Siqi Shen was born in Zhaoan, China, September 27th, 1985. Siqi obtained his Bachelor and Master degree of computer science from National University of Defense Technology, China in 2007 and 2009, respectively. In 2010, he was a research assistant at Parallel and Distributed Processing Laboratory of National University of Defense Technology. Since October 2010, he is a PhD student in the Parallel and Distributed Systems group at Delft University of Technology, the Netherlands. He is interested in the analysis, the design, and the implementation of distributed systems, especially on Networked Virtual Environments and clouds.

Journal articles

1. Alexandru Iosup, Ruud van de Bovenkamp, **Siqi Shen**, Adele Lu Jia, and Fernando Kuipers, "An Analysis of Implicit Social Networks in Multiplayer Online Games," *IEEE Internet computing* 18(3), pp. 36-44, 2014.

2. **Siqi Shen**, Shun-Yun Hu, Alexandru Iosup, and Dick Epema, "Area of Simulation: Mechanism and Architecture for Multi-Avatar Virtual Environments," *ACM Transactions on Multimedia Computing, Communications and Application*, under minor revision.

3. Adele Lu Jia, **Siqi Shen**, Ruud van de Bovenkamp, Alexandru Iosup, Fernando Kuipers, and Dick Epema, "Socializing by Gaming: Revealing Social Relationships in Multiplayer Online Games," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, accepted for publication.

Conference articles

4. **Siqi Shen**, Alexandru Iosup, Assaf Israel, Danny Raz, Walfredo Cirne, and Dick Epema, "An Availability-on-Demand Mechanism for Datacenters," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015, to appear.

5. **Siqi Shen**, Vincent van Beek, and Alexandru Iosup, "Towards Characterizing Business-Critical Workloads Hosted in Cloud Datacenters," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015, to appear.

6. **Siqi Shen**, Niels Brouwers, Alexandru Iosup, and Dick Epema, "Characterization of Human Mobility in Networked Virtual Environments," *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2014.

7. **Siqi Shen** and Alexandru Iosup, "Modeling Avatar Mobility of Networked Virtual Environments," *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2014.

8. Yunhua Deng, **Siqi Shen**, Zhe Huang, Alexandru Iosup, and Rynson Lau, "Dynamic Resource Management in Cloud-based Distributed Virtual Environments," *ACM Multimedia*, 2014.

9. Alexandru Iosup, **Siqi Shen**, Yong Guo, Stefan Hugtenburg, Jesse Donkervliet, and Radu Prodan, "Massivizing Online Games using Cloud Computing: a Vision," *Cloud Gaming Systems and Networks, held in conjunction with IEEE International Conference on Multimedia & Expo (ICME)*, 2014.

10. **Siqi Shen**, Kefeng Deng, Alexandru Iosup, and Dick Epema, "Scheduling Jobs in the Cloud Using On-demand and Reserved Instances," *Euro-Par*, 2013.

11. **Siqi Shen**, Alexandru Iosup, and Dick Epema, "Massivizing Multi-Player Online Games on Clouds," *Doctoral Symposium at IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013.

12. Ruud van de Bovenkamp, **Siqi Shen**, Alexandru Iosup, and Fernando Kuipers, "Understanding and Recommending Play Relationships in Online Social Gaming," *International Conference on COMmunication Systems and NETworkS (COMSNETS)*, 2013.

13. Yong Guo, **Siqi Shen**, Otto Visser, and Alexandru Iosup, "An Analysis of Online Match-Based Games," *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2012.

14. **Siqi Shen**, Otto Visser, and Alexandru Iosup, "RTSenv: An Experimental Environment for Real-Time Strategy Games," *Annual Workshop on Network and Systems Support for Games (NetGames)*, 2011.

15. **Siqi Shen** and Alexandru Iosup, "The XFire Online Meta-Gaming Network: Observation and High-Level Analysis," *International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2011.

Technical reports

1. **Siqi Shen**, Vincent van Beek, Alexandru Iosup. *Workload characterization of cloud datacenter of Bitbrains*, PDS Technical Report PDS-2014-001.

2. **Siqi Shen**, Niels Brouwers, and Alexandru Iosup, *Human Mobility in Virtual and Real Worlds: Characterization, Modeling, and Implications*, PDS Technical Report PDS-2011-007.

3. **Siqi Shen**, Otto Visser, and Alexandru Iosup, *RTSenv: An Experimental Environment for Real-Time Strategy Games on Multi-Clusters*, PDS Technical Report PDS-2011-002.