中文图书分类号：TP39
密　　级：公开
UDC：　004
学　校　代　码：10005

工程硕士学位论文

M.E. DISSERTATION

论　文　题　目：实时网络游戏中关键技术的研究

论　文　作　者：云淼
领　　域：计算机应用
指　导　教　师：肖创柏

论文提交日期：2015.1.9

UDC：004 学校代码：10005
中文图书分类号：TP39 学　　号：G201007007
密　　级：公开

北京工业大学硕士专业学位论文

（非全日制）

题　　目：实时网络游戏中关键技术的研究

英文题目：RESEACH ON KEY TECHNOLOGIES OF REAL-TIME NETWORK GAME

论 文 作 者：云淼
领　　域：计算机
研 究 方 向：计算机应用技术
申 请 学 位：工程硕士专业学位
指 导 教 师：肖创柏 教授
所 在 单 位：计算机学院
答 辩 日 期：2014 年 12 月
授予学位单位：北京工业大学

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京工业大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签　名：云淼

日　　期：2015　年 1　月 9　日

关于论文使用授权的说明

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签　　名：云淼　　日　　期：2015　年　1 月　9 日

导师签名：肖创柏　　日　　期：2015　年 1　月 9　日

摘 要

网络游戏在中国经过数十年的发展，现在已经成为一种新的娱乐热点。数据显示，2014 年上半年，中国网络游戏市场规模环比增长 10%，全行业市场规模达到 518.1 亿元人民币，全年预计市场规模将超过千亿。而2008 年上半年的市场规模仅为 84 亿元，我们可以对比看出，中国的网络游戏市场的规模在 6 年间已经增长了将近 6 倍。而以此预计未来 3 年，中国游戏市场规模将在 2017 年将达到 2000 亿元人命币。同时纵观中国的一线互联网厂商：腾讯、网易、盛大中的游戏业务的收入基本占据了整体收入的 50%。同时，2014 年上半年资本并购投资持续活跃：腾讯投资 CJGame：5 亿美元收购 CJGame 28%的股份，成为 CJGame 第三大股东。光线传媒 2.3 亿元收购仙海科技。奥飞动漫 3.25 亿元收购方寸科技、3.67 亿元收购爱乐游。游族信息借壳梅花伞。拓维信息 8.1 亿收购火溶信息。爱使股份收购游久时代。2014 上半年，我国自主研发能力进一步增强，民族原创网络游戏实现销售收入 360.3 亿元，同比增长 50.5％，占市场销售总额的 60.5%。伴随文化产业政策的日益完善和对游戏行业潜在价值的认同，我国移动游戏乃至整个游戏产业已成为资本市场追逐的热点，投融资活动更趋活跃，从业和创业的积极性也空前高涨。

对于游戏开发来者来说，根据对网络延迟要求的高低，可以将网络游戏分为非实时网络游戏和实时网络游戏。非实时网络游戏主要包括网络棋牌类游戏、益智类游戏等，这类游戏对网络延迟的要求并不高，几秒的延迟也是可以忍受的，如 QQ 游戏平台中的斗地主、麻将、台球等游戏。实时网络游戏一般对网络延迟的要求比较高，较高的网络延迟通常使得游戏不可以顺利进行。其中最有代表的两类游戏就是：第一人称射击游戏（FPS），目前最有代表的游戏产品有《激战》、《Counter Strike》等等。即时战略游戏(Realtime Strategy Game， RTS)，目前最有代表游戏产品有《魔兽争霸》、《英雄联盟》等等。

由于在网络中延迟是不可避免的存在，而延迟则是实时网络游戏开发过程中的一个瓶颈，延迟的大小往往决定着游戏的体验、游戏的实现方式、游戏的可玩性。在游戏中，如果游戏运行过程中出现跳动、拉扯的现象，将会对这类游戏的可玩性产生致命的打击。通常为我们必须采用一系列算法来减少网络延迟对于游戏性的影响。本文将会具体对这一技术进行介绍与实现。

本文首先从实时网络游戏服务器的设计特点出发，描述了网络游戏开发中的一些主要问题，然后介绍了常见网络游戏服务器架构，并结合了分布式服务器架构的特点实现了一套适用于游戏服务器的分布式架构，并对于 protocolbuffers 协议引擎，libevent 网络引擎进行了介绍，并对使用这两种引擎的具体编程方法进行了实现。然后详细描述了实时网络游戏开发中的主要问题，其中主要的问题有：不同客户端间的时间同步问题，网络延迟的问题，游戏中物体的移动同步问题，服务器客户端一致性问题。本文对这几个问题进行了描述，介绍并实现了解决这些问题常用的算法：航位推测技术（Dead Rockoning），航位推测技术能够将网络延迟对实时网络游戏的冲击减少到最小，同时航位推测技术包括预测技术，平滑处理技术，本文详细描述了这两种技术，并对该算法在游戏中的使用给出了具体的实例。同时比较了移动同步中的保守同步机制和乐观同步机制，详细描述了 Bucket、Time Warp 同步机制，并给出了 Bucket 在实时网络游戏中的详细实现。最后对于网络延迟所带来了客户端服务器不一致性问题进行了描述，并对解决这一问题的延迟补偿技术（Lag Compensation）进行了编程实现。

关键词： 实时网络游戏；分布式架构；网络同步；航位推测技术（Dead Rockoning）；延迟补偿技术；

Abstract

Online games in China after decades of development, has now become a new kind of entertainment hotspots. According to statistics, the first half of 2014, China's online game market growth of 10%, the industry market scale reached 51.81 billion yuan, the annual market size is expected to be over one hundred billion. The comparison of 2008 to work in the market size, 8.4 billion yuan, we can see that the size of the market in six years has increased nearly six-fold. This is expected in the next three years, the Chinese game market will reach 200 billion yuan in 2017, lives currency. Meanwhile Throughout Tencent, Netease, Shanda, China's first-tier Internet companies income in the basic game business accounted for 50% of overall revenue. Meanwhile, the first half of 2014 continued to be active M & A investment capital: Tencent invested CJGame: 5 billion acquisition CJGame 28% of the shares, becoming the third largest shareholder CJGame. Enlight 230 million yuan acquisition of Xian Haike technology. 325 million yuan acquisition of Alpha Technology inch, 3.67 billion acquisition of the Philharmonic tour. Family travel information backdoor plum umbrella. Talkweb 810 million acquisition of fire melting information. Aishigufen purchase travel long era. Fascinated by the network is eligible to invest 130 million yuan, the capital market is constantly chasing the market's entertainment hotspots. Major companies are constantly in increasing investment for the game industry.

For game developers to those who, according to the level of requirements for network delay can be simply divided into non-real-time online games online games and real-time online games. Non-real-time online games including network chess games, puzzle games, these games on the network delay is not high, a few seconds of delay can be tolerated, such as QQ game platform Landlords, mahjong, billiards other games. Real-time online games generally require a relatively high network latency, high network latency is usually makes the game not smooth. One of the most representative of the types of game is this: First Person Shooter (FPS), currently the most representative of the game products are

"fighting", "Counter Strike" and so on. Real-time strategy game (Realtime Strategy Game, RTS), currently the most representative games of "Warcraft," "Heroes Union" and so on.

Due to delays in the network is inevitable existence, so that the delay is real-time network game development process of a bottleneck, the delay often determines the size of the gaming experience, the game's implementation, gameplay. In the game, if the game is running in there beating, pulling phenomenon, will produce a fatal blow to this type of gameplay. Usually we have to adopt a series of algorithms to reduce network latency effects for the game. This article will introduce specific to this technology and implementation.

Firstly, the design features of real-time network game server starting online game developer describes some of the main issues, and then describes the common network game server architecture, combined with the characteristics of a distributed server architecture to achieve a suitable game server distributed architecture and protocol engine for protocolbuffers, libevent network engine characteristics were introduced, and the specific programming methods are described. Then according to real-time network game development major problem is described in detail, the main problems are: time synchronization between different clients, network latency issues, mobile synchronization objects in the game, the consistency of the client server . This paper describes these few issues, and describes commonly used algorithms to solve these problems: dead reckoning technology (Dead Rockoning), dead reckoning technology can impact network latency real-time network game is reduced to a minimum, while the dead speculation bale forecasting techniques and smoothing techniques, this paper describes in detail the two technologies, and the algorithm used in the game gives specific examples. Compare mobile synchronization while conservative and optimistic synchronization mechanism synchronization mechanism, a detailed description of the Bucket synchronization mechanism, and gives detailed Bucket achieve real-time network game. Finally, the network delay for the client-server brought inconsistencies are described, and the delay compensation technology to solve this problem have been programmed.

Keywords: Real time network game; Distributed architecture; network synchronization; dead reckoning Technology (DeadRockoning); Delay compensation technology;

目 录

第1章. 绪论

1.1. 网络游戏概述

1.1.1. 网络游戏的分类

网络游戏的分类来看，目前主要分为：大型多人在线网络游戏、即时战略游戏，第一人称设计游戏，棋牌休闲游戏。

- 大型多人在线网络游戏（Massive Multiplayer Online Role Playing Game）

大型多人在线网络游戏我们也简称为 MMOPRG，主要游戏特点是：架构了一个虚幻或者类现实的网络世界，在此类游戏中，我们关注的重点是多人玩家的交互，其中一个主要的算法则是 AOI(Area of Interest),区域多人广播，在本文不做过多叙述。[1]

- 即时战略游戏（RTS）

即时战略游戏是战略游戏发展的一种形态，对于玩家而言，所有的操作都是有着强烈的“即时进行”感觉，RTS 相比 MMORG，侧重的实时性，客户端的每一个操作指令，服务器对每一次指令判断都必须公平，真实，及时。在这里同步问题便是游戏的主要问题。同步的算法也是本文描述的重点之一。

- 第一人称设计游戏（FPS）

对比其他游戏，FPS 游戏中的通信量并不大，但是实时性却要求非常高，对延迟特别敏感。鉴于其本身的特点，游戏中各个目标对象的位置信息非常重要，一致性在 FPS 游戏中是最为看重。在这里我们主要论述游戏中的延迟补偿技术。

- 棋牌休闲类游戏

棋牌休闲类游戏相比以上几种，则可以称之为“轻度”游戏，这类游戏主要是对现实世界中游戏的电子化，网络化，同时会有大批量的玩家，玩家的通信量则更加少，这种游戏我们主要注意的是整个游戏系统的架构，在本文不做过多叙述。

1.1.2. 国内网络游戏的发展历史与现状

从 1998 年开始，中国的游戏市场被打开，单机游戏在中国正式起步，其代表性的作品有《仙剑奇侠传》。经过几年的发展，在 2001 年达到了顶峰，年度市场规模为 2.1 亿元。而正当国内开发商还在发展单机游戏时，国外的网络游戏迅速发展了起来，直到 2001 年中国的网络游戏市场被韩国网络游戏迅速占领，使得国内众多游戏厂商苏醒，调整市场目标，大力发展网络游戏。直到 2003，国内网游开始反攻国外网游，其中的代表性作品有目标软件《天骄》、金山软件《剑侠情缘》等，这一年国内的厂商树立起了国内游戏发展的骄傲榜样。

到了 2005 年，单机游戏市场陷入低谷，年度市场的收入仅仅为 0.7 亿元。但是，在市场的另外一边，越来越多的单机游戏开发商迅速转型成网络游戏开发商。

直到 2006 年 1 月完美时空公司大型 3D 网游《完美世界》正式进入商业化运营，《完美世界》受到了全国各地玩家的普遍青睐，短时间内用户线数量一路飙升，在线人数超过 30 万，注册人数已经达到 700 万。同年 7 月，北京完美时空与日本著名网络游戏运营公司 C&C Media 联合举行《完美世界》日文版版权代理签约发布会。这也是“中国网游首次出口日本”。这一年国内网游市场迅速发展，市场的辉煌前景也逐渐显露出来。

2007 年，完美时空成功登陆美国纳斯达克股票市场。同年 5 月份，搜狐旗下子公司-搜狐畅游商业化运营《天龙八部》，这款游戏改编自极受欢迎的金庸名武侠小说《天龙八部》。7 月份，搜狐财报公布《天龙八部》公测 50 天在线人数超过 40 万，同时畅游与越南最大的电信运营商 FPT Telecom 签约，授权其在越南独家运营《天龙八部》，10 月《天龙八部》海外版荣获越南大奖，游戏在线人数位列越南三甲。

2008 年，由美国金融危机席卷全球，很多行业和企业都受到了冲击。国外的网络游戏公司不断缩减投资，同时裁员降薪也不在少数，以降低成本。与之形成鲜明对比的是，国内网络游戏行业依然良好稳定在发展，同时还进军国外市场，得到了不错的海外收入。国内的游戏厂商的研发力度、人才储备也都在不断的增大、扩张。表现出一派欣欣向荣的局面。盛大、九城、网易、搜狐等多家网游公司都在 08 年启动了大型校园招聘计划。网游行业的竞争日益激烈，对优秀人才的需求量也不断增加，网游企业求贤若渴，纷纷开出丰厚的条件来吸引人才。

到了 2009 年，中国网络游戏市场规模达到 271 亿元，同比增长

30.2%。在 2009 年 6 月由完美时空旗下子公司完美时空文化传播有限公司投资拍摄，章子怡、范冰冰、苏志燮、林心如、何润东、姚晨等国内外知名影星主演的新型爱情喜剧片《非常完美》于 2009 年暑期正式上映。同时完美时空首次参加了于美国洛杉矶举行的 E3 大展，这同时也是中国国内游戏企业首次参展 E3。而畅游的《天龙八部》最高同时在线人数突破 80 万，同时，畅游有限公司在纳斯达克全球精选市场上市交易（CYOU），畅游成为中国互联网在 2009 年的第一个 IPO，搜狐也由此成为同时拥有门户业务和网游业务两个美国上市公司的双子星，这在中国互联网企业中还是第一例。同年畅游首家引进 Cry Engine3 引擎，同时签约 BigWorld 引擎 全面进军 3D 领域。12 月中青宝网 IPO 申请顺利过会，成 A 股首家网游 IPO。种种事件表明，国内自主研发的民族网游在一步一步扩大竞争优势，资本市场也越来越亲耐这一片迅速发展的市场区域。

2010 年，中国网络游戏市场进入了盈利瓶颈阶段，经历了网络游戏的疯狂发展后，用户的消费逐渐趋于理性化。同时，游戏产品内容同质化较为严重，游戏营销手段相对贫乏，缺少对网络游戏用户的有效刺激。而在这一年，SNS 社交网站雄起，开心网、人人网、QQ 等。于此同时，开心网模式带起了一股网页游戏的潮流，其代表作《偷菜》更是席卷大江南北，渐渐成为了一种全民运动。2001 年 4 月，腾讯宣布代理网页游戏《七雄争霸》。更在年末《七雄争霸》月收入已经超过 1 亿人民币，《七雄争霸》已经毫无争议的成为了当时国内第一大网页游戏。而百度游戏事业部总经理魏洪蕊在接受媒体采访时表示，百度将于未来半年斥资至少千万元人民币，战略投资网页游戏公司。

2011 年中国网络游戏市场规模将达到 414.3 亿元，较去年增长 18.1%。5 月，南京军区有关部门与无锡巨人网络科技有限公司历时 2 年共同开发的军事游戏《光荣使命》正式完成。12 月，国防部网站报道称，《光荣使命》已经开始成为部队训练课目。标志着国家对于网游行业的政策已经由谨慎转为支持。也反应出国内确实需要一款自己的军事网游。这一年中国的网游市场上出现了许多前所未见的变化，从国内第一家上市网游公司盛大宣布私有化，到腾讯入股金山并获得董事席。国内游戏人也在不断的打破身边的牢笼，不断的寻求新的发展和机遇。

2012 年，游戏行业实现全面互联网行业化，网页游戏迅速兴起，而网页的低门槛特性吸引了大量互联网企业，同时也吸引了大量的中小创业团队。从网络游戏行业格局来看，游戏行业结构正在发生变化，网页游戏、社交游戏、移动游戏，逐渐被资本市场目光汇聚。智能手机的兴起，让中

国移动网络游进入了行业成长期。碎片化时间、弱联网、轻社交等概念受大量用户的欢迎。

到了2013年，则是游戏市场并购的一年。A股上市公司共有16起重大游戏并购事件，总估值约 222 亿元，资产出卖方合计获得 93.77 亿现金和322 亿股权市值。13 年，国内 A 股市场的涉及网络游戏概率的公司已然达到了47家。2013年中国网络游戏市场规模逼近900亿元，同比增长33%，其中腾讯的游戏营收超过300亿元，独家占据2013年全国游戏收入的三成以上。综合来看，未来的智能移动终端游戏市场的争夺将会越发激烈。

2014 年 1-6 月，中国游戏用户数量 4 亿人，同比增长 9.5%。2014 年 1-6 月，中国游戏市场实际销售收入达到 496.2 亿元，同比增长 46.4%。中国游戏市场实际销售收入构成如下：客户端游戏市场实际销售收入255.7亿元，网页游戏市场实际销售收入 91.8 亿元，移动游戏市场实际销售收入125.2亿元，社交游戏市场实际销售收入23.4亿元，单机游戏市场实际销售收入 0.1 亿元。

可以看出中国网络游戏行业正发生着深刻的变革。主导网络游戏行业的客户端游戏市场发展越发成熟，集中度日益提高，MOBA、射击等细分市场也逐渐成为各大厂商争夺的战略领域；网页游戏市场随着研发技术的发展、行业集中度的提升，市场的竞争也更为有序，寻求产品差异化也成为各大厂商的首要战略；移动游戏市场正处于爆发式增长的时期，资本的大量涌入、大型游戏厂商移动业务的开展、海外企业逐步参与竞争等都不断改变着移动游戏市场的竞争格局。

越来越多的专业游戏开发商和发行商介入网络游戏，一个规模庞大、分工明确的产业链也在逐步形成。国内的相应开发技术也在慢慢的积累中。本文将对实时网络游戏中的部分关键技术做出一个整理与论述。

1.2. 国内外的网络游戏技术研究现状

在 21 世纪开始之初，网络游戏商业化的进程在国外明显加速，这里我们介绍一下国外商业化引擎的著名产品。

- Unity3D 游戏引擎， Unity 公司在 2001 年开始开发自己的游戏引擎。当时的主要诱因是创建游戏，而创建这些游戏的基础，便是创造一套良好的工具。于是 unity 游戏工具集开始发展起来。直到现在，unity3D 以及成为了一个跨平台的，可以让让玩家轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多

平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。

- BigWorld 游戏引擎，澳大利亚 BigWorld Pty. Ltd 所开发的 BigWorld 引擎，由服务器软件、内容创建工具、3D 客户端引擎、服务器端实时管理工具组成，为致力于构建富有创造力的一流的新一代网络游戏的开发商降低了开发周期和成本。
- Source 游戏引擎，代表作：《半条命 2》、《反恐精英》Source 引擎(起源引擎)由 Vavlve 公司研发，包括了 3D 图像渲染、材质系统、AI 人工智能计算、Havok 物理引擎、游戏界面、游戏声效等各个组件，而且创造性地使用了模块化理念，是当今主流引擎之一。

若干年前，这些来自国外的网络游戏产品抢滩大陆市场，众多由国外游戏开发商研发的网络游戏大举进军国内网络游戏市场。多年之后的今天，在国内游戏开发商们的不懈努力之下，国产网络游戏产品在市场上的比重已经大幅度提升，完全超越了国外的网游产品。同时：自主研发的引擎也越来越多，越来越成熟。其中富有代表性的有：

- Angelica 3D 游戏引擎：2004 年完美世界现主要基于自主研发的 Angelica 3D 游戏引擎、Cube 引擎以及 Eparch 2D 引擎为平台开发各类网络游戏，陆续推出了《完美世界》、《武林外传》、《诛仙》等网络游戏，《完美世界》更是成为了第一款进军韩国网络游戏市场的国内网络游戏。
- OverMax 游戏引擎：出口新加坡的 OverMax 游戏引擎，作为国内知名游戏厂商目标软件自主研发的作品，开创了我国首次游戏出口开发引擎的先例，这款打造了目标软件首款 FPS 网络游戏产品《MKZ》和经典游戏续作《天骄 3》的引擎，凭借两款游戏精美的画面，强大的性能博得了新加坡游戏厂商 Visual Factory 的青睐。该引擎项目是在 2005 年 8 月国家科委与目标软件联合成立的“网络游戏核心技术开发及平台化”课题组的成果，该课题组也是北京科技计划中的“数字娱乐软件共性技术研发与平台支撑服务”项目的核心部分。
- 剑网 3 游戏引擎：，金山凭借自己的研发实力，完全自主研发，获得国家 863 计划支持的图形 3D 引擎，经过不断的完善 3D 引擎的功能和效果，《剑网 3》的即时演算效果已经达到了电影水准。从游戏截图可以看到，草地树木演示的细节表现、林间的光影效果、水面的动态引擎技术，都已经达到了国内网游的顶级水准。

由以上国内外游戏引擎的介绍我们可以看出，国内游戏引擎的发展速

度突飞猛进，在很多方面也基本地向国外引擎不相上下。而其中国内游戏引擎中的实时网络游戏引擎发展则显得相对落后。对比 Source 游戏引擎，国内并没有一款游戏引擎能够在实时网络游戏类型中与此引擎抗衡。由于实时网络游戏中的核心技术和算法都掌握在国外手中，其中很多技术也并未对外公开，因此国内网络游戏中的实时网络游戏基本是国外厂商的天下，国内厂商不得不支付高额的授权费用以及代理费用来引入到中国，因此掌握实时网络游戏中的关键技术，加快国产网络游戏的发展也是一项不可或缺的工作。

1.3. 本文的主要内容

本论文对一个实时网络游戏中的关键问题进行了阐述，并且，对这些关键问题所对应的解决技术进行了比较与技术实现，最后基于这些实现方案，完成了一款实时网络游戏的开发工作。本文的主要内容包括：

- 类分布式的游戏服务器架构设计，提出了一种适用于现阶段网络游戏的类分布式架构。
- 描述了实时网络游戏中的关键问题之一：网络延迟问题，并阐述航位推测技术在处理实时网络游戏中的的应用，该技术主要包括预测技术和平滑处理技术。同时实现了基于立方体抖动的航位推测技术。
- 移动同步技术在实时网络游戏中的应用，该技术目前来说分为两大类：保守同步技术和乐观同步技术，并最终实现了乐观同步技术中的一种桶同步（Bucket）机制。
- 射击游戏中的延迟补偿技术，该技术也是一种隐藏网络延迟的有效手段。该技术使得服务器在判断射击游戏的精确度方面大大提高。同时对于游戏外挂也起到了一个很好的防护作用。

通过以上的理论算法，本文总结了实现了一个适用于实时网络游戏的类分布式系统架构。并利用航位推测算法，桶同步机制，延迟补偿技术等实现了一个 2D 实时网络射击游戏。

1.4. 本文的组织结构

全文共分六章，具体的章节安排如下：

第一章 简单介绍了网络游戏及网络游戏的分类，以及国内网络游戏的

市场和发展现状；描述了国内外网络游戏引擎发展的差异，指出了论文的主要描述内容和在章节方面的安排。

第二章 介绍了网络游戏中服务器的设计，对不同的架构设计进行了对比比较。网络游戏服务器架构主要分为单服架构，多服架构，分布式架构。并且对每种游戏服务器的优缺点进行了详细描述。

第三章 主要描述并实现了网络游戏中的基础技术问题，对这些基础问题的实现方式、编程方式做了实现。包括：服务器整体的架构设计，protocolbuffers 网络协议框架的编程及应用与实现，libevent 网络框架的编程及应用与实现。

第四章 主要描述了在实时网络游戏中会遇到问题，以及解决技术，描述了时间同步问题，网络延迟问题，移动同步问题，安全问题，以及网络带宽消耗的问题。本章节对非关键问题：时间同步问题的解决技术 NTP 协议进行了简单描述，同时对网络游戏中的带宽消耗问题给出了详细解决方案与具体技术实现，对于实时网络游戏的关键问题将在下一章节中描述。

第五章 主要实现了实时网络游戏中的关键技术算法，对于延迟问题，移动同步问题的解决方案进行了阐述，描述了航位推测算法的算法及其具体使用，比较了移动同步技术中的保守同步技术和乐观同步技术，并详细介绍了乐观同步技术中的 Bucket 技术，同时对其中技术解决方案进行了实现。

第六章 主要对上文提到的系统进行验证，通过压力测试，验证了服务器整体框架的稳定性以及可用性，通过客户端对比表现，验证了核心算法中的航位推测算法及其平滑算法的可用性以及其的表现。

最后，在本文结论部分进行了全面的系统总结，并对指出了本论文中所未涉及到的技术，并展望未来实时网络游戏技术的发展方向。

第2章. 网络游戏服务器的设计

2.1. 简述

一款实时网络游戏的要能够得到良好的表现力，网络游戏服务器是则是首要考验。实时网络游戏在架构设计上要考虑的细节比一般类网络游戏复杂度高很多，由于实时游戏对数据计算的正确性和一致性要求非常高，同时它也要应对大量的同时在线用户，不仅如此，在追求性能的同时还需要兼顾反作弊、反外挂等等需求。

由于网络延迟也是不可改变的事实，目前 Internet 上在大多数情况下延迟都是及其不稳定的，同时网络带宽也是有限的。因此，如何处理客户端之间、客户端与服务器之间数据的交互，如果在保证用户游戏体验的同时尽量减少数据通信量，也是在网络游戏服务器设计开发过程中需要考虑的一个重点。本章将对网络游戏中的服务器设计进行简单的比较。

2.2. 网络游戏服务器的架构

一般来说，我们可以把游戏服务器的架构主要分成以下三类：单服务器结构、多服务器结构、分布式服务器结构[2]。

2.2.1. 单服务器架构

从最早的韩国游戏开始，单服务器架构非常流行。如（图 2-1）由一台服务器里的单个进程处理所有的信息，包括：客户端的 socket 连接，游戏的逻辑，数据的逻辑等全部功能，根据当时的硬件配置这样的一台服务器容纳人数大概在 1000 人左右。单服务器架构开发简单，逻辑的复杂度小。同时带来的缺陷就是：服务器负载量小，功能扩展复杂。现在随着硬件设备的升级，大量用户的涌入，单服务器架构已经很难满足现在市场的需要。

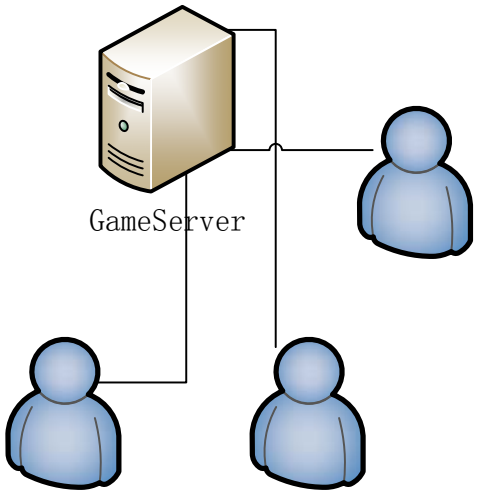

图 2-1 单服务器架构

Fig 2-1 Signal server architecture

2.2.2. 多服务器架构

大量用户的涌入，使得我们不得不提高服务器的负载上限，这样多服务器架构就应声而出。现在很多游戏服务器也是基于该类结构的设计。根据游戏类型的不同，我们大致可以分为以下两种：按地图区分服务器和按游戏功能区分服务器。

对于 RPG[1]游戏来说，游戏中存在很多区域，整个游戏世界由各式各样的地图组成。于是我们可以看出，按地图来区分游戏服务器是一个比较合理的负载均衡方案。按地图区分就是将整个游戏地图划分为固定的几个区域，每个区域由一台服务器来负责管理。一般来说，服务器分为以下三个部分：地图服务器、世界服务器、游戏外围逻辑服务器。

[1] RPG：角色扮演游戏

- 世界服务器负责世界地图的管理，同时它也管理者地图服务器的分配工作，以及数据从地图 A 转移到地图 B 的中转转移功能。
- 地图服务器负责该地图区域类的一切功能，包括区域类物体的刷新，物体的移动同步，怪物的 AI[2]，NPC[3]等等所有可能存在的信息，同时最重要的一点，不同地图之间的数据是完全隔离开的。这个时候，如果一名玩家从一个地图切换到另外一个地图时，地图服务器将会把这名玩家的数据发送给世界服务器，世界服务器再通知目标服务器这名玩家的到来。可以看出，由于存在不同服务器上数据上的交互，当客户端在切换地图的时候都会出现一个等待的界面，然后才能继续游戏。
- 其他外围逻辑服务器则是负责玩家的登陆，聊天，邮件，等等其他的一些游戏基础功能。

我们可以看出，按地图这种划分方式结构清晰、实现难度小、效率也相对比较高。但是我们也可以看到其的不足之处：地图区域是静态分配，且不能改变。由于玩家在各个地图的分布式非常随机的，同时也随着游戏的进展，地图的热点随之改变，游戏地图服务器的负载总是呈现随机的过载，而另外一些区域的服务器则相对来说非常空闲。同时可以看出，一旦世界服务器出现宕机的话，会导致地图服务器与世界服务器之间的交互无法正常工作。而且一旦地图服务器宕机，则有可能这个地图上的数据都会遗失。

对于其他游戏类型来说，如 ACT[4]类的游戏，则大多数的采取按功能划分服务器思想来进行架构设计。从名字可以看出，这是一种面向服务的设计思想，游戏逻辑中常用的、密集型的功能分离开来，单独作为一个服务器进行控制操作。例如：怪物 AI 单独一个服务器，物品掉落单独一个服务器，玩家交易单独一个服务器，这样按照逻辑进行服务器分离。这样做的好处是，当一个服务器出现问题的时候，并不会影响其他的模块运行，而且扩充性相对来说优秀便捷。但是一旦重要的服务器宕机，也将是一个灾难性的的结果。比如 AI 服务器宕机，则 AI 相关的逻辑将会出现异常。对比按地图划分的服务器来说，各有千秋。

[2] 人工智能(Artificial Intelligence)
[3] Non-Player-Controlled Character，非人控制玩家角色
[4] ACT：动作类游戏

2.2.3. 分布式服务器架构

分布式系统遵循几个基本原则：CAP 原理，CAP 原理有三个要素，数据一致性（Consistency）、可用性（Availability）、分区容忍性（Partition tolerance）。同时在分布式系统中这三者不能兼顾。

数据一致性：数据一致性意味着，一旦数据库中的某个数据更新，其效果对之后的任何操作可见，不论数据库本身分布在多少个节点上或拥有多少个副本，数据在多节点上的更新具有原子性[3]。

可用性：可用性理论上是与性能、速率无关的一个概念，无论是服务器、客户端设备性能还是网络速率。系统具有“好的响应性能”、“速度”或者“每一个操作总是能够在确定的时间内返回”等关于可用性的描述，严格地看，可用性可以理解为“所有被请求的在线节点都能正常返回数据”。

分区容忍性：分区容忍性是分布式架构的重要特征。在分布式环境下，分区容忍性意味着出现故障节点或者连接部分节点的网络中断时，系统仍能正常响应。由于硬件总会出现故障，分布式环境下分区容忍性就必须存在，因此是否具有分区容忍性从定性的角度来看可以理解为是否为分布式架构，进一步可以理解为是否具有水平扩展性，还有人引申为“可靠性”，增加节点数量同时具有增强硬件可靠性的效果。

如此可见，游戏服务器架构也将会向分布式服务器架构演变，分布式架构的种种优越性能够有效的减低人工维护的成本，并降低服务器宕机对于用户的体验的影响，本文将在后续章节对于分布式架构在游戏服务器中的应用给出具体实现。。

2.3. 本章小结

本章主要介绍了网络游戏的架构设计，以及现有常用架构分类：主要分为单服务器架构，多服务器架构，分布式服务器架构。并详细介绍了单服务器架构和多服务器架构的具体应用场景。同时也详细介绍了分布式架构的特点，分布式架构对于互联网应用，是一个非常成熟的领域，但是分布式服务器对于网络游戏来说，现在还处在一个初期阶段。分布式系统具有良好的可伸缩性，非常适合处理拥有大规模场景的网络游戏。但是分布式系统设计比较复杂，它需要将任务进一步细分为可独立并行执行的子任务。同时网络游戏服务器处理的任务极多，而且任务之间有着种种联系，

任务的变化速度也极快，同时还需要考虑到游戏的可靠性、安全性等等。所以对游戏逻辑任务的划分非常困难，现阶段来说并不能将游戏逻辑细分为可独立并行执行的子任务，于是只能是模仿分布式架构的思想，通过各种技术手段，来打造一个类分布式架构的游戏服务器架构。后文将详细介绍这种类分布式服务器架构的实现。

第3章. 网络游戏中的基础设计与实现

3.1. 引言

本章对实时网络游戏的基础技术进行了设计与实现。主要包括了类分布式系统在网络游戏服务器中应用，以及协议框架 protocolbuffers 在网络游戏架构中的应用与网络底层框架 libevent 的使用。深入分析了 libevent 的网络引擎的基本体系架构，最后对 protocolbuffers 以及 libevent 库在游戏编程方面的实现方法进行了简介。

3.2. 分布式游戏服务器架构设计与实现

3.2.1. 设计思路

在上文中，我们提到分布式系统遵循几个基本原则：CAP 原理。而通过对这上三个特点的分析，结合网络游戏系统特点来说，分区容忍性是最基本的要求，一个频繁宕机的服务器对于网络游戏来说，将会导致大量用户的流失。而高可用性也是重中之重，用户的体验，游戏效果在用户口碑中的传递也和此关联密切。对于数据的一致性要求，由于网络游戏中玩家都是存在于某个“世界”中，而这些“世界”是相互独立，所以我们可以不对数据的一致性要求进行过多的考虑。

3.2.2. 服务器整体框架的设计与实现

现在的网络游戏服务器一般是用一组服务器以一个整体对外提供服务，其中有些会与客户端有直接的 TCP 连接，有些则没有。为了保证分布式系统中的特点之一：高可用性，常见的做法之一是设置专门的网关服务器，它负责保持与客户端之间的 TCP 长连接，承担客户端与服务器组之间消息的中转功能。

网关服务器的主要职责是将客户端和游戏主题逻辑服务器隔离，客户端程序直接与这些网关服务器通信，并不需要知道具体的游戏服务器内部架构。客户端只与网关服务器相连，通过网关服务器转发数据包间接地与游戏服务器交互。同样地，游戏服务器也不直接和客户端通信，发给客户

端的协议都通过网关服务器进行转发。

这样的好处有：

- 只把网关服务器暴露给外网，负责维护将内网和外网隔离开，使外部无法直接访问内部服务器，保障内网服务器的安全，提高整个系统的安全性。
- 网关服务器负责解析数据包、加解密、超时处理和一定逻辑处理，这样可以提前过滤掉错误数据包和非法数据包
- 服务器组内的其他服务器不用处理大量的网络连接，在玩家由于某些逻辑操作分布在不同服务器上时，不需要断开与网关服务器的连接，玩家的数据在不同游戏服务器间的切换是内网切换，切换工作瞬间完成，玩家几乎察觉不到，这保证了游戏的流畅性和良好的用户体验。
- 可以用多个网关服务器来分散负载，即使其中某个网关服务器宕机，在客户端我们也可以进行特殊处理，在某个网关服务器宕机时，无缝切换到另外一台网关服务器，并且宕机的网关服务器可以迅速恢复。这样也提高系统的可用性[4]。

根据以上特点，我们最终设计的网络游戏服务器架构如下（图 3-1）：

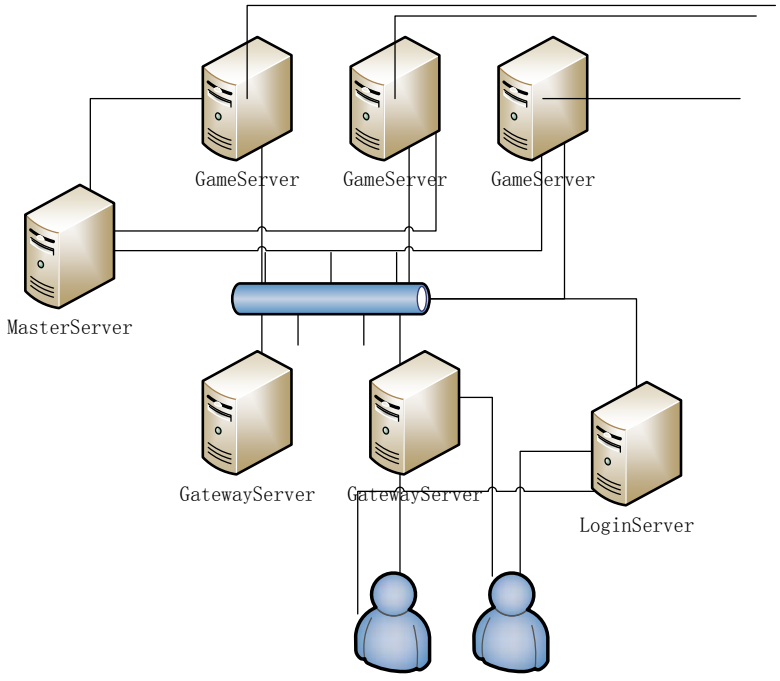

图 3-1 分布式游戏服务器架构

Figure 3-1 Game server with distributed architecture

可以看出，一组服务器包括 LoginServer（LS）、GatewayServer（GWS）、GameServer（GS）、DBServer（DS）和 MServer（MS）等多种服务器。GWS

就是网关服务器，一般一组服务器会配置多台GWS，因为GWS需要保持大量游戏客户端的 socket 连接，因此 GWS 的稳定性非常重要，为了减低单个 GWS 宕机的风险，所以将会配置多台 GWS，同时还有一个功能，就是 GWS 也承担着负载均衡的职责。

下面将对各个服务器进行一些详细的介绍。

- LoginServer（LS）

LS 是登陆服务器，一般来说，我们只需要一个就够了，它的主要功能是验证玩家的用户名和密码，同时依据特定的路由规则，把客户端分配到固定的 GWS 中去。玩家连上 LS 后，发送加密后的用户名和密码到 LS，这个时候 LS 访问 DS，得到用户的基本数据，包括用户名、密码、服务器组 ID 等等信息，并在收到这些信息后进行验证，验证通过后，将会把这些基本信息返回给客户端，客户端再根据收到的这些基本信息进行后面的游戏逻辑。这里有一个重要的信息会被返回，一个登陆 token 和 GWS 的 IP 和端口号。

- GatewayServer (GSW)

GWS 是网关服务器，在上一步中，客户端连接到了 LS 并通过了验证，拿到了登陆 token 和 GWS 的 IP 端口信息，这个时候客户端断开与 LS 的连接，构造一个登陆 GWS 的协议，其中带上收到的登陆 token 来登陆 GWS。GWS 对客户端发生过来的登陆 token 进行验证，如果验证通过，则 GWS 会一直保持着该客户端与 GWS 的连接。GWS 主要负责如下几个功能：维持游戏客户端的数据通信；同时它还要对客户端所发过来的通信协议进行解密和校验处理，并过滤掉错误的数据包；对连接上来的客户端进行“心跳”，如果在一个设定的范围内客户端“心跳”超时了，则可以认为客户端出现了某种问题，主动踢掉客户端，释放网络资源；最后一个功能，也是逻辑最为核心的一个功能，对不同类型的客户端协议包，路由到后端不同的逻辑模块上去。我们可以看到 GWS 同时还会与服务器的各个逻辑模块相连，如 GS、MS、DS 等。

- GameServer(GS)

GS 是游戏服务器，它是逻辑服务器的统称，它主要负责游戏中各式各样的逻辑处理。在很多时候，它都包含了一系列的逻辑处理模块，如：装备系统，技能系统，地图系统、NPC 系统、物品系统等等。客户端的协议包会通过 GWS 路由到各个相应的系统中去处理，处理完成之后又返回给相应的 GWS，再返回给客户端。这些系统组合在一起，控制着整个游戏的具体逻辑运算，所以说 GS 将是整个游戏架构中最为核心的一个，为了保证这

个服务器的高可用性，我们将会配合共享内存在一起使用，共享内存技术将会在后续详细介绍。

- MServer(MS)

MS 表示主控服务器，每一个服务器集群会有一个或者多个 MS，它主要负责维持多台 GS 之间数据的转发和数据的广播。上文中提到，GS 数量庞大，同时也会有很多游戏逻辑需要跨越不同的 GS 来进行数据交互和运算，这个时候 MS 将承担这个中间角色。比如跨服战斗，不同的 GS 内的玩家数据将会统一汇合到 MS 中，再由 MS 统一分配到一个新的 GS 中去进行跨服逻辑。

- DBServer(DS)

DS 表示数据缓存服务器，它主要的功能是缓存数据库中的热点数据，保证这些热点数据能够快速的读取和保存。同时也是为了统一其他服务器对数据库的访问接口。比如数据库可能会用到 mysql 的主、从数据库，或者数据表又依据时间来进行切割等等，这些访问逻辑都统一放入 DS 中，其他的服务器只需要简单调用接口就可以了。DS 保持着数据库的简洁性，以及数据访问的高效性。

3.2.3. 共享内存模块设计实现

另外，还有一个非常重要的模块，它保证着我们游戏架构的 CAP 原则中的高可用性。这个模块，我们可以称之为：共享内存模块（如图 3-2）。

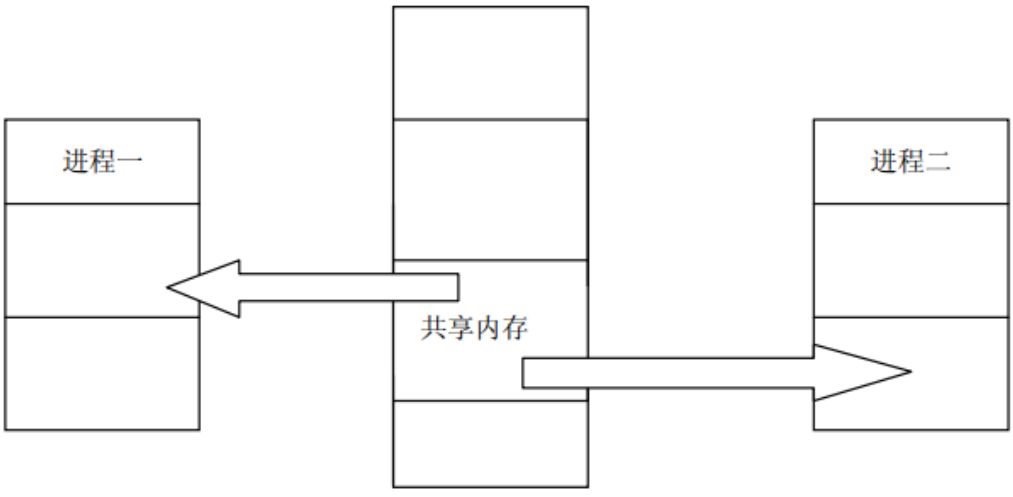

图 3-2 共享内存与进程的通信

Figure 3-2 Shared memory communication to other process

共享内存(shared memory)是 Unix 下的多进程之间的通信方法，这种方法通常用于一个程序的多进程间通信，实际上多个程序间也可以通过共享

内存来传递信息。

对于分布式游戏架构中的共享内存模块，是指在内存中开辟一块共享内存，游戏服务器及数据存储程序都能对其进行读写操作，游戏服务器定时把数据写入到共享内存中，数据存储程序定时把数据存入数据库中。它的特点是游戏服务器数据更新迅速，并且尽可能保证了数据的安全。是典型的用内存换效率的做法。同时也达到了分布式架构中的分区容忍性要求。

在上文的架构中，我们提到 GameServer 是主要负责游戏的逻辑数据处理，在一个 GameServer 中，我们设计为它由如下图所示的几个主要进程组成：

我们可以简单的认为游戏中的所有逻辑服务器都将和 shared memory 连接(如图 3-3)，也即是说如果 shared memory 足够稳定性，GameServer 即使由于各种问题宕机，我们也可以保证数据不受任何的影响，因为数据都在共享内存中，而其他的 GameServer 一样可以访问到。其实由于 shared memory 的设计简洁，并没有过多的逻辑，我们可以认为它是足够稳定的。

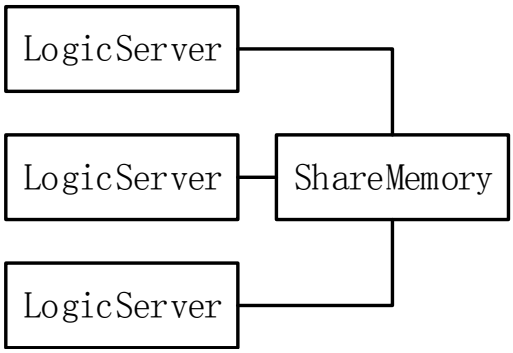

图 3-3 ShareMemory 与其他进程交互

Fig 3-3 ShareMemory interact with other processes

共享内存让我们进程可以直接读写内存，不需要任何数据的拷贝。为了在多个进程间交换信息，内核专门留出了一块内存区，这段内存区可以由需要访问的进程将其映射到自己的私有地址空间。因此，进程就可以直接读写这一内存区而不需要进行数据的拷贝，从而大大的提高了效率。

共享内存模块主要使用了以下几个系统调用：

shmget 函数：

```
#incldue <sys/types.h>
#include <sys/ipc.h>
#include<sys/shm.h>
```

函数原型：int shmget(key_t key,int size, int shmflg)

参数： key：IPC_PRIVATE

size：共享内存区大小

shmflg：同 open 函数的权位，也可以用八进制表示法

返回值： 成功：共享内存段标识，出错：-1

shmat 函数：

#incldue <sys/types.h>

#include <sys/ipc.h>

#include<sys/shm.h>

函数原型：char *shmat(int shmid,const void *shmaddr,int shmflg)

参数：shmid：要映射的共享内存区标示符

shmaddr：将要共享内存映射到指定位置（若为 0 则表示把这段共享内存映射到调用进程的地址空间）

shmaddr：SHM_RDONLY：共享内存只读；默认 0：共享内存可读写

返回值：成功：被映射的段地址

出错： -1

shmdt 函数：

#incldue <sys/types.h>

#include <sys/ipc.h>

#include<sys/shm.h>

函数原型：int shmdt(const void shmaddr)

参数：shmaddt：被映射的共享内存的段地址

返回值：成功：0 失败： -1

为了更好的使用共享内存模块，我们在设计时需要考虑以下几点：

1. 结构一致性：指的是模块中的存储结构尽量与数据库中表的字段类型等一致，方便存储及加载。例如数据库中的人物属性表，每一行表示一个人物的属性数据，那么，游戏中人物属性结构就尽量与数据人物表中的字段保持一致。
2. 数据的分离性：即游戏中的产生的数据结构与数据存储结构分开。存储结构，即共享内存中使用的数据结构，游戏中产生的数据结构，为游戏服务器中使用的数据结构，为何这么做呢，因为我们知道，游戏在进行过程中会产生很多数据，而这些数据有一些是不需要存储的。比如一个角色的基本数据，假设基本数据有血量，防御

值，装备，而在游戏进行过程中，为了逻辑计算的方便，我们会把装备所带来附加数据属性也存储到角色基本数据中，这样数据管理方面，同时逻辑编写也会简单很多。但是这些附加数据属性并不需要存储到共享内存数据存储结构中，只需要存储基本数据就够了，在下次重新加载的时候，有程序统一计算出这些附加角色属性就可以了。

3. 数据的独立性：游戏中数据、逻辑层操作分开。比如一个玩家的游戏数据，我们定义为 PlayerData，来进行集中管理，而这些数据中，我们又划分为各个小数据，如 HpData，DefanceData，BufferData，同时这些小数据也对应了各个不同的功能逻辑，这样对玩家数据的更新、存储将更加清晰简单。

我们在设计我们的共享内存模块时，还需要注意如下几个问题：

- 如何保证数据的安全性

共享内存是属于系统开辟的内存空间，不属于任何进程，共享内存创建后，会对打开的程序进行计数。在 windows 系统下，记数为 0 时，系统会回收内存，而在 linux 系统下，需要使用 ipcrm 命令来回收，那怕记数为 0，系统也不会自动回收。也就是说，游戏服务器和存储程序只要不同时宕掉，数据就还会在内存中。要保证数据安全性，具体方法有：

在存储程序中加入对游戏服务器的监控，如果发现游戏服务器没反应了，则马上存储所有数据；存储程序设置启动参数，如果存储程序宕机了，可以重新启动时，不从数据库中加载数据，而直接打开共享内存，然后把共享内存的数据回写到数据库中。

- 如何保证数据的一致性

多个进程访问共享内存时，必然会造成脏数据，需要程序员来保证对共享内存操作时的锁定操作，而刚才说过，共享内存属于系统所有，没有相应的加锁函数，所以，只能由程序员自己写一个算法来处理加解锁。

- 如何判断数据的完整性

存储程序对游戏数据进行存储时，如果存储到一半出现了故障，比如停电，这个时候，数据就会不完整。可以在每条数据后面加一个值，代表当前数据存储 version，数据每次存储前对 version 加 1，如果发现数据第一条与最后一条的 version 不一致，则可确定数据存储不完整，并且根据 version 值，很快就能找到存储到哪一条数据时出现了问题。

- 如何保证数据的正确性

数据存储时，对重要的数据做 CRC 校验，并且存储校验值，下次加载

数据后，再对那些数据做同样的校验，如果校验值不一致，则可认为数据库中的数据与存储时的数据不一致了，在这个时候进行数据更新[5]。

3.3. ProtocolBuffers 的编程应用简介

3.3.1. 简介

ProtocolBuffers（以下简称 PB）是 Google 公司开发的一种数据描述语言，它类似于 XML 提供了将结构化数据进行序列化和反序列化的方法，用于数据存储、通信协议等方面。它不依赖于语言和平台并且可扩展性极强。

PB 是一种敏捷、高效、自动化的数据结构框架。同 XML 比较，比 XML 更小、更快、更简单。你一旦定义了期望的数据结构，就可以根据定义生成特定的源码，从而轻而易举地对你的数据进行读写操作，你甚至可以在不修改原来的程序源码的情况下，更新自己的之前定义的数据结构。

使用 Protocol Buffers 有三个步骤：

- 在.proto 文件中定义 message。
- 使用 protoc 编译器，对 proto 文件进行编译。
- 使用 protocol buffers message API 来读写 message。

3.3.2. 协议定义

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32,sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages,packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, floa

表格 4-1 PB 数据格式

Table 4-1 PB data type

如（表格 4-1）PB 所支持数据格式如下，在此基础上，我们定义我们

的游戏登陆协议，在协议设计中，我们借鉴 HTTP 协议的设计思想，将协议划分为协议头与协议体两部分，协议头指的是那些通用性质的协议内容：如协议号，协议长度，协议加密标志等等，而协议体就是我们的具体协议内容，在登陆协议中，我们考虑协议体必须包含的字段：用户名，用户加密后的密码，用户所登陆的服务器等。最终，我们的协议文件（LoginRequest.proto）定义如下：

```
// package 声明，用于防止名称冲突
// 此 proto 文件生成的 C++ 文件中，Login 将作为名称空间
package Login;

// 协议头（Header）的定义
message Header
{
    required int protocol_id= 1;        // 协议号
    required int length= 2;             // 协议长度
    required int magic_id = 3;          // 加密标识
}

// message 的定义
// 此 proto 文件生成的 C++ 文件中，Person 将作为一个类存在
message LoginRequest
{
    required Header header = 1;         // 协议头

   // 以下内容为协议体
    required string name = 2;           // 用户名
    required string password = 3;       // 用户密码
    optional int realm_id = 4;          // 用户区服 ID
}
```

其中的字段修饰词：required 表示：必须提供此值，否则 message 会被认为是未初始化的。optional 表示可以设置也可以不设置此值。如果未设置 optional field 将使用默认值。

3.3.3. 协议生成与协议的应用与实现

我们现在已经有了.proto 文件，接下来需要编译 LoginRequest.proto 文件。通过 ProtocolBuffer 编译器 protoc（ProtocolBuffers 的编译过程中生成了此工具）来编译.proto 文件。编译的结果是生成 LoginRequest 类。protoc 的用法范例如下：

protoc -I=$SRC_DIR --cpp_out=$DST_DIR $SRC_DIR/ LoginRequest.proto

我们需要指定：

- source 目录--$SRC_DIR
- 输出的 C++ 类的目录--$DST_DIR
- .proto 文件的路径---$SRC_DIR/ LoginRequest.proto

编译将生成两个文件：loginrequest.pb.h、loginrequest.pb.cc。接下来，我们可以使用 ProtocolBuffers message API 了。如下，我们主要使用两个 API：

- bool SerializeToString(string* output) const;
  将定义好的协议序列化成字节数组，并存储在 output 中，需要注意的是，output 中存储的内容为二进制字节流并非 text 文本。
- bool ParseFromString(const string& data);
  将 data 中的数据反序列化成协议体。

接下来，利用以上生成的内容和 api 我们的客户端与服务器端程序如下所示：

```
// 客户端
LoginRequest request;
request.set_header (XXXX);
request.set_name(“alex”);
request.set_password(“pppssswwword”);

string sRequest;
request.SerailzeToString(&sRequest);
// 然后调用 socket 的通讯库把序列化之后的字符串发送出去
// .....
————————————————————————————————
// 服务端
```

```
string sRequest;
// 先通过网络通讯库接收到数据，存放到某字符串 sRequest
// .....

LoginRequest request;
if(request.ParseFromString(sRequest)) // 解析该字符串
{
   // do logic
}
else
{
   cerr << "parse error!" << endl;
}
```

从以上代码看来，结构清晰简单，有了这种协议代码生成机制，开发人员只要维护 proto 文件就可以了，协议的修改再也不需要对具体协议代码进行修改。PB 直接生成特定语言的类，提高了开发的效率，降低了维护成本。同时它的可扩展性也非常好，具有强大的“向后兼容”性：由于 required，optional 关键字在 PB 中的定义，我们可以把一个字段从 required 变成 optional 或者反之亦然。举例来说。如上例子假设老版本中没有“realm_id”这个属性，在扩充协议时，把“realm_id”属性设置 optional 的，这样，我们的程序就可以无缝的兼容老版本的协议了。

3.4. Libevent 网络库的编程应用简介

3.4.1. 简介

Libevent 是一个轻量级的开源高性能网络库，它是用 c 语言编写的，它是一个事件触发的网络库，适用于 windows、linux、bsd 等多种平台，内部使用 select、epoll、kqueue 等系统调用管理事件机制。同时 libevent 在使用上可以做到跨平台

Libevent 有几个显著的亮点：

- 事件驱动（event-driven），高性能;
- 轻量级，专注于网络，不如 ACE 那么臃肿庞大;
- 源代码相当精炼、易读;

- 跨平台，支持 Windows、Linux、BSD 和 Mac Os；
- 支持多种 I/O 多路复用技术，epoll、poll、dev/poll、select 和 kqueue 等；
- 支持 I/O，定时器和信号等事件；
- 注册事件优先级；

libevent 库通过对不同类型平台网络函数的包装，让我们可以不再关注与 select 或者 poll 机制，而是统一的使用其封装好的函数来调用，大大减少了程序人为犯错的几率。同时 libevent 库提供一种事件机制，这种事件机制处理着各式各样的 I/O 事件，让我们能够从复杂的系统事件中抽离出来，而只用关注于系统逻辑的实现，大大提高了系统实现的效率。本文是基于最新的 libevent2.0.x 进行了具体的技术实现。在后文中我们将会详细介绍利用此库的编程方法。

3.4.2. 网络编程的应用与实现

在网络编程中，socket 的管理非常繁琐，需要去处理各式各样的 error、异常等。同时对 socket 中接收到的数据，我们又需要在内存中申请一片 buffer，用来缓存我们所收到的数据，由于对于内存的直接操作，非常容易导致内存泄露，我们不得不又得用心的去管理这块内存区域。而在 libevent 库中，bufferevent 结构体的封装，完美的把 socket 的管理和内存的管理结合到了一起。我们可以看到 bufferevent 的数据结构如下：

```
struct bufferevent {
    struct event_base *ev_base;
    struct event ev_read;//读事件 event
    struct event ev_write;//写事件 event

    struct evbuffer *input;//读缓冲区
    struct evbuffer *output; //写缓冲区

    bufferevent_data_cb readcb;//可读时的回调函数指针
    bufferevent_data_cb writecb;//可写时的回调函数指针
    bufferevent_event_cb errorcb;//错误发生时的回调函数指针
    void *cbarg;//回调函数的参数
    … // 其他省略
```

};

可以看出 struct bufferevent 内置了两个 event（读/写）和对应的缓冲区。当有数据被读入(input)的时候，readcb 被调用，当 output 被输出完成的时候，writecb 被调用，当网络 I/O 出现错误，如链接中断，超时或其他错误时，errorcb 被调用。

基于 bufferevent，我们主要使用如下几个 api：

- struct bufferevent * bufferevent_socket_new(struct event_base *base, evutil_socket_t fd, int options)
  使用 bufferevent_socket_new 创建一个 structbufferevent *bev，关联该 sockfd，托管给 event_base
- void bufferevent_setcb(struct bufferevent *bufev, bufferevent_data_cb readcb, bufferevent_data_cb writecb, bufferevent_event_cb eventcb, void *cbarg)
  设置读写对应的回调函数

通过以上 API：我们的网络框架实现如下：

- 客户端网络框架如下所示：

```
------------------------------------------------------------
// 创建 socket
int fd = socket(AF_INET, SOCK_STREAM, 0);

// 创建 event base 并绑定 bufferevent
struct event_base * evbase = event_base_new();
struct bufferevent * bufevt = bufferevent_socket_new(evbase, fd, 0);

// 设置 sock 读取事件回调函数(bufferevent_read_callback)，
// 写回调和错误回调暂不设置，并开启读取事件通知开关。
bufferevent_setcb(bufevt, bufferevent_read_callback, NULL, NULL, NULL);
bufferevent_enable(bufevt, EV_READ);

// 连接服务器，并启动事件循环
bufferevent_socket_connect(bufevt, paddr, addrlen);
event_base_dispatch(evbase);
------------------------------------------------------------
```

// socket 读取事件回调函数，每收到一条远程消息，该函数将会被调用一次。

```
void bufferevent_read_callback(struct bufferevent *bufevt, void *arg)
{
    char buf[1024];
    // 从 bufferevent 中，读取接收到的消息
    size_t sz = bufferevent_read(bufevt, buf, 1024);

    // logic process
    // 发送回包
    bufferevent_write(bufevt, "user user1\n", 11);
}
```

- 服户端网络框架如下所示：

```
// 创建 socket 监听本机端口
fd = socket(AF_INET, SOCK_STREAM, 0);
bind(fd, …);
listen(fd, 10);

// 初始化 base
base = event_base_new();
struct event evListen;
// 设置事件 ，注册回调
event_set(&evListen, fd, EV_READ|EV_PERSIST, onAccept, NULL);
// 设置为 base 事件
event_base_set(base, &evListen);
// 添加事件
event_add(&evListen, NULL);

// 事件循环
event_base_dispatch(base);
```

```
// 读事件回调函数
```

```
void onRead(int iCliFd, short iEvent, void *arg)
{
      len = recv();

      // do logic
}
// 连接请求事件回调函数
void onAccept(int iSvrFd, short iEvent, void *arg)
{
    fd = accept();
    // do logic
}
```

3.5. 本章小结

本章主要介绍了本文所开发系统中涉及的基本技术，主要包括分布系统的特点，并给出了一种适合于实时网络游戏的分布式网络游戏架构。同时还重点介绍了共享内存模块在分布式网络游戏架构中的应用以及它的实现方式，并对共享内存的使用进行了详细的描述，共享内存为游戏服务器高可用性起着不可替代的作用。详细介绍了网络协议相关的 ProtocolBuffers 库在网络游戏编程中的应用，以及基于此协议格式的登陆协议实现，它使用简单，数据描述文件小，解析速度快，同时减少了二义性，平台无关，生成了更容易在编程中使用的数据访问类，使得我们的编程效率大大提高。最后简要介绍了 libevent 网络引擎的功能，简要介绍了 bufferevent 的 API 和相应的编程方法，实现了基于 libevent 网络库的客户端和服务器端编程。

第4章. 实时网络游戏的主要问题与技术

4.1. 引言

本章对实时网络游戏的概念进行了阐述。比较实时网络游戏和其他网络游戏的区别。通过实时网络游戏的特点，本文对几类主要的问题进行了归类：时间同步问题，网络延迟问题，移动同步问题，带宽消耗问题，并且对于其中的时间同步问题、带宽消耗问题给出了具体的技术实现。而对于网络延迟问题，移动同步问题我们认为这个是实时网络游戏的关键问题，将会在下一章进行详细实现。

4.2. 实时网络游戏的定义

实时网络游戏英文名：Real-Time Game，此类网络游戏对网络延迟的要求比较高，在一定的时间内必须系统必须做出响应，同时对网络包的时序要求非常严格，系统必须按照规定的时序来处理，同时统一同步给游戏客户端[6]。

在这类游戏中，因为这类游戏的流畅度往往都决定于延迟的大小，所以延迟问题的处理是一个大大的瓶颈。在游戏过程中，如果出现延迟的大幅度抖动，我们也可以看到游戏中的我们所控制人物的跳动，拉扯等现象。

由此看到，我们必须通过各式各样的算法和方案来改善网络延迟对游戏体验的冲击，特别是实时网络游戏，相比其他同类产品，一款对延迟问题处理得好的实时游戏，能够大大提高其的市场竞争力。

4.3. 实时网络游戏的主要问题

实时网络游戏中最重要的一点就是必须保证每个玩家在屏幕上看到的东西是一致的、必须是同步的、玩家的操作是必须有实时性的、服务器的判定必须是公平的。然而，现有的网络环境中，不可避免的存在着延迟、带宽等等问题。同时在网络世界中，为了保证所有玩家展现的一致性，那么所有玩家的时间也必须有一致性。

4.3.1. 时间同步问题

我们可以认为，时间同步，是同步算法的基础。对不同客户端中的时间同步问题，我们可以采取 NTP(Network Time Protocol)的来解决。

NTP 是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源（如石英钟，GPS 等等)做同步化，它可以提供高精准度的时间校正（LAN 上与标准间差小于 1 毫秒，WAN 上几十毫秒），且可介由加密确认的方式来防止恶毒的协议攻击。

目前，在通常环境下，NTP 提供的时间精度在 WAN 上为几十毫秒级别，而在 LAN 上则为亚毫秒级别。而在专用的时间服务器上，精度则更加高。

NTP 以客户机和服务器的方式进行通信。每次通信共计两个包。客户端发送一个请求数据包，服务器接收后回送一个应答数据包。两个数据包都带有时间戳。SNTP 根据这两个数据包带的时间戳确定时间误差，并通过一系列算法来消除网络传输的不确定性的影响。

因此，应用此服务于网络游戏中的服务器和客户端时间。这样可以有一个统一的仿真时间，以便以后可以有效的服务与游戏的同步算法。

- NTP 协议在游戏客户端中的应用：

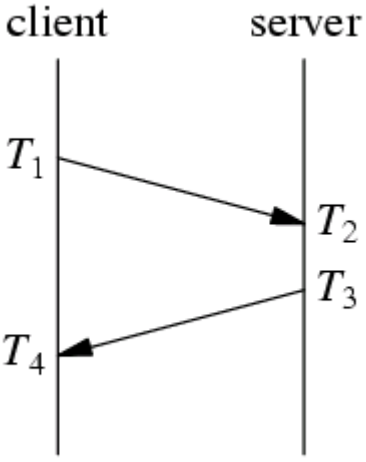

图 4-1 NTP 协议在游戏中的实现

Fig 4-1 NTP 协议在游戏中的实现

NTP 协议在游戏中的实现（如图 4-1）：客户端向游戏服务器中的 NTP 服务器发送请求数据包，NTP 服务器回应一个数据包。

分别记录客户端发送请求包时的时刻t_1和接收到回应包的时刻t_4，服务器回应的数据包内包含了服务器接收到请求包的时刻t_2和

服务器发送回应包的时刻t_3。

$t_4 - t_1$表示整个消息传递过程所需要的时间；

$t_3 - t_2$表示消息传递过程在服务器停留的时间；

$(t_4 - t_1) - (t_3 - t_2)$则是来回路上消耗的时间，在这里我们认为来回传输所用的时间一样，那么，单程的时间为：

$$t_s = \frac{((t_4 - t_1) - (t_3 - t_2))}{2}$$

假定客户端相对于服务器的时间误差是t_d，则有下列等式：

$$t_2 = t_1 + t_d + t_s$$

$$t_4 = t_3 - t_d + t_s$$

则：

$$t_d = ((t_2 - t_1) + (t_3 - t_4)) / 2$$

然后根据差值t_d重新设置时间即可。

- 实现伪码如下：

```
time_t t1,t2,t3,t4,td;   // 分别对应上文中所提到的时间

time(&t1);   // 获取 t1 的时间
send(t1);   // 发送协议到服务器

// 服务器在 t2 收到协议，并在 t3 发给客户端

time(&t4);   // 客户端在 t4 时间收到协议

td = ( (t2-t1)+(t3-t4) )/2;
if (td <= 0)
{
    // 表示本地时间比服务器时间快
}
else
```

```
{
    // 表示本地时间比服务器时间慢
}
setTime(td)          // 重设本地时间
```

4.3.2. 网络延迟问题

网络延迟是数据包在传输介质中传输所用的时间。在游戏中，数据包从游戏客户端中发出，到达服务武器端，服务器端给出响应，并传达给客户端的这一个过程所消耗的时间，我们可以简单的认为，这个耗时就游戏中的网络延迟。对于网络游戏来说，延迟是一个非常重要的指标。怎么样使网络延迟的波动对我们的游戏所产生的影响最小就成为了我们的一个主要问题。

网络延迟首先带来的就是，物体位置的不同步，物体的位置在各个游戏客户端中表现得并不一样。现在，我们可以通过一系列的算法来改善网络延迟对游戏中物体位置不同所带来的影响。目前，比较成熟的算法有航位推测（Dead Reckoning）技术。Dead Reckoning 起源于 17 世纪航海，指由已知的定点以罗盘及航速推算出目前所在位置的方法。将此方法运用于网络游戏中可以减少网络延迟波动对游戏产生的抖动影响。该算法在处理游戏数据时，如果数据包还没有到达，则采用预测的方法来处理。该算法允许客户端不受网络数据包延迟的影响，能够无中断的运行。而对于 Dead Recknoing 中的详细技术实现将在本文后续章节说明。

- 网络延迟的计算：

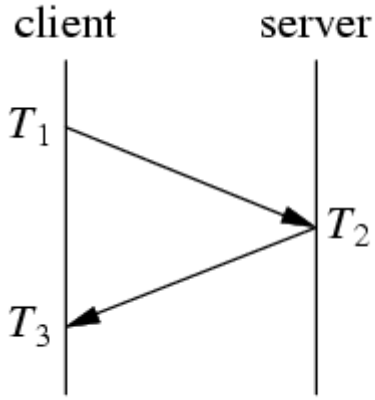

Figure 4-2 network lag calculate

图 4-2 网络延迟的计算

那么网络延迟是怎么来计算的呢，如图 4-2 所示，我们通过此种方式来

计算网络延迟：

首先，构造一条协议，ttl_protocol，此协议中包含一个字段，字段取值为客户端本地时间毫秒数，我们认为此时时间为T_1；

然后客户端把此协议通过 UDP，或者 TCP 协议发送至服务器的接收程序，注意：服务器接收程序对此协议不做任何操作，立即直接返回此包给客户端，此时客户端收到此协议的时间为T_3。则我们可以计算出，此时的网络延迟如下：

$$lag = \frac{T_3 - T_1}{2}$$

- 我们用伪码实现如下：

```
typedef struct _ttl_protocol {
    int64 m_time;
}ttl_protocol;

客户端逻辑：
ttl_protocol ttl;
ttl.m_time = Now();                    // 取本地时间毫秒数
client.sendto(server.address, &ttl);   // 发送给服务器
…
client.recv(&ttl);                     // 客户端收到服务器的回包
now = Now();                           // 取本地时间毫秒数
int lag = (now – ttl.m_time) / 2       // 计算网络延迟

服务器端逻辑：
server.recv(&ttl);      // 服务器接收到客户端发过来的协议
server.send(&ttl);      // 服务器立即把原协议发回去
```

4.3.3. 移动同步问题

同步是网络游戏的基础，一个不同步的网络游戏是无法给玩家带来乐趣以及公平的感受的，同时同步问题也是现有网络游戏开发过程中的关键

问题之一。根据游戏的类型、游戏设计的世界观，游戏的复杂度等等，可以对同步设计的要求从低到高排一个序。最低的我们可以认为就是棋牌类休闲游戏，最高的则是实时网络游戏。由于网络数据在介质中的传输，网络延迟的存在是不可避免的，所以我们也必须认识到，各个客户端之间不同物体的绝对同步是不存在的，但是就好比早期的电影画面也只是以 24 帧/秒的速度播放着静止的图片一样，我们依然有很多种方式方法来“欺骗”我们的眼睛，减少延迟对游戏体验的影响，提高游戏画面在延迟情况下的表现。在后续章节中我们将详细介绍现阶段比较流行的两种物体移动同步机制：保守同步机制、乐观同步机制。

4.3.4. 游戏带宽问题

在网络中，带宽有限的，是一种固定限度的资源。而在实时网络游戏中，由于客户端总是以一种高频率，小数据的方式来同步数据。而且对于 TCP 协议来说，在每个数据包的包头，都会额外包含一个 TCP 包包首，更加会加剧高频率协议的带宽消耗。由此可看，对于游戏服务器来说，我们必须利用一些算法与机制来节约服务器的带宽的使用。

1. 采取二进制字节流的形式来传输数据，比如：对于一个 int 类型，采用二进制字节流传输，只需要 4 个字节，而如果作为文本传输，则需要传输数字的实际长度。
2. 对于静态数据，只发送一次或者不发送。例如，游戏中道具的属性，NPC 的属性，怪物的属性等等这些不会随着游戏内容而改变的数据。我们只需在最开始初始化时同步给客户端本地，在后续的游戏过程中就再也不用再同步了。
3. 采取数据压缩。例如，某个协议中可能发送了一个 int 类型的字节流过去，但是这个 int 的数据范围是 0 到 1000000。在内存中，这个 int 数据只占用了 17 位，而剩下的 15 位久白白浪费掉了，于是可以指定将该整数为 17 个 bit 位进行发送。比如说，我们采用 Huffman 编码的方式来压缩数据，则可以拥有 20%-50% 的压缩率，大大减少的数据量。
4. 只更新那些被改变的数据。只发送与客户端相关的信息。由于客户端存在“视野范围”，而服务器中拥有所有的“视野”。所以说，我们分别对不同的客户端同步不同的视野范围内数据。例如，在 游戏世界中，玩家 A 与另外一个对象 B，这个对

象 B 在某个阻挡之后，根据物理学定律是玩家 A 是看不见 B 的，就没有不需要向此客户端同步这个对象的数据。再假设，玩家人物的状态，有位置，血量，方向等等，这个时候玩家移动了一下，其他状态并未改变，所以此时服务器只需同步移动信息，而并不需要同步血量等等的一些信息。因此我们的协议设计中，提供了一种机制，可以独立的，只更新某个状态的机制。

5. 协议消息区分优先级。在游戏中有很多时候，并不需要同步当前玩家身边的所有信息，比如身边其他玩家的详细状态，而只需要知道一个简短的信息如名字，位置等。只有在这两名玩家人物做出交互的时候才同步详细信息。
6. 对于可以通过固定算法或者公式来得到的数据，我们服务器端则只需要把基础数据传输给客户端就可以了，客户端服务器依据相同的算法公式计算出最后的结果。比如传递随机种子，在客户端产生和服务器端相一致的随机数字序列。

4.4. 本章小结

本章主要给出了实时网络游戏中常见的问题，以及常用的设计策略。具体说明了实时网络游戏中的时间同步，网络延迟，移动同步，游戏带宽等一系列问题，并对其中的时间同步问题的解决方案给出了详细的实现，采用 NTP 协议，我们保证客户端毫秒级的同步时间。对于网络延迟的计算，采用一种简单的发包回包算法来实现。对于游戏中的带宽问题，采取二进制流、Huffman 压缩、优先级策略更新、复杂数据客户端、服务器端共同计算等方案来达到一个节省带宽的目的。而对于本章中提及的延迟问题的解决方案，航位推测技术，以及移动同步的乐观同步技术和保守同步技术则放到下一章详细的进行描述与技术实现。

第5章. 实时网络游戏中的关键算法设计与实现

5.1. 引言

本章主要论述了在实时网络游戏中所用到的一些关键技术。主要包括：

航位推测技术在实时网络游戏中的应用，该技术主要包括两大类：航位预测技术和数据平滑处理技术。本文详细描述了该技术，并简述了相应的实现算法。

实时网络游戏中的同步技术也是本文阐述的重点，比较了保守同步算法和乐观同步算法，并实现了乐观同步算法在游戏客户端与服务器之间的算法应用。

为了保证实时网络游戏客户端与服务器数据的一致性，延迟补偿也是很重要的算法之一，本章详细也对于延迟补偿技术在游戏中的应用进行了简单实现。

5.2. 航位推测算法

航位推测算法最早是由美国国防部提出来的用来解决军事仿真中的延迟问题，它是一种位置、方向预测方案。最开始该算法应用于航海和航空，本质上航位推测算法（DeadRecknoing）是根据物体前一刻的状态位置来预测下一时刻的位置，它现在也应用于现在的网络游戏中。因为大多数游戏中物体和任务的动作都可以认为是有规律和具有继承性的，在网络延迟的这段时间中，游戏中物体和任务的速度，加速度和方向改变微小，在这种情况下客户端对于物体的运动是可以进行预测的[7]。它可以用预测的方法来隐藏网络延迟带来的滞后效应。当采用航位推测算法（DeadRecknoing）算法时不需要每一次都预测下一时刻的位置，只有当物体的实际位置和方向与预测的超过一定的阈值时才需要更新位置，这样来达到降低延迟对物体运动的影响。即使在延迟的这段时间，客户端的物体继续按照以前的运动轨迹进行运动，直到有新的服务器更新数据到达，同时根据移动的偏差大小是否超过设定好的阈值来决定是否向服务器同步更新位置。数据表明，这种预测轨迹运动很大程度都是有效的，这样就弥补了网络带宽的不足，提到了网络传输的性能，同时隐藏了延迟对客户端所

造成的影响。

在此算法具体设计实现中，我们需要注意的是：每一个实体在游戏世界中都有一个自己的内部模型，我们称之为 ghost 物体。ghost 物体在接收到服务器发来的数据包后，用此数据包推测自己的位置。当实际的位置和方向与预定义的位置和方向超过阈值时，客户端要与服务器来同步自己的最新信息来进行修正。同时服务器也将该客户端的更新通知给其他客户端。该算法利用简单的物体物理运动特性来进行预测模拟，不仅提高了各个客户端在同一时刻的游戏状态同步，同时也减少了数据在网络中的传输。

在新的服务器更新信息到达后，客户端需要更新游戏状态为最新的状态，由于预测和实际信息可能会有误差这个时候更新面临两种选择，一种方式是在收到新的信息后直接拉扯到新信息的状态，同时还有一种更好的方式就是采取一些算法，平滑拉扯到真实的位置。在下文中将详细描述预测技术和平滑处理技术。

5.2.1. 航位预测技术

在上文中提到，客户端根据收到的服务器更新信息包进行运算，预测一下个状态的信息，这种预测算法中有 9 种标准算法，而其中有二种算法是最基本的，其他几种只是由于坐标系采用的不用，其中的二种算法公式如下：

- 公式 1：

$$P_{dest} = P_{orign} + v_0 t \qquad 5\text{-}5\text{-}1$$

- 公式 2：

$$P_{dest} = P_{orign} + v_0 t + \frac{1}{2} a t^2 \qquad 5\text{-}5\text{-}2$$

其实这两个算法，也就是基于基本的物理学公式，公式（5-1）描述了从时间t_0时候起，以恒定的速度V_0运动到了t_1时刻；公式(5-2)描述了从t_0时刻起，以V_0为初速度，以加速度 a 运动到t_1时刻；

以上算法主要是针对物体在游戏世界中的位置预测，如果预测的位置于后来的实际位置不一致怎么办呢？实际上，在游戏中还需要采用相对复杂的技术来进行进一步的处理。这就是下文所要讲到的平滑处理技术。

5.2.2. 平滑处理技术

在航位推测技术中，我们利用平滑技术来处理实际位置与预测位置不一致的情况，我们主要采用某种方式对实际位置与预测位置之间进行插值计算，从而获得平滑效果。不用的插值方法往往有不同的效果。

在讨论该算法之前，我们假设该算法应用于平面二维空间，每个点都由（x，y）组成，同时 x，y 方向上的也有两个速度分量。后文中我们把主要包含了物体的位置、速度信息的协议统一称为 PDU（Protocol Data Unit）。

- 直接拉扯（point-to-point）：

直接拉扯（图 5-1）的思想是当航位预测超过阈值时，直接对物体更新更新操作，将该物体的状态立即更新到新的位置上。此算法由于并没有 2 点之间的数据过渡，所以在客户端的表现上，我们将会看到物体出现一阵一阵的“卡顿”现象。

这种算法实现起来简单，对于实体的每一次移动都要发送一个更新包，对本来就很紧张的带宽来说是很大一个压力；同时在客户端的表现上，实体也是一跳一跳的不连续前进的，给人一种很不真实的视觉效果。因此除了一些早期的局域网游戏，现在来说其他的网络游戏一般不会使用这种算法。

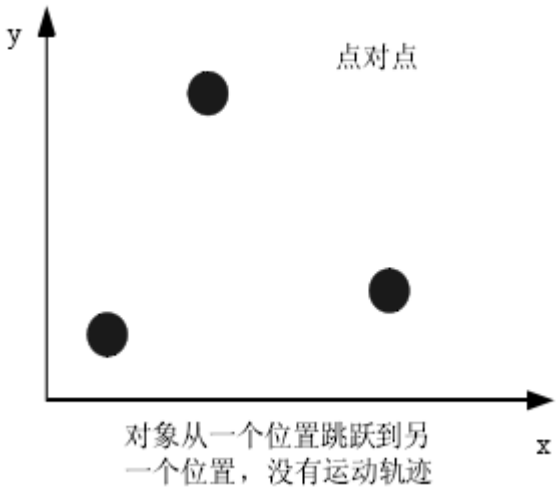

图 5-1 直线拉扯

Fig 5-1 point to point

- 线性插值（Linear）：

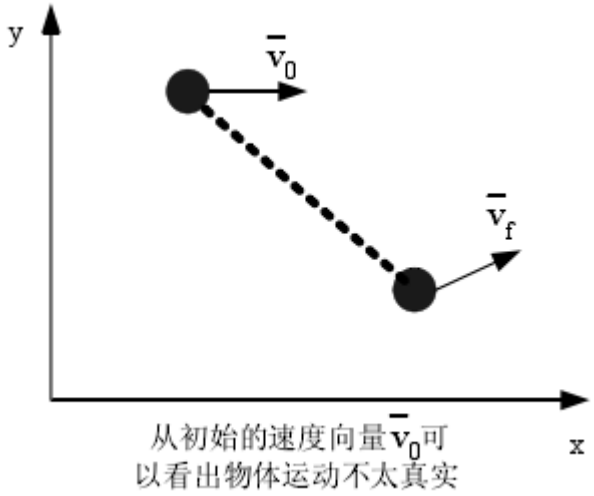

图 5-2 线性插值

Fig 5-2 Lineaar

线性插值（如图 5-2）即让物体 A 从他当前的的坐标直线运动到新的坐标点上去，同时采取上文所述的公式（5-2）：

$$P_{dest} = P_{origin} + v_0 t + \frac{1}{2}at^2$$

该算法的主要思想就是以当前位置为原点，依据物体直线运动规律，直线移动到新的位置。我们可以看出，该算法不需要过多的 CPU 运算，所以运算速度很快，但是模拟出来的轨迹非常僵硬。

该算法我们描述如下：

1. 物体在游戏进行中收到了一个新的 PDU 包，这个时候开始模拟运动
2. 首先计算运动耗时，从当前坐标P_{origin}到新的 PDU 包中描叙的坐标P_{target}，所需要的时间：

$$T_{dest} = \frac{P_{target} - P_{origin}}{V}$$

3. 依据运动耗时计算出在此时间之后的下一个目标点位置，利用新 PDU 包中所描叙的坐标信息与上一步计算出来的耗时T_{dest}，计算出最后时刻所应该达到的位置P_{new}：

$$P_{new} = PDU_v * T_{dest}$$

4. 计算修正速度，依据第 3 步计算出来的结果P_{new}和的第 2 步计算出来的时间T_{dest}，算出一个修正过的速度v_{new}：

$$v_{new} = \frac{P_{new} - P_{origin}}{T_{dest}}$$

5. 最后在游戏中让物体以速度v_{new}直线运动到P_{new}，并在到达目标点之后更新物体的速度，加速度等信息为新的 PDU 包中信息一致。

从现实效果上来看这种方式：相比直接拉扯算法，有效的杜绝了物体运动的“卡顿”感觉，变得稍微平滑了一些。但是物体的速度由于计算结果的差异而忽快忽慢，同时运动中的物体方向也会由于 PDU 数据的更新而有可能突然的改变，这时物体的移动也显得非常的诡异。

- 二次曲线行走（Quardratic）：

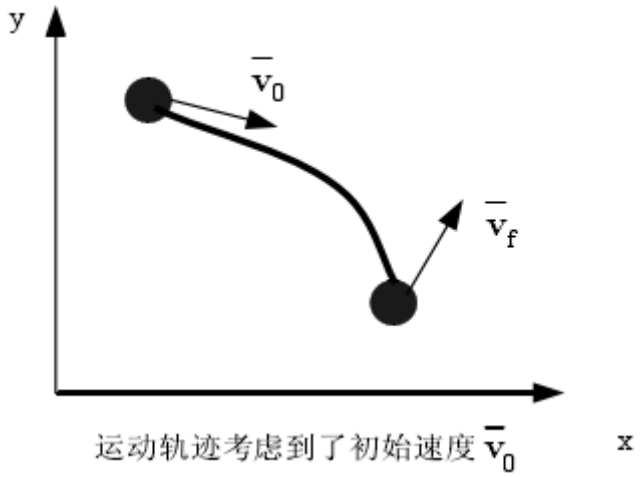

图 5-3 二次曲线行走

Fig 5-3 Quardratic

二次曲线行走（如图 5-3）的原理，同线性插值类似，就是在直线行走的模拟过程中，加入二次方程来计算一条曲线路径，让物体从当前点移动到目标点的过程是一条曲线，而不是一条直线。

具体的实现方法，就是在 Linear 方法的计算中，设定一个二次方程，在目标函数计算距离的时候根据设定的二次方程来计算，这样一来，可以使物体在玩家的屏幕上的移动变得更加有真实感。但是该方法的考虑也是不周全的，仅仅只考虑了从当前点到目标点的方向，而没有考虑新的 PDU 包中的方向描叙，那么开始模拟行走的时候，仍然是会出现比较生硬的拐角，那么下面所述的最终解决方案，将彻底解决这个问题。

- 立方体抖动（Cubic Splines）：

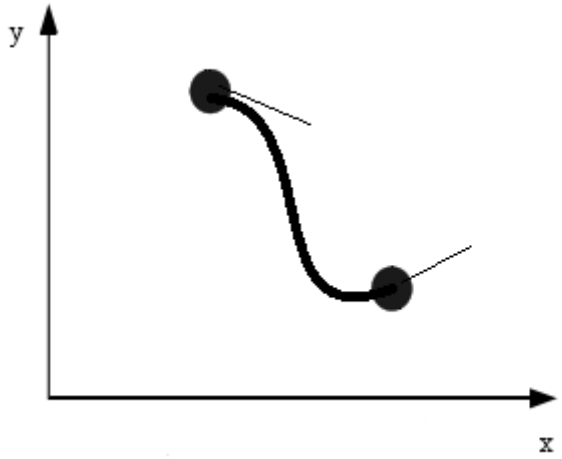

图 5-4 立方体抖动

Fig 5-4 Cubic Splines

立方体抖动（如图 5-4）也即是常用的三次方插值方法。我们需要 4 个坐标参数来进行计算：第一个参数是Pos_1，表示平滑处理的起始位置，第二个参数Pos_2，表示从起始位置模拟一秒以后的位置，第三个参数Pos_3，表示到达平滑处理重终点位置前一秒的位置，第四个参数Pos_4，表示平滑处理终点的位置。如下：

Pos_1：开始位置（即本地当前位置）

Pos_2：坐标 1 经过一定时间后的位置（速度为当前速度）

Pos_4：最终位置（即网络协议发送的最新位置加上一定的延迟时间后的位置）

Pos_3：坐标 4 反向移动一定时间后的位置（速度为网络最新速度）

运动物体的运动轨迹中的坐标点(x,y)由以下三次多项式计算得到：

$$x = A * t^3 + B * t^2 + C * t + D$$

$$y = E * t^3 + F * t^2 + G * t + H$$

其中 t 表示时间变量，我们设定 t 的取值范围为[0,1]，在Pos_1的时候 t=0，在Pos_4的时候 t=1，其中多项式中的系数取值如下：

$$A = x_3 - 3 * x_2 + 3 * x_1 - x_0$$

$$B = 3 * x_2 - 6 * x_1 + 3 * x_0$$

$$C = 3 * x_1 + 3 * x_0$$

$$D = x_0$$

$$E = x_3 - 3 * x_2 + 3 * x_1 - x_0$$

$$F = 3 * x_2 - 6 * x_1 + 3 * x_0$$

$$G = 3 * x_1 + 3 * x_0$$

$$H = x_0$$

其中x_0，x_1，x_2，x_3依次表示Pos_1，Pos_2，Pos_3，Pos_4中的 x 坐标。Y 坐标也同样如此表示。

现在，我们怎么应用于游戏中呢？我们来选择一种简单而有效的方法，如下：

1. 让物体按照自身的速度和加速度进行移动。
2. 当有一个新的数据包到达时，采取以上的方法来计算运动轨迹的下一个点。下面详细解释如何用 Cubic Splines 方法来计算坐标点：

Pos_1和Pos_2很容易计算，我们采用现在的坐标点和速度来计算，如下：

$$Pos_1 = P_{origin}$$

$$Pos_2 = P_{origin} + v_0 t$$

Pos_3和Pos_4的计算方式则没这么简单，我们认为经过时间 t 之后，这个物体会到达终点Pos_4。对于时间t选择，我们可以采取客户端的延迟时间的 3 到 5 倍数来选择，也即是说客户端延迟越大，我们预测的最终点Pos_4也就越远。

$$Pos_4 = Pos_{new} + (v_{new} t + \frac{1}{2} a t^2)$$

对于Pos_3来说，我们使用新的数据包来计算在时间t_{delta}之前的物体的途经点。

$$Pos_3 = Pos_4 - (v_{new} * t_{delta} + \frac{1}{2} a * t_{delta}^2)$$

3. 根据上面的方式计算出（x，y），使这个物体移动 t 秒。
4. 结束之后回到步骤 1[8]。

5.2.3. 算法的实现

如果航位推测技术应用到游戏的设计中，我们将结合推测技术与平滑处理基础一起来实现，该算法伪码描述如下：

```
t = 1;
bSmooth = false;
while (1)
{
    // 有新的数据到达，开始平滑处理
    if server.recv(&msg) && !bSmooth
    {
        bStartSmooth = true;
        // 选用上文所提到的二次方插值或立方体抖动插值计算出相应的(x,y)

    }
    // 开始平滑处理
    else if (bSmooth)
    {
        if t > 0
        {
            t -= t1;            // 减去逝去的时间
            // 根据平滑点进行图像渲染
        }
        // 渲染结束
        else
```

```
            {
                bSmooth = false；
                t = 1;
            }
        }
    }
```

5.3. 移动同步技术

5.3.1. 引言

在实时网络游戏中，服务器通常扮演者非常重要的角色，它主宰着整个游戏世界，并且负责游戏规则的判定，并处理着客户端传来的玩家信息。客户端和服务器以非常高的速率发送数据量很小的数据包进行通信，为此的目的就是让游戏世界保持高度的统一、一致性。于是在同步数据这一点上，同步性能的好坏玩玩就成了游戏性能好坏的关键因素之一。

简单粗暴的方式就是把每个玩家的 PDU 信息包都向其他玩家广播。然而，在网络中，存在带宽、网络延迟等等问题，因此针对带宽、网络延迟我们必须设定一个能够有效利用带宽，同时能够把网络延迟的影响减低到最小程度的算法。

5.3.2. 同步机制比较

在实时网络游戏中，为了保证所有客户端所显示的物体状态都一致，通过服务器来转发以及广播给每个客户端物体的状态，并且必须保证实时性。在利用同步算法时不可避免的要考虑到现有网络架构已经适用于现有网络环境的策略，现在我们比较最为基础的预测拉扯、验证同步两种方法。

- 预测拉扯

此算法的核心问题是计算出一个延迟误差值（lag），它指的是客户端与服务器通信的时候，客户端收到服务器通信数据的时间延迟。预测拉扯具体的算法步骤如下：

1. 客户端向服务器建立连接，并进行一次特殊的数据通信，这次数据通信服务器不做任何处理，只是简单的把原数据包回应过去。这个数据包在客户端发送时，客户端取本地时间带上，并记录到协议

中。然后等待服务器回包后，客户端把收到回包那一时刻的时间减去协议包中的时间，得出一个延迟 lag，并保存下来。以后客户端定时的通过同样的方式来修正这个 lag 值。

2. 服务器与客户端进行游戏逻辑的过程中。我们假设服务器在T_0时刻收到了客户端 A 中得玩家P_a发送的一条移动协议 PDU，此协议告诉服务器，玩家P_a要在T_0时刻从位置P_s移动到位置P_e。其中P_s是玩家P_a的起始位置，P_e是该玩家这次移动的终点位置，v 是玩家的移动速度。
3. 由于网络延迟的存在，服务器在T_1时刻收到了客户端 A 的玩家P_a发来的移动指令 PDU，根据前面所保存的延迟误差 lag 和客户端指令发出的时间T_0，计算得出$T_c = T_0 + lag$。然后服务器将收到的数据改为PDU_c，然后以广播的形式发送给所有相关的游戏客户端。
4. 假设该服务器中的另外一个客户端 B 在T_2时刻收到了来自服务器的这条指令PDU_c。我们计算出时间差，即此时从 A 发送数据到现在已经经过的时间：$T_d = T_2 - T_c$，若$T_d < (P_e - P_s)/v$，则$P_2 = P_s + v * T_d$，即在客户端 B 中，P_a在T_2时刻移动到点P_2，若不成立，则P_a在T_2时刻移动到P_e点。
5. 最后为了保证客户端所表现的运动自然不突兀，我们不直接拉扯，而是算出P_2偏后的一点P_3，用$(P_3 - P_2)/(T_3 - T_2)$来计算出V_{new}，然后让客户端 B 中的P_a用速度V_{new}在时间T_3快速移动到P_3。[9]。

- 验证同步

验证同步我们从名字上可以看出，就是客户端的每条指令只有通过服务器的验证后才能执行，该同步算法的具体步骤如下：

1. 如上文所述，获取 lag 值。
2. 同上文第 2 步。
3. 服务器在T_1时刻收到了客户端 A 的玩家P_a发来的移动指令 PDU。服

务器收到该指令后在逻辑层进行验证，这时客户端中的玩家并不会直接移动。服务器依据延迟 lag，对每个游戏客户端生成不同的消息时间，然后开始广播这条同步消息[10]。

验证同步的优点是：能够保证各个客户端的绝对同步，但是实时性不高，客户端的运行也不够流畅，给玩家带来的突兀和不连续的感觉非常不好。

5.3.3. 同步算法的实现

通过前面的比较，我们可以看出，预测拉扯机制更加有利于实时网络游戏的数据同步。目前来说，我们又将预测拉扯机制同步算法分为两大类：保守同步算法（Conservative Sunchronization）和乐观同步算法（Optimistic Synchronization）。保守同步算法在保证数据是安全、正确时才进行处理；而乐观同步算法则是在处理数据前不做任何检测工作，而是乐观的认为数据都是正确的，直到后来发现了错误再进行修正等一系列操作来保证游戏世界中物体的一致性。

需要注意的事，所有同步算法中需要把逻辑帧和显示渲染帧分开。

5.3.3.1 保守同步算法实现

在网络游戏中，保守同步算法一般来说采取的主要算法则是：帧同步，我们也叫它为帧锁定同步（Lockstep）算法。顾名思义，就是每一帧都需要同步，这类同步算法应用于早期的 RTS 游戏，如《星际》，还有一些早期的格斗游戏，如《拳皇》等等。这种同步方式能够很好的保障游戏数据的正确性。

Lockstep 算法是一种步进和同步机制，目的是为了让所有客户端只需要同步操作，这样运行的结果也就自然而然一致了。Lockstep 的关键在于同步性和确定性，一个固定的输入集合必定能产生确定的固定输出。客户端之间各自的输入消息，并通过帧锁定计算让，每个客户端之间保证每一帧的一致性。

Lockstep 算法的具体思想如下：所有客户端从同一个游戏状态开始，其游戏时间分为若干周期。在每一个周期里面，每台计算机都在自己的游戏世界里接受一个动作以前进一步；在此周期的结尾，每个客户端都会告诉其他所有的客户端玩家究竟做了一些什么动作；这样，所有的玩家都知道其他玩家在做什么，因此他们就可以共同进行到下一个游戏周期，从而

保持了游戏世界的同步 [11]。

举例来说，此算法流程如下（如图 5-5）：

1. 客户端定时（比如每隔 n 帧）把客户端所收集到的鼠标以及键盘相应消息上传。
2. 定义第 n 帧为关键帧。
3. 服务器受到所有客户端的上传信息后再广播给所有客户端，我们称之为更新消息。
4. 客户端用服务器发来的更新消息来继续进行游戏
5. 如果客户端在进行到下一个关键帧时没有收到服务器发来的更新消息则等待。
6. 如果客户端进行到下一个关键帧时已经接收到服务器的更新消息，则将更新消息用于游戏，并继续运行下去[12]。
7. 游戏继续，如此循环

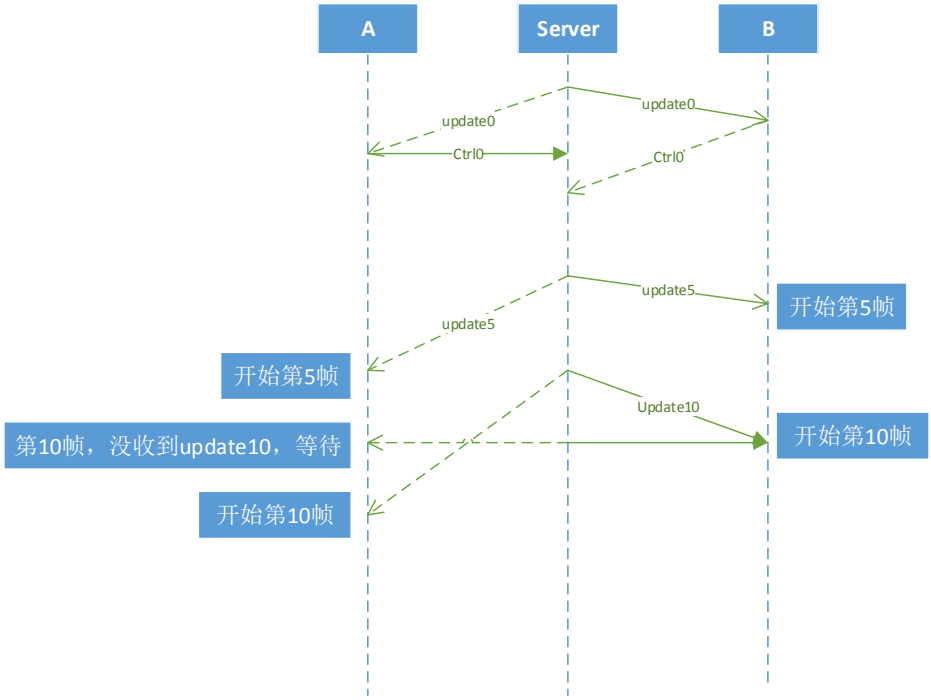

图 5-5 LockStep 同步技术

Fig 5-5 LockStep synchronization

客户端逻辑：

1. 判断当前帧 F 是否关键帧 K1：如果不是跳转（7）。

2. 如果是关键帧，则察看有没有 K1 的 UPDATE 数据，如果没有的话重复 2 等待。
3. 采集当前 K1 的输入作为 CTRL 数据与 K1 编号一起发送给服务器
4. 从 UPDATE K1 中得到下一个关键帧的号码 K2 以及到下一个关键帧之间的输入数据 I。
5. 从这个关键帧到下 一个关键帧 K2 之间的虚拟输入都用 I。
6. 令 K1 = K2。
7. 执行该帧逻辑
8. 跳转（1）

服务端逻辑：

1. 收集所有客户端本关键帧 K1 的 CTRL 数据（Ctrl-K）等待知道收集完成所有的 CTRL-K。
2. 根据所有 CTRL-K，计算下一个关键帧 K2 的 Update，计算再下一个关键帧的编号 K3。
3. 将 Update 发送给所有客户端
4. 令 K1 = K2
5. 跳转（1）

在游戏中，客户端等待关键帧更新消息的过程我们称之为“帧锁定”。同时，在服务器的运行过程中，服务器根据所有客户端的最大网络延迟，平滑计算下一个关键帧的编号，让延迟根据网络情况自动调整。此算法能够保证绝对的实时性。

这个协议的一个缺点就是每秒游戏的周期数是受到网络延迟的限制的。在每个游戏周期的结尾，这些客户端都必须至少收到从所有其它客户端发来的一个确认信息，这样才能继续得到下一个游戏周期的世界。这就确保了所有的游戏都是在处理相同的信息。而且必须时刻遵循这条规则。

5.3.3.2 乐观同步算法实现

乐观同步算法与保守同步算法相反，如上文所述，在保守同步算法中，客户端的表现往往受限于网络延迟最大的一个客户端，由于在关键帧时的等待，客户端的流畅性会有一定的折扣。而乐观同步算法并不会等待，它会继续的去进行游戏，直到更新包到达，然后再去检查数据的一致性，如有错误，则更正状态来保证一致性。典型的乐观同步算法有基于 DR

桶（Bucket）同步算法。

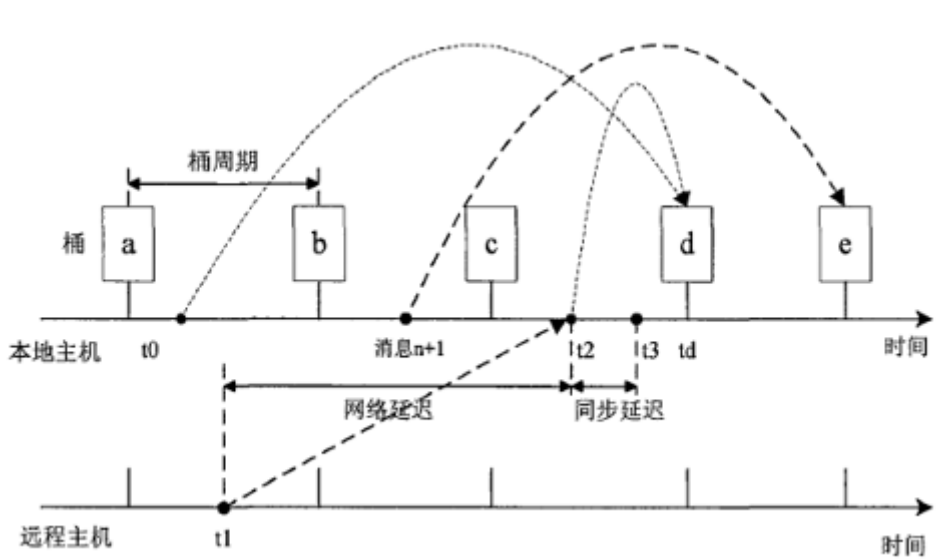

图 5-6 桶同步算法

Fig 5-6 Bucket synchronization

Bucket 同步算法（如图 5-6）是由 C.Diot and L.Gautier 于 1999 年提出来的乐观同步算法，最早在存分布式的 P2P 游戏 MiMaze 中实现。该算法的特点是：把时间分成固定的时间片，消息必须在固定的时间片内到达目的地，所有玩家在时间片借宿和执行桶中受到的游戏事件。同时该算法并不检测游戏的一致性，若当前桶中没有可用的消息，则根据前一个桶中的消息进行 DR 推测，由此得出最新的状态。我们可以看出，时间点的间隔决定游戏状态的速率，对于人们来说，25 帧/秒视觉上已经是很连贯了，于是 MiMaze 选择每隔 40ms 设置一个时间桶。

在此算法中，游戏的时间安排非常重要，保持基于时间的更新是保证各个客户端同步的基础，所以所有的客户端必须采取一种机制，保证有一种固定周期的更新方法。在此，我们设计一个固定长度的周期，我们称作（Timestep），同时，设计一种算法使得在整个游戏大循环中能够以固定的时间长度，周期的发送和处理数据。现在的解决办法是：使用一个时间累加器（Accumulator），每次更新的时候，将每次游戏更新的时间间隔（Delta Time）加到累加器（Accumulator）上，那么。当累计时间大于 Timestep 的时候就开始一次物理模拟，同时，将累计的时间减去 Timestep，该算法能够保证游戏完全以 Timestep 的时间步长进行游戏更新。

缺点也显而易见，时间片越大，响应和一致性就越低；时间片越小消息量又太大，丢弃的概率又更大。在大型的实时网络游戏中，并不能确定一个合适的时间片来满足所有玩家的需求。

- 算法伪码实现：

```
Bucket[1024]:            // 假设队列的最大容量为 1024

// 收到服务器中发来的数据包，其中时间=t

while (时间>队列中头部的时间，同时队列头尾并不一致)
    erase(head):              // 移除头部过时的操作

if ( 队列不为空，且与服务器上操作的时间相同 )
{
    if (队列中的位置和实际服务器的位置超过一定的阈值)
    {
        time.save;       // 保存服务器的时间
        position.save;   // 保存服务器的物理状态
        input.save       // 保存服务器的输入
        erase(head);     //移出队列首 move
        index = head;
        while(index != tail) // 重新模拟 t 时刻之后的所有过程
        {
            // 计算出时间差值
            deltaTime= Bucket[index].time 一 currentTime;
            entry.update;    // 通过差值和当前时间对物体模拟运动
            // 保存当前时间、当前状态、以及当前输入，继续进行下
一次模拟
            index++;
        }
    }
}
```

- Time Warp 算法

该算法最早应用于分布式虚拟环境。是一种典型的乐观同步算法，它的同步机制已被普遍认为是分布式系统中最重要的乐观同步协议。Time Warp 是通过检测和纠正不一致来确保同步。在 Time warp 中，各处理节点之间没有外在的同步，这些节点以不同的速率各自进行本地的仿真时钟，处理节点在每个事件执行前会对游戏状态做个快照(snapshot)，如果收到一个迟到的事件，并且此事件之前的事件已经被处理完毕，也就是说事件收

到的顺序是有错，就会发起回滚任务，快照和刚执行的事件之间发生的所有事件都要被撤销状态。回到早先的快照那里，此快照是没有按序进行处理的事件之前最新的一个。之后，这些被回滚的事件将按序重新执行。进行回滚操作通常包括两步，一是把状态恢复到迟到事件的时间戳之前的时刻；二是取消没有按序处理的事件所带来的一切影响。

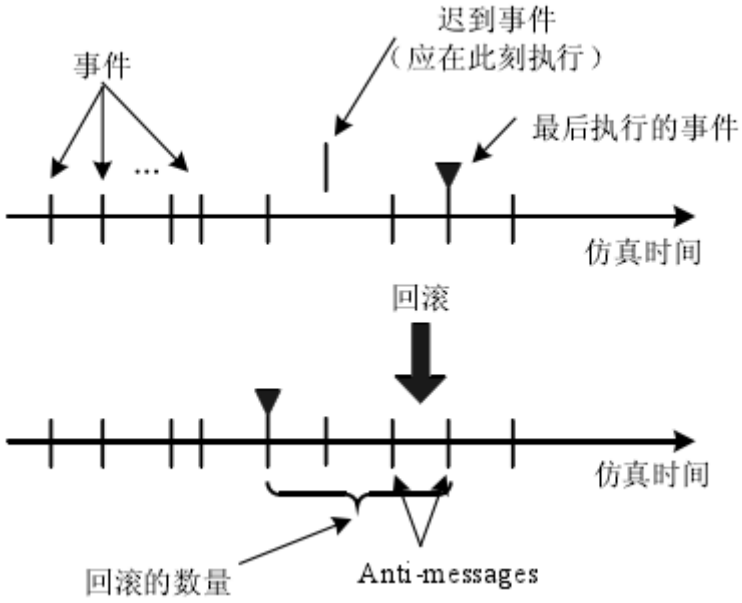

图 5-7 TimeWarp 算法

Fig 5-7 TimeWarp

可以看到（如图 5-7），该算法会引起大量的内存消耗，以及消耗很多 CPU 资源，但是该算法对于所有物体能够非常严格的保证游戏世界中物体状态的一致性。

5.4. 延迟补偿技术

5.4.1. 引言

在实时网络游戏中有一个分支：射击游戏。在此类型的游戏中，延迟补偿技术（Lag Compensation）尤为重要。

在单机射击游戏中，我们要判定子弹的命中非常容易，只需要做一个数学运算就可以判定该射击动作是否命中，而且射击命中的精度可以达到像素的级别，但是在网络游戏中，这种方法是不可取的。因为服务器不会相信客户端所做的判定，由于射击判定是会影响游戏公平性的问题，那么就必须在服务器上进行射击命中的判定。因此必须在游戏服务器上运行射

击命中的算法。

为什么需要延迟补偿？由于网络延迟的原因，使得服务器在处理过程并不能和客户端的数据保持一致，也就是不一致性的风险。好比玩家在 0.5 秒的时候向一个目标射击，射击的信息发送到服务器前，服务器仍然在不停的模拟整个游戏事件的事件处理，假设网络延迟 0.2 秒，则玩家的射击数据到达服务器的时间就是 0.7 秒，而在这个时候，在服务器中射击的对象可能已经运动到另外一个位置去了，这样就会造成，本来在客户端上已经打中的目标，实际上服务器上却并没有命中，客户端上的表现则为打中了，却并没产生相应的效果。这个时候，可以看出，并不能简单的由服务器来仲裁射击命令，还需要另外一个方式来弥补延迟所造成的错误。这种技术称为延迟补偿（Lag Compensation）[13]。

我们知道，服务器和客户端上都保持着玩家的历史状态信息。服务器需要依据延迟等参数算出命令执行时间，然后根据服务器上的历史信息，将服务器上的玩家状态恢复到 T 时刻，然后再进行相应的射击判断逻辑，这样我们的服务器就能够达到近似的精确性。

简单归纳一下，就是在服务器考虑了客户端的网络延迟，将服务器状态回滚到延迟前，再进行运算。所以这个策略有时候也被称作“rewind replay”。 延迟补偿技术除了用于改善因延迟带来的用户体验，更重要的也是为了防止客户端的作弊行为。

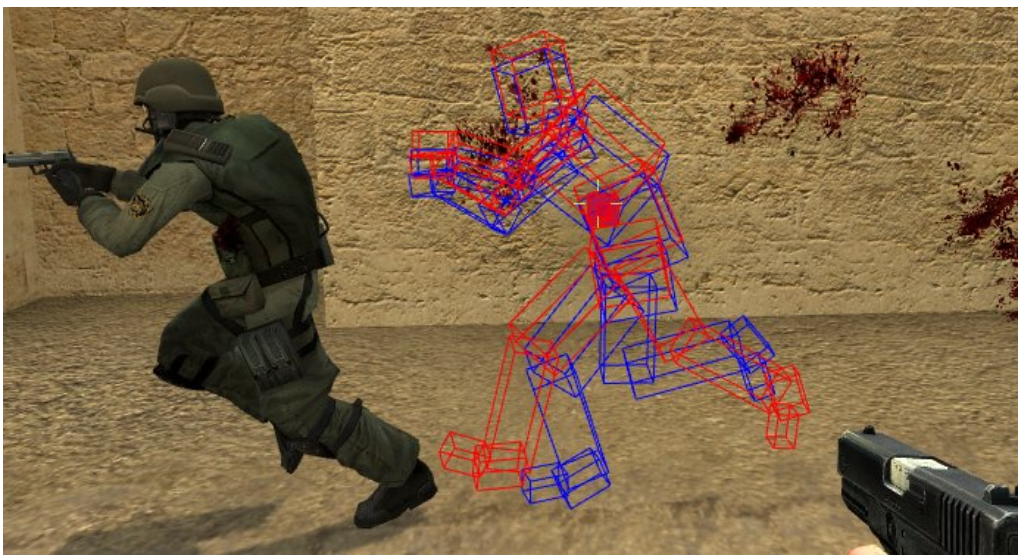

图 5-8 延迟补偿技术

Fig 5-8 Lag Compensation

如图5-8，其中人物的方块格分别表示客户端上100ms的目标位置、服

务器通过预测射击命令执行时间恢复的目标位置，也即是延迟补偿技术后的目标位置，可以看出，虽然有一定的误差，但是还是很大概率的还原了真实现场。

5.4.2. 算法实现

服务器端逻辑循环我们简单描述如下：

1. 接收从客户端发过来的玩家操作信息。
2. 收到消息后，根据玩家操作信息执行相应的逻辑运算。
3. 此时，采回滚到相应的模拟时间来模拟服务器控制的物体移动状态。
4. 对每一个连接的客户端，发送相应的物体、世界状态过去。

我们从上文可以看出，延迟补偿的关键步骤在于第二步、第三步，根据客户端的操作执行逻辑运算。对于此操作步骤，我们伪码描述如下：

```
while (1)
{
    server.recv(user_command); // 服务器接收到客户端发过来的射击命令消息
    if user_command.frame < server.frame // 如果用户帧 < 服务器帧
    {
        server_logic.roll_back(server.frame); // 服务器回滚到用户帧那一刻的状态
        logic_result = server.do_logic(user_command); // 服务器回滚后执行逻辑
        server.brocast(logic_result); // 服务器广播消息结果给所有客户端
        server_logic.forward(server.frame) // 服务器逻辑运算向前快进到服务器帧的那一刻去
    }
    else
    {
        logic_result = server.do_logic(user_command);
        server.brocast(logic_result);
    }
```

```
        server_logic.next_frame();              // 服务器执行下一帧逻辑
    }
```

5.5. 本章小结

本章主要描述并实现了实时网络游戏中的关键技术及其相关算法，其中包括：

- 航位推测技术可以用预测的方法来隐藏网络延迟带来的滞后效应。及其平滑处理方法的实现。同时比较了几种平滑处理技术的优缺点。
- 比较了保守同步同步算法和乐观同步算法的优缺点。并对保守同步算法中的帧锁定同步技术和乐观同步算法中的 Bucket 技术进行了实现，并给出其相应的代码。
- 最后，由于服务器在处理过程中存在着可能与客户端的数据不一致性的风险，本章实现了在实时网络设计游戏经常采用的延迟补偿方法。

第6章. 验证系统的设计与实现

6.1. 引言

该系统主要分为服务器端和客户端两个大部分，服务器端系统在 linux 下开发完成，系统版本号：Red Hat Enterprise Linux5.5，使用语言:C++。客户端则使用 ActionScript 3.0 语言，windows 环境下来进程编程开发。该系统服务器端采用了第 3 章所述架构，以及其中提到的网络库，协议库进行了基础的底层框架搭建。游戏客户端主要运用了第五章所述的算法：航位推测技术、桶同步算法实现了多个客户端之间的同步，网络延迟的隐藏。其中对于服务器架构我们可以采取编写自动测试机器人的方式来检验。而对比客户端的算法行为，只能通过人工测试来对比验证。

6.2. 服务器框架的验证测试

6.2.1. 测试环境

整套服务器系统部署到了 4 台 8 核刀片服务器和 2 台 16 核刀片服务器上，其中 2 台 16 核服务器用于运行压力测试工具。具体的服务器组硬件和软件配置如表 6-1 所示。

服务器型号	服务器规格	进程部署
HP ProLiant BL620c G7	CPU:8 核 内存：64GB 网卡：1000M	GatewayServer MasterServer
HP ProLiant BL620c G7	CPU:8 核 内存：64GB 网卡：1000M	GatewayServer GameServer
HP ProLiant BL620c G7	CPU:8 核 内存：64GB 网卡：1000M	GatewayServer GameServer
HP ProLiant BL620c G7	CPU:8 核 内存：64GB 网卡：1000M	LoginServer GatewayServer DBServer

IBM System x3650 M4	CPU:16 核 内存：16GB 网卡：1000M	Robot-1
IBM System x3650 M4	CPU:16 核 内存：16GB 网卡：1000M	Robot-2

表格 6-1 服务器部署配置

Table 6-2 Server deployment configuration

6.2.2. 压力测试

压力测试（Stress Testing）是指模拟巨大的工作负荷以查看应用程序在峰值使用情况下如何执行操作。扩展开来说，其一压力测试应该是较短时间的，其次是模拟巨大的工作负荷的，再次压力测试是要使应用程序的使用达到峰值。

本文采用的压力测试工具采用 C++语言编写，网络层模块使用与服务器一致的代码，逻辑层模拟多个客户端发送消息到服务器。压力测试工具主要模拟以下玩家的逻辑：

- 登录 Login 服务器
- 登录 Gameteway 服务器
- 进入游戏服务器。
- 发送位置同步协议
- 离开服务器，玩家下线。
- 等待一段时间后重新进行以上游戏逻辑

由于每台物理的负载设计要求是 5000 人，于是压力测试程序将会对 4 台物理服务器进行峰值情况下同时在线 2 万人的压力测试。在此监控服务器的稳定性，CPU 的负载，内存的占用。同时比较，在采用第四章游戏带宽的优化方法之后的网络带宽情况。

压力测试机器人不间断运行 48 小时。

6.2.3. 测试结果

6.2.3.1 CPU，内存占用情况

在压力测试机器人稳定运行 24 小说后，我们随机的查看部署有游戏服务器的一台机器的 cpu，内存情况。

```
top - 10:36:29 up 18:41,  2 users,  load average: 1.83, 0.94, 0.37
Tasks: 205 total,   2 running, 203 sleeping,   0 stopped,   0 zombie
Cpu0  :  1.7%us,  3.0%sy,  0.0%ni, 93.4%id,  0.0%wa,  0.0%hi,  2.0%si,  0.0%st
Cpu1  :  7.6%us, 13.9%sy,  0.0%ni, 60.9%id,  0.0%wa,  1.7%hi, 15.9%si,  0.0%st
Cpu2  :  5.6%us, 11.0%sy,  0.0%ni, 70.8%id,  0.0%wa,  1.7%hi, 11.0%si,  0.0%st
Cpu3  :  7.0%us, 13.9%sy,  0.0%ni, 66.9%id,  0.0%wa,  1.0%hi, 11.3%si,  0.0%st
Cpu4  : 10.3%us, 18.6%sy,  0.0%ni, 58.1%id,  0.0%wa,  1.0%hi, 12.0%si,  0.0%st
Cpu5  :  6.0%us, 11.0%sy,  0.0%ni, 71.0%id,  0.0%wa,  1.3%hi, 10.7%si,  0.0%st
Cpu6  :  6.9%us, 14.9%sy,  0.0%ni, 65.3%id,  0.0%wa,  1.0%hi, 11.9%si,  0.0%st
Cpu7  :  9.7%us, 17.7%sy,  0.0%ni, 59.3%id,  0.0%wa,  1.0%hi, 12.3%si,  0.0%st
Mem:  65996340k total,  2444132k used, 63552208k free,   196044k buffers
Swap: 65537156k total,        0k used, 65537156k free,  1733708k cached
```

图 6-1 cpu，内存占用

Fig 6-1 cpu，memory usage

从上图我们看出：未发现 cpu 有瓶颈、内存大量剩余、系统负载很低、没有磁盘 IO、网卡软中断未出现瓶颈。服务器各进程稳定，未有宕机现象。

6.2.3.2 网络带宽占用情况

我们通过 linux 系统自带工具 iptraf 在监控带宽的占用情况。随机查看其中一台服务器，在未采用带宽优化策略的情况下，网络带宽占用如下图（图 6-2）：

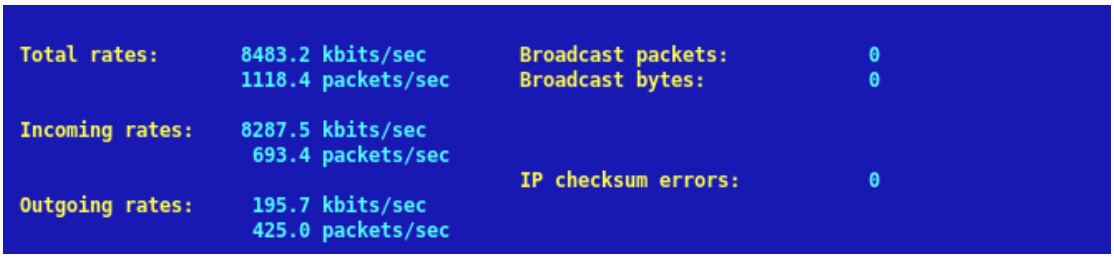

图 6-2 未采用带宽优化

Fig 6-2 Not using bandwidth optimization

采用第四章所述的带宽优化策略，带宽占用情况如下图（图 6-3）：

图 6-3 采用带宽优化

Fig 6-3 Using bandwidth optimization

可以看出，采用了带宽优化策略之后服务器的带宽减少了一倍之多。

6.2.4. 结果评估

对以上的测试结果分析如下：

因为采用了类分布式设计，所以各个服务器应用程序的 CPU 占用率基本按照线性增长，因此可以通过增加服务器的数量来应对玩家数量的增长。

结论：服务器框架的设计和实现基本满足预期目标，可以稳定地在比较长的时间内支持万人以上的同时在线玩家。同时文中的带宽优化策略能够有效的减少游戏服务器的带宽占用。

6.3. 客户端算法的验证测试

6.3.1. 测试环境

客户端使用 ActionScript 3.0 语言编写一个 2D 平面的物体运动模型。其中，玩家控制的物体和客户端收到服务器中的物体分别显示在一个屏幕内，如下图（图 6-4）所示：

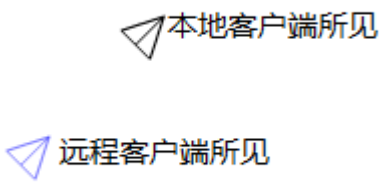

图 6-4 航位推测算法运动测试

Fig 6-4 DeadReckoning testing

同时在网络延迟的模拟方面，我们使用 clumsy 工具模拟网络延迟情况，clumsy 不需要额外设置，不需要修改你的程序的代码。系统级别的网络控制，可以适用于命令行，图形界面等任何 Windows 应用程序。不仅仅只支持 HTTP，任何 TCP, UDP 的网络连接都可以被处理。支持动态开启，你的程序可以一直运行，而 clumsy 可以随时开启和关闭。实时调节各种参数，详细控制网络情况。

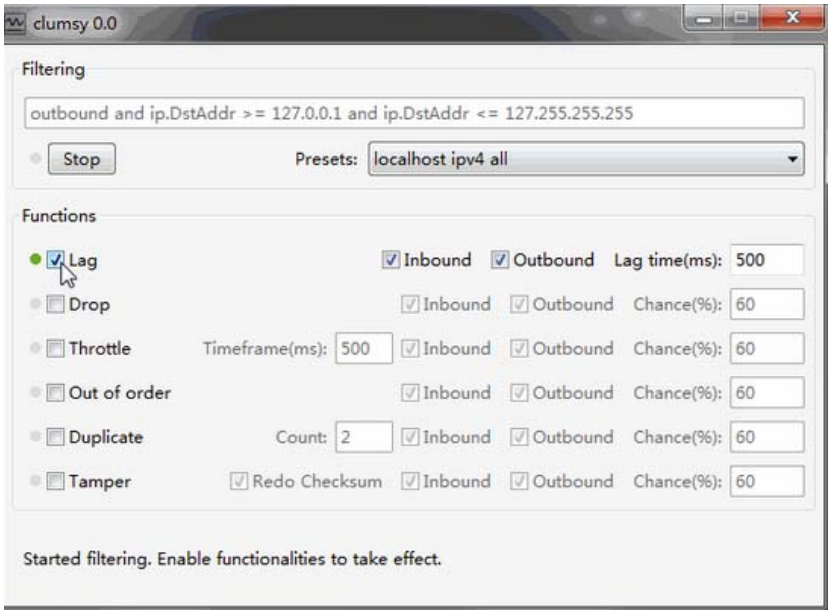

图 6-5 clumsy 模拟网络延迟

Fig 6-5 Use clumsy Simulation network lag

我们使用 clumsy（如图 6-5）把延迟设在 500ms。

6.3.2. 测试内容

人工测试客户端的航位推测算法，并比较其不同的平滑技术在客户端中的表现结果。同时模拟服务器的延迟让其在 500ms 之内。

6.3.3. 测试结果

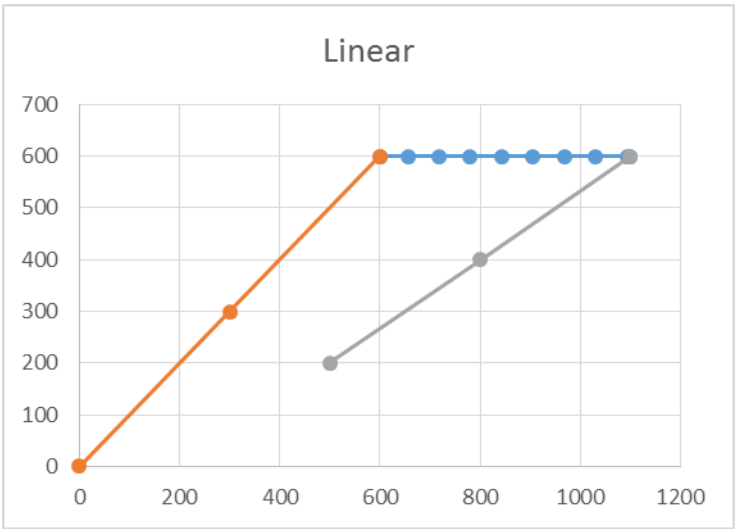

图 6-6 Linear 模拟

Fig 6-6 Linear simulation

从图 6-6 我们可以看出 Linear 的模拟完全是直线状态，而且没有任何方

向性的改变。

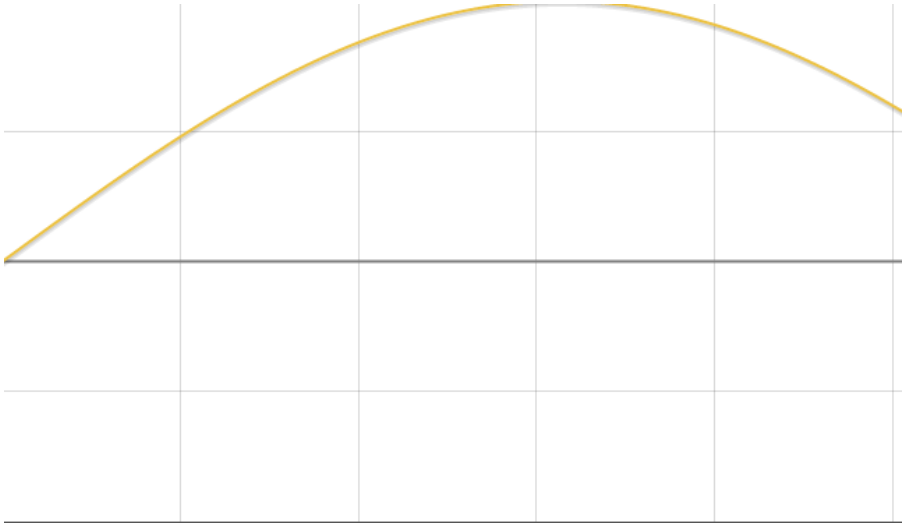

图 6-7 二次插值

Figure 6-7 quadratic

从图 6-7 我们可以看出二次插值的模拟比线性插值好很多，但是对于方向性的改变有点突然。

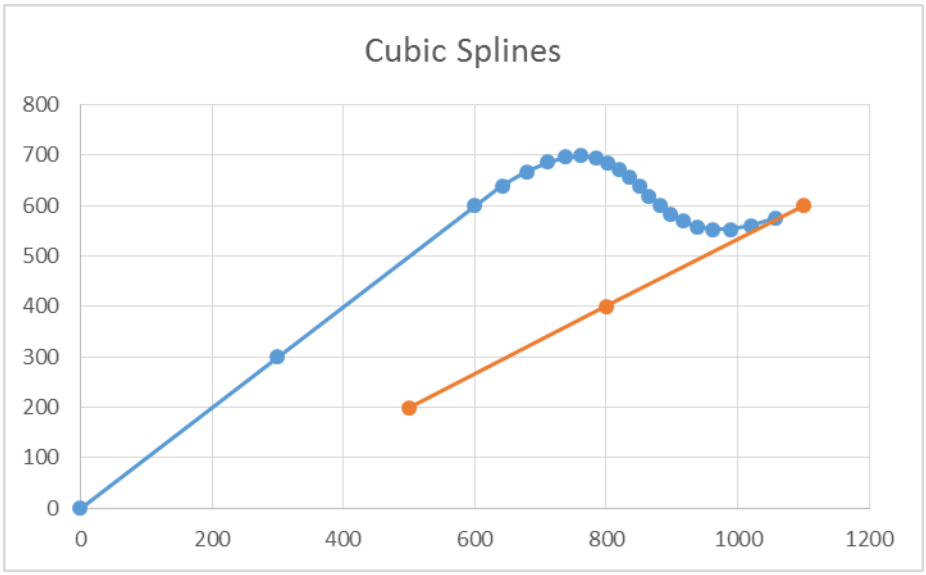

图 6-8 立方体抖动模拟

Fig 6-8 The Cube simulation

最后我们从图 6-8 看到，立方体抖动插值无论从平滑度，还是对于实际物体的方向性来说都有很好效果。

6.3.4. 结果评估

采用立方体抖动平滑算法在以上几种算法中给用户带来的体验是最平滑的，也是最好的。

6.4. 本章小结

本章阐述了服务器组的压力测试过程，包括测试环境、压力测试工具以及服务器组的压力测试结果和评估，验证了本文提出的服务器框架的可行性，同时通过 iptraf 工具监控网络带宽流量也验证了本文提出的带宽优化策略的可操作性。最后对于客户端算法也进行了人工测试，测试结果表明，采用立方体抖动平滑算是情况下的航位推测技术能够有效的减少网络延迟对于游戏用户体验的影

结 论

本文对网络游戏进行了分类比较，比较实时网络游戏与非实时网络游戏的差异：非实时网络游戏主要是以棋牌为主的休闲游戏，而实时网络游戏主要是以 RTS、FPS 为代表的即时性游戏。同时对网络游戏中的基础设计实现进行了详细描述，包括：分布式架构在游戏服务器架构中的实现，protocolbuffers 和 libevent 库在游戏编程中的基础编程方法。

由于网络延迟是在网络传输中不可避免的因素，解决好延迟问题对实时网络游戏的影响能够让游戏产品在激烈的市场上获得更有利的竞争地位。本文介绍了解决这一问题的航位推测技术的原理与平滑算法，该算法主要通过预测和平滑处理来解决网络延迟在实时网络游戏中所产生的不流畅性影响。对于该算法在网络游戏中的编程中的应用给出了具体的实现，比较了平滑算法，得出结论利用二次方插值或者立方体抖动插值在解决航位推测算法中的抖动问题上有良好的表现。

对于移动同步问题，本文比较了两类有代表性的同步机制和同步算法，并给出了 Lock step 保守同步算法的具体实现和 Bucket 乐观同步算法的具体实现，并结合实时网络游戏的特点，得出结论，并将 Bucket 算法具体的应用到了实际的项目中。而对于由于网络延迟引起的服务器与客户端数据不一致性问题，本文对于常用的延迟补偿技术给出了具体实现，该技术能够有效提高在实时网络游戏中 FPS 游戏的射击命中准确性。

尽管本文在实时网络游戏的网络关键技术进行了一系列的描述和编程实现，但由于实时网络游戏涉及到的范围广阔，本文实现的技术算法还是非常有限的，如人工智能，网络安全问题等等。本人认为在以下几个方面还有待进一步的深入挖掘:

- 安全问题

商业化的大型网络游戏最关心的安全问题，针对第一人称射击类游戏中的自动瞄准欺骗目前来说就是一种很难防范的欺骗技术，现在并没有非常完美的方法来对这种欺骗技术进行防范。

- 将人工智能引入网络游戏

由于时间的原因,本文并没有深入分析人工智能问题在网络环境下的应用，如何将非玩家角色和人工智能完美的融入本系统是有待今后进一步研究的内容。

- 游戏内容的动态自我发展

现在所有游戏都是基于游戏“剧本”的表现，游戏中的事件，任务，物体掉落等等都是约定好的，在未来经过神经网络技术的发展，相信这一切约定好的剧本也能够智能化，动态的自我发展。

最后，希望本文的工作能为实时网络游戏及其他相关领域技术的发展起到一定的借鉴作用。

参 考 文 献

[1] 云风. 游戏之旅[M]. 电子工业出版社, 2005-12.

[2] cnetnews. chinajoy10 周年[R/OL]. [2012-7].
http://www.cnetnews.com.cn/2012/0724/2103220.shtml.

[3] （美）特尼博姆. 分布式系统原理与范型[M]. 清华大学出版社, 2008-6.

[4] 德洛拉（Mark A.Deloura）. 游戏编程精粹 1[M]. 人民邮电出版社, 2004-10.

[5] Dickheiser M. 游戏编程精粹 6[M]. 人民邮电出版社, 2007-11.

[6] qdslp. 游戏数据存储设计概述[EB/OL]. [2013-03-10].
http://www.xuebuyuan.com/1560613.html.

[7] 梁白鸥, 陈雷霆, 蔡洪斌. Dead Reckoning 技术在网络游戏中的应用[J]. 计算机应用研究 , 2007(9).

[8] 信息产业部软件与集成电路促进中. 网络游戏客户端编程[M]. 电子工业出版社, 2007-8.

[9] Wendy Stahler、冯宝坤、曹英. 游戏编程中的数理应用 ：游戏编程中的数理应用[M]. 北京希望电子出版社, 2005-3.

[10] W.Richard Stevens、Bill Fenner、Andrew M. Rudoff. UNIX 网络编程 卷 1：套接字联网 API（第 3 版）[M]. 人民邮电出版社, 2010-6.

[11] 邹怡, 刘芳. 一种基于 P2P MMOG 的乐观同步算法[J]. 沈阳航空工业学院学报 , 2008 年 3 月 5 日(5).

[12] CALDWELL N. Defeating lag with cubic splines[S/OL]. [2000].
http://www.gamedev.net/reference/articles/article914.asp.

[13] Dalton. 帧锁定算法[EB/OL]. [2014-2-1]. http://www.zhust.com/index.php/2014/02/网络游戏的移动同步（四）帧锁定算法/.

[14] Mattern F. Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation[J]. Journal of Parallel and Distributed Computing, 1993(23).

[15] Cronin E, Filstrup B, Jamin S. An Efficient Synchronization Mechanism for Mirrored Game Architectures (extented version)[J]. Multimedia Tools and Applications, 2004(1).

[16] 王学慧, 张磊, 黄柯棣. 基于 PDES 乐观时间管理的周期时间桶算法研究[J]. 计算机工程与科学, 2005(8).

[17] 梁白鸥, 陈雷霆. 网络游戏中移动同步问题的解决方案[J]. 计算机应用研究 , 2007(5).

[18] CRONIN E, KURC A R, FILSTRUP B. An efficient synchronization mechanism for mirrored game architectures[J]. .Multimedia Tools and Applications, 2004(1).

[19] BERNIER Y W. Latency compensating methods in Client/Server ingame protocol design and optimization[J/OL]. [2001]. .

[20] 毕静. 一种适合网络游戏的多 NPC 协同运动策略[J]. 计算机工程 , 2011(7).

[21] 信息产业部软件与集成电路促进中心. 网络游戏服务器端编程[M]. 电子工业出版社, 2007-8.

[22] ARONSON J. Defeating lag with cubic splines[S/OL]. [1997].
http://www.gamasutra.com/features/19970919/aronson_01.htm.

[23] 陆松超. 多人在线角色扮演网络游戏的架构分析[J]. 电脑知识与技术, 2008 年 7 月 14 日(12).

[24] SMED J, KAUKORANTA T, HAKONEN H. A review on networking and multiplayer computer games[S/OL]. [2002]. http://www.tucs.fi/publications/attachment.php%3ffname=TR454.pdf.

[25] CRONIN E, FILSTRUP B, JAMIN S. Cheat-proofing dead reckoned multiplayer games(extended abstract)[S/OL]. [2003]. http://warriors.eecs.umich.edu/games/papers/adcog03-cheat.pdf.

[26] Rhalibi A E, Merabti M. Agents-Based Modeling for a Peer-to-Peer MMOG Architecture[J]. .ACM Computers in Entertainment, 2005(2).
[27] Claypool M, Claypool K. Latency and player actions inonline games[J]. Communications of the ACM, 2006(11).
[28] Cronin E, Filstrup B, Jamin S. An efficient synchronization mechanism for mirrored game architectures[J]. Multimedia Tools and Applications, 2004(3).
[29] Ferretti S. A synchronization protocol for supporting peerto-peer multiplayer online games in overlay networks[J]. ACM Special Interest Croup on Software Engineering, 2008(4).
[30] Mauve M, Vogel J, Hilt V. Local-lag and timewarp:providing consistency for replicated continuous applications[J]. IEEE Transactions on Multimedi, 2004(1).
[31] Cronin E, Filstrup B, Kurc A R. A distributed multiplayer game server system[D]. University of Michigan, 2001.
[32] Koster R. A Theory of Fun for Game Design[M]. Paraglyph Press , 2004-11-6.
[33] III R R. Game Design : Theory and Practice (2nd Edition) (Wordware Game Developer's Library)[M]. Wordware Publishing, Inc, 2001-02-25 .
[34] Ian Millington / John Funge. Artificial Intelligence for Games, Second Edition[M]. Morgan Kaufmann, 2009-08-20.
[35] Buckland M. Programming Game AI by Example[M]. Jones & Bartlett Publishers, 2004-09-30.
[36] Bergen I V D. Game Physics Pearls[M]. A K Peters Ltd., 2010-9-1.
[37] 艺术 游戏外挂攻防艺术 更新描述或封面 作者: 徐胜. 游戏外挂攻防艺术[M]. 电子工业出版社, 2013-2.
[38] Gregory J. Game Engine Architecture[M]. A K Peters, 2009-7-10.
[39] MBourg D. 游戏开发物理学[M]. 电子工业出版社, 2004-1.
[40] Nystrom R. Game Programming Patterns[M]. Genever Benning, 2014-11-2.
[41] 亚历山大. 大型多人在线游戏开发[M]. 人民邮电出版社, 2006-12.
[42] McShaffry M. 游戏编程全接触[M]. 人民邮电出版社 , 2006-1.
[43] 拉莫泽. 3D 游戏编程大师技巧[M]. 人民邮电出版社 , 2005-6.
[44] W.Richard Stevens / Stephen A.Rago. UNIX 环境高级编程[M]. 人民邮电出版社, 2006.
[45] 巴克兰德 (Mat Buckland). 游戏人工智能编程案例精粹[M]. 人民邮电出版社, 2008-6.
[46] Jefferson D. ACM Transactions on Programming Languages and Systems[J]. Virtual Time, 1985(3).
[47] SanchezCrespo D. 游戏核心算法编程内幕[M]. 中国环境科学出版社, 2004-1 .

致 谢

时光转瞬即逝，硕士学习的生活即将结束。在经历过寻找工作过程中的繁复与无奈之后，我依然眷恋书写论文过程中的那份宁静，那份深思。回顾研究生生活的步步历程，我感激那些曾经帮助我、激励我、引导过我的老师、同学和朋友。

本论文是我在攻读硕士期间的主要工作总结，首先要感谢我的导师肖创柏老师，感谢肖老师在这段期间给与我的悉心指导，为我答疑解惑。从毕业论文的开题、立意、定稿、到整体书写过程，肖老师都细心的指点，付出了大量心血。每一次指点都使我受益良多。肖老师也经常关心我们的生活，让我感念至深。祝我们的肖老师在学术事业上取得更大成绩。

自论文开始以来，经过了许多艰苦的学习和探索，非常感觉这几年来为我提供良好学习环境北京工业大学以及孜孜不倦的计算机学院的老师们。是他们的辛勤教学让我对计算机科学的前沿技术有了更深刻的了解，是他们的无私传授给了我的知识源泉，在这里衷心祝福各位老师和同学们的事业走向一个有一个的巅峰。感谢北京工业大学，我的母校，是母校搭建了我们学习成长的平台，让我们不断的学习新的知识，不断的充实完善自己。一路成长因为有你，我们的母校。

谨以此文献给我的父母，你们的养育之恩不能用语言表达.这十几年的学习生涯，求学路上是你们在做坚强后盾，你们是我不断努力进取的坚强动力。深深的向所有关爱过我的人真诚的道一声谢谢。感谢他们在我漫长学业的生涯中给与我的全力支持和全心照顾；感谢他们为我做的一切。

向百忙之中抽出宝贵时间来评阅本文的各位专家教授致以衷心的感谢！

最后,再一次衷心感谢在我硕士学习期间所有帮助和支持我的师长和朋友们,感谢他们为我做的一切，祝他们一生平安！幸福美满！