学校代码： 10246
学　　号：14210240028

复旦大学

硕 士 学 位 论 文

（学术学位）

基于 AST 的协同旅游路线规划算法研究

An AST Based Algorithm for Collaborative Itinerary Planning

院　　系：　计算机科学技术学院
专　　业：　计算机软件与理论
姓　　名：　杨 达 一
指 导 教 师：　卢 暾　副教授
完 成 日 期：　2017 年 2 月 26 日

指导小组成员名单

顾　宁　　教　授

张　亮　　教　授

卢　暾　　副教授

丁向华　　副教授

目　　录

摘　要

随着旅游产业和信息技术的发展，越来越多的人开始在网络上进行他们的旅游路线规划。在这类旅游路线规划的过程中，协作的重要性进一步加强。随着用户人数的上升和协作模型的复杂，支持越来越多的参与者进行协同旅游路线规划的目标变得越来越具有挑战性，因为用户数目的上升和复杂模型对于操作的响应速度和数据的一致性带来了更高的要求。地址空间转换（AST, Address Space Transformation）技术是一种支持实时协作的技术。除了确保共享数据一致，该技术能够掩盖网络延迟等带来的影响，从而给用户提供友好的快速协作交互体验。然而，协同旅游路线规划平台对于传统的地址空间转换技术中的数据模型以及通信协议都带来了新的问题和挑战，这些问题和挑战要求地址空间转换技术在通信协议，数据模型以及并发控制算法上做出改进和优化。

本文针对传统旅游规划系统在协同交互环境下所展现出来的新特点和新需求，基于现有的一致性维护技术研究成果，实现了协同旅游路规划系统 Clip，并且在系统的启示下优化了传统的地址空间转换算法，主要研究工作如下:

(1) 基于对传统旅游规划行为特点的分析，面向团体旅游路线规划的特点，首次提出了协同旅游路线规划系统所依赖的最基本的数据模型和操作模型。将协同旅游路线规划的模型和地址空间转换算法进行结合，对于并发操作所带来的冲突情况进行分析和分类，给出了相应的冲突解决方案。操作模型能够满足 CCI 模型，并将传统的一致性维护算法所支持的线性结构扩展成为“旅游规划图”结构，丰富了一致性维护算法所能支持的数据结构。

(2) 针对 Web 2.0 应用的特点，通过对通信机制的分析和操作关系的研究，提出了一种新的通信协议。该通信协议采用新的标量时间戳，避免了参与人数众多引起的性能开销，并且支持异步通信。异步的通信模式受网络延迟、参与人数变化以及并发操作数目波动的影响较小，并且传播时延也更加稳定，优异的性能适用于 Web2.0 应用中复杂的网络环境。此外，该通信协议也支持同步通信模式。

(3) 通过对“旅游路线图”的总结，发现了基于标识符定位目标节点的方法，从而在传统的地址空间转换算法基础上提出了改进算法。改进算法避免了传统地址空间转换算法中必不可少却花费大量时间的“回溯”过程，大大提高了算法的效率。设计并实现了协同旅游路线规划平台，验证和分析了协同旅游

规划平台中几种工作模式的性能情况。实验结果和理论分析表明，本文提出的改进地址空间转换算法相对与传统的算法具有更加优异的性能，能够满足协同旅游规划带来的动态性和实时性要求，对于其他基于地址空间转换算法的应用，也能带来的一定的启示作用。

关键词：计算机支持的协同工作；地址空间转换；协同交互应用；一致性维护；并发控制；互操作；旅游规划

Abstract

With the development of information technology, a growing number of users begin to use Internet as a platform to plan their itinerary collaboratively. During the online itinerary planning process, collaboration plays an increasingly significant role. As the number of travelers grows, it is challenging to support more and more participants to plan itineraries online with a sophisticated model collaboratively, due to the requirement of responsiveness and data consistency. Address space transformation (AST) is a kind of real-time collaboration method, which proves to the high performance in addressing the issue of interactive responsiveness and consistency maintenance. Network latency can be masked and friendly user experience is offered. However, collaborative itinerary planning system brings newer and bigger challenges to traditional AST algorithms. New data model, consistency maintenance algorithms and communication protocol are required to address these issues.

In order to meet the new requirements of collaborative itinerary planning systems, in this paper, a novel AST based system called Clip is designed to support collaborative itinerary planning. Inspired by Clip, traditional AST algorithm is improved. The research work of this paper is threefold and will be described as follows:

(1) This work analyzes the behavior of traditional itinerary planning and the feature of group travelling. To the best of our knowledge, it is the first work of applying consistency maintenance techenique to collaborative itinerary planning. Basic data model and operation model are presented. The conflicts caused by concurrent editing are analyzed and classified. The resolution of these conflicts is given too. New models meet the requirement of CCI model. Traditional linear data model is extended to “itinerary planning graph” model. New model enriches the data models that consistency maintenance algorithms can support.

(2) Aiming to the feature of Web 2.0 applications, we analyze the communication protocol and relationship among operations then a novel communication protocol is presented. New scalar-timestamp in this protocol is adopted to avoid the cost of massive paritcipants and supports asynchronous communication. Network latency, change in the number of paticipants and opreations have a less impact on

asynchronous communication protocol. Besides, propagation delay is more stable. Asynchronous communication protocol performs well in the complex situations of Web 2.0 applications. Moreover, operation detection method based on scalar-timestamp also can support synchronous communication.

(3) After summarizing the data model of "itinerary planning graph", a location method based on identifier is discovered. An improved address space transformation algorithm based on traditional one is proposed. Improved one avoids "retracing", which is a necessary step of traditional AST algorithms but consume too much time. Improved algorithm increases the efficiency of concurrent operations greatly. Collaborative itinerary planning system is designed and achieved. The performance of different working patterns is tested and analyzed. The result shows that improved AST algorithm and asynchronous communication protocol perform better than traditional one. Dynamic and real-time requirements from collaborative itinerary planning sytems are achieved. Inspirations to other appicaltion based on AST are presented too.

Key words： CSCW; Address Space Transformation; Interactive Collaboration Applications; Consistency Maintenance; Concurrency Control; Interoperation; Itinerary Planning

第一章　绪　　论

在过去的几年中，旅游产业成为了全球最为繁荣的产业之一，同时也刺激着区域的经济发展，社会就业，文化和环境等方方面面。旅游成为大众生活中尤为关键的一部分。而在旅游之前，人们通常会进行旅游路线的规划，从而更好的组织自己的行程。随着互联网技术的发展，这份需求被体现到了互联网应用当中。因此，关于旅游的互联网应用如雨后春笋般出现来满足这方面的需求。他们大多关注如何通过计算机技术来支持单个用户的旅游规划行为。后来，随着旅游团体的出现和众包技术的成熟，如何为团体提供实时协同的旅游路线规划技术成为了研究者关注的一个重点。本章将首先介绍本文主要内容的相关研究背景，然后介绍本文的主要工作，包括关键问题的定义以及解决方案，最后简述本文的组织结构。

1.1　研究背景

经济的蓬勃发展和社会的显著进步促使越来越多的人喜欢外出旅游。根据最近中国旅游研究中心发布的一项调查表明，2015 年中国有 1.2 亿的出境游客，他们花费了 1045 亿美元在旅游上，跟 2014 年比分别增长了 12%和 16.7%[1]。通常来说，为了追求更好的旅游体验和处理旅游当中可能遇到的情况，大部分旅游者会进行旅游路线的规划。比如通过行程表的方式去安排自己的游览顺序[2]。而计算机技术以及互联网技术的发展促成了很多基于互联网的旅游路线规划平台的出现。从此之后，基于互联网的规划平台成为了旅游爱好者首选的路线规划方式[3]。用户可以充分的利用互联网提供的资源，比如在线论坛，评分网站和推荐系统等在线资源去规划他们的行程[4]。现存的在线旅游规划平台，比如 TouristEye[5]，Travefy[6]和 WikiTravel[7]等，都是被设计来支持用户规划行程。在这样的系统中，用户可以根据自己的动机、需要和偏好去规划行程。此外，这些系统都不约而同的突出了旅游路线规划中的协作行为，允许用户通过共享和协作的方式进行路径的规划，并且因此取得了巨大的成功。由此也可以看出协作在旅游路线规划任务中的突出地位。然而，团体旅游规划系统的研究依然在进行当中，还有太多的地方需要人们的探索。

如何去规划一个最好的旅游路线一直是一个非常具有挑战性的学术问题。在数学研究中，过去被认为是一个最优化问题[8]，并且吸引了大量的研究者。因此，如今许多的现有路径规划方法是基于数学模型下的最优问题，比如旅行商问题等等。后来随着机器学习和数据挖掘等技术的出现和成熟，自动化路径推荐的方式成为了

旅游路线规划重要的一个组成部分，而这些自动化路径规划方法很少考虑群体智慧。而有研究表明，群体智慧在某些方面是能够和专家发挥出相同的效果，甚至超越专家[9,10]。如果能充分利用群体智慧去进行路径规划，不仅能够将大家的工作有机的结合在一起，也能够反映参与者个体的特点。因此，允许用户路径规划中进行交流和协作是非常重要的，比如利用行程表和地图作为共享文档进行规划。理想的旅游路线规划需要全体用户高度的参与、交互以及协作。更需要指出的是，用户应当被允许通过交互界面去贡献他们的意见和修改共享的内容，从而达到最终的一致结果。这种集思广益的“群体智慧”也被用在大多数现在的旅游规划系统当中，然而“群体智慧”的表现形式往往是文档和图片，路线规划中比较重要的信息隐藏在文字和图片之中，对于用户来说不够直观，也没有太深刻的影响，旅游路线图往往是最简单并且给用户留下最直观印象的模式，然而由于图形结构的复杂之处，很少有能够将群体智慧和旅游路线图结合的旅游路线规划系统，因为这种结合不单对于模型有新的要求，对于用户的操作以及系统的工作方式都有新的要求。

网络延迟是影响用户体验的关键因素之一。一个常见的问题是，客户端执行操作后需要等待服务器的响应，这需要花费一定的时间。更糟糕的是，有时候网络暂时不可用，等待半天返回的结果是失败。乐观复制算法是用来解决网络延迟的方法[11]。操作可以在非阻塞的工作方式下执行，这是解决响应问题的理想方式。然而这种组操作方式可能会导致操作之间的冲突从而导致副本的不一致[12]，这导致隐藏网络情况的应用变得非常困难和不稳定。此外，算法的效率不能太低，如果处理时间超过网络传输的时间则毫无意义。这些问题得不到解决或者解决的不好，都会导致用户体验的下降。因此，如何高效的维持数据副本之间一致的从而支持在线大规模协同成为了实时协同旅游路线规划中一个亟待解决的关键问题，这也是当前许多此类应用采用传统一致性方法无力解决的一个问题，迫切的需要一种新的方法和方式，去解决这个为协同旅游路线规划任务所重点关心的问题。

因此，实时一致性技术中的协同编辑算法的优良特性被相关研究者所注意，该算法能够处理好网络延迟的影响，并且能够满足副本结果一致的要求，不过对模型的要求比较高。如何利用协同编辑技术，从而实现旅游协同编辑，这些年吸引了一些研究者。协同编辑技术本身是一项存在很久的技术，大量的并发控制算法被提出来解决该技术中存在的一致性维护问题。然而，协同旅游路线规划依然给前人提出来算法带来了新的挑战，主要体现在旅游路线规划本身对于一致性维护技术提出了新的数据结构的要求，这也是一致性维护技术研究的重点问题之一。而如何去解决这些问题，不仅对于旅游规划问题本身带来一定的启示，对于具有相似的数据结构

或者网络结构的协同应用，也能带来一定的启示意义。该研究的意义和挑战，都是非常巨大的。

1.2 本文主要工作

1.2.1 问题发现

随着旅游行业的繁荣，参与旅游人数的上升，团体旅游的情况也越来越多。而随着人员增多，团体规模的扩大，人员的旅游意愿会变的更加的丰富。过去的旅游路线结构往往采用单一的旅游线路去完成规划，越来越丰富的团体旅游情况使得这种结构的限制变得越来越明显，因为它不再能够充分反映所有团体成员的旅游意愿。此外，具有旅游经验的人们参与旅游路线的规划对于旅游路线规划有着很大的帮助。在这两种需求的驱动下，研究人员开始将协同技术与新型旅游路线规划技术结合起来，进而满足人们日益增长和丰富的旅游路线规划需求。在这些研究者当中，Zhang 率先做出了相应的探索，指出旅游路线规划任务有一个重要的组成部分叫做“旅游路线的组合”[10]。参与者在共享文档之中添加和编辑的想法，从而优化旅游路线。为了解决这种多人协作所产生的效率问题和一致性问题，协同编辑技术被引入旅游路线规划系统中，从而使得旅游与协同有机结合，给用户带来更好的用户体验和更加灵活的操作，Zhang 的工作开启了此类旅游协作的先河，然而对于协同编辑技术没有过多的探索，使用的依旧是传统的一致性技术，系统本身和传统的协同编辑系统的区别不大，这种方式对于旅游路线规划这种场景来说不够直观，系统本身的性能也和传统的协同编辑系统一样，新的数据结构和更加稳定的协同编辑系统成为了如今协同旅游规划任务所关注的重点。

协同编辑系统（Collaborative Editing System，CES）已经出现了接近三十年，最早由 Ellis 等人在 1989 年代提出[12]。这是协同编辑领域最重要的几种算法之一，也是操作转换（Operation Trannsformation，OT）算法的第一篇文章。OT 算法的主要思想在于本地操作能够在副本上立刻执行，远程操作需要跟本地并发操作进行相应的操作转换，然后执行转换之后的远程操作。不同于传统的锁机制等悲观一致性算法，OT 算法的本地响应速度更快，并且能够根据用户的操作意图自动合并用户的并发操作，从而避免了操作合并所带来的其他开销[13]。经过多年的发展，OT 算法已经可以支持多种不同的数据结构[14-16]。然而，OT 算法依然面临着特定场景下可能会出现“难题”（puzzle），为了解决这些“难题”，OT 算法的结构和描述变得越来越

复杂[13]。状态向量（State Vector, SV）的大量使用使得算法的伸缩性收到了一定的影响[17]。

随后，可交换复制数据类型（Commutative Replicated Data Type，CRDT）算法在一致性维护领域的影响越来越大。CRDT 以一种完全不同的思路去构建协同编辑系统。此算法不需要对远程操作进行转换，因此不需要保存本地操作的历史。CRDT 算法通常通过给所有操作对象分配一个全局标识符，操作保存相关的标识符信息来组织共享文档的机构，使得并发操作的执行不依赖因果关系的确定，通过和全局标识符和全序关系来确定操作对象在副本当中的位置，从而确保副本之间的一致性得到维护，该算法的特点是更加适用于大规模 p2p 协同编辑系统[18]。然而，该类型算法的数据结构设计的困难度较高，导致支持的数据类型较少。此外，如何合理的分配全局标识符等挑战也制约着该项技术的发展和应用。

地址空间转换技术（Address Space Transformation）不同于以上两种一致性维护方式，该算法需要依赖状态向量从而对操作的地址空间进行转换，然后操作才能够通过全序关系正确的执行[19]。它判断因果关系的方法和操作转换方法类似，而确定操作对象的顺序的方法和 CRDT 有异曲同工之处。因此，该算法能够避免操作转换算法的复杂性以及各种“难题”（puzzle）[13]。此外，该算法通过“标记回溯”（Mark and Retrace）和“范围扫描”（Range Scan）的思想确定操作的目标位置，而无须依赖全局标识符。充分结合了 OT 和 CRDT 算法的优点。尽管地址空间转换技术已经探索了不同背景下的应用[20,21]，但是依然有许多新的数据结构等待着人们去使用地址空间转换算法。过去，地址空间转换算法还从未应用到协同旅游环境中，针对在线协同旅游系统中的特点，比如 web 服务通常需要集中式服务器而非 p2p 结构，需要支持人员的动态变化，用户必须支持高响应甚至能够忽略算法和网络传输所造成的性能下降以及维护旅游路线意愿等。种种均对传统的地址空间转换算法带来的全新的挑战，这也是本文所重点关注和着手解决的问题。最关键的是，新的模型究竟能不能对地址空间转换算法本身带来新的启示，从而提高一致性维护算法本身的性能？这些都是这篇文章研究的重点。

1.2.2 问题解决

针对协同旅游参与用户人数较多，而且协同旅游的人员时间和空间上分散的特点。协同旅游应用需要放到互联网环境下才能更好的满足用户的需求。然而，传统的协同编辑技术面向的几乎都是 P2P 架构下的，不依赖中心服务器，需要向量时间戳或者设计良好的标识符来表示操作或者操作对象之间的关系[12,19,22-24]。然而，他们

因为各自存在的局限，比如向量时间戳不适应用户的动态变化，标识符的分配对于算法的要求较高等因素，无法满足 web2.0 下互联网服务的新特点。因此，后来有相关研究者提出了针对互联网服务下的面向客户端-服务器架构的一致性维护算法 Jupiter 和 TIPS[25,26]，采用集中式服务器对用户的操作进行转发和处理。这种架构往往能够大幅度简化传统一致性算法的复杂度，并且能够适应人员动态变化的特点，服务的可伸缩性较高。然而他们面向的均为操作转换算法，对于其它一致性维护算法来说难于使用。

因此，为了让地址空间转换算法能够处理集中式服务器架构下应用的一致性维护问题，本文提出了新的通信协议。本地操作执行后，相关的原子操作将会被附加一个跟参与协同旅游任务的用户数目无关的时间戳作为客户端发送的消息，从而可以让系统由于人员动态变化对性能造成的影响下降。该消息通过服务器端主动推送给所有参与协同旅游任务的使用者，而不是等待用户主动请求未同步的消息。我们通过设计合适的通信协议，来确保主动推送的消息的发送和接受能够被客户端正确识别，并进行因果关系的判定。

在客户端，服务器端发送来的消息需要根据原子操作信息以及相关的时间戳进行地址空间的转换，从而使得具有特定关系的远程操作能够正确的在客户端执行，参与协同旅游规划的用户的操作意愿能够如实反映。无论是本地操作还是远程操作，操作的执行效率都需要非常的高，从而能够提高用户的体验。此外，数据模型需要和旅游路线规划的需求结合起来，传统的一致性维护算法已经能够正确维护的结构，如线性结构和树形结构等等[14-16,19]无法很好的满足这一需求，因此需要提出新的数据结构和原子操作。

1.3 本文的研究内容

本文主要解决在线协同旅游路线规划平台所需要面对数据结构、算法性能、以及消息通信协议上存在的问题。通过结合协同旅游路线规划任务的特点，对地址空间转换算法进行数据结构、消息通信机制以及性能上的改进。提出了在线实时协同旅游路线规划平台的重点关注的问题以及解决方案。

本文的主要工作如下：

(1) 提出了适用于协同旅游路线规划平台的地址空间转换算法。通过在原有的地址空间转换算法的基础上，结合协同旅游路线规划任务的特点，对任务涉及的元素进行抽象和建模，提炼出简化的数据模型以及原子操作，并且将这些

模型和操作跟地址空间算法结合起来。新的算法能够有效的维护用户的操作意愿以及共享文档的副本一致性，满足协同旅游任务规划的需求。

(2) 提出了针对服务器消息推送的中心式网络结构的通信机制。该机制不依赖传统的向量时间戳，而是使用对于用户参与人数不敏感的标量时间戳，并且该时间戳在通信机制的控制下，支持同步和异步两种工作模式，可以加快消息的传播，提高实时性。

(3) 提出了无需回溯的改进地址空间转换算法并验证了正确性。通过改变地址空间的组织形式，新的组织形式使得算法性能进一步的优化，加快了对于目标对象的查找和操作，从而大大提高了算法性能。

1.4 本文的组织结构

本文主要分为六个部分，本文的组织结构如下：

第一部分，绪论：阐述本文的研究背景和研究意义，说明旅游路线规划和一致性维护的研究现状以及本文的主要研究内容。

第二部分，相关原理和技术分析：介绍与本文研究内容相关的技术研究现状、适用场景和各自优势。

第三部分，协同旅游规划行为分析与建模：分析协同旅游规划的需求。针对协同旅游规划模型抽象出基本操作和数据模型，分析操作和模型之间存在的会造成互相影响的关系。

第四部分，消息通信机制：介绍了本文所采用的支持异步通信的算法，定义了操作关系，指出如何在通信机制中检测并发操作，通信算法的工作流程，阐述消息通信机制对于一致性维护的重要作用。

第五部分，算法改进和效率分析：在前面提出的模型的基础上，介绍了通用的地址空间转换算法优化方案，并且对该方案进行了正确性证明以及效率分析，然后给出了协同旅游规划平台 Clip 的模型以及 AST 引擎的结构。最后，设计了相应的性能测试实验，证明改进算法和优异性能以及工作算法的稳定性。

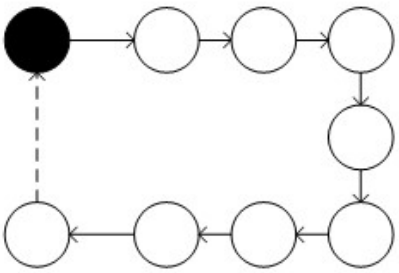
第六部分，总结和展望：总结本文研究工作的重点，指出本文对于旅游路线规划方法和一致性维护技术的贡献。在此基础上，为未来工作提供思路和方向。

第二章　相关原理和技术分析

2.1　旅游路线规划问题的发展与演变

相关研究表明 ，旅游的路径模式并非无章可循。路径规划问题最早被看做是“定向问题”（Orienteering Problem，OP）[8,27,28]。定向问题通常被认为是这类问题:有特定的起点和终点，如何按照一定的先后次序经过一系列其他地点，使得这段路程的总效用最高[29]。后来，相关研究者通过对大量用户的旅游模式进行分析，归纳出了线性路径模型的旅游模式特点[2]。线性路径模型主要指的是点到点模型，环形模型和复杂模型，其中最常见的是点到点模型和环形模型。点到点模型和原始的定向问题类似，指的是起点和终点不同，中间经过一系列其他点。环形模型是起点和终点相同，并且中间经过一系列其他点。复杂模型指的是除了二者之外的其他难以用简单的模型来概括的路径模型。大部分现有的旅游路线规划算法都是基于点到点和环形模型进行讨论和说明的。无法充分体现团体协同旅游路线规划的特点，后面的章节将会着重介绍这一点。

点到点模型



环形模型

图 2-1：基于线性模型的两种旅游路线规划图

2.1.1 传统旅游路线规划方法

旅游行程规划问题一直以来都是一个充满挑战的问题，并且进来也越来越引起相关研究者的关注。最开始旅游规划问题被认为是一个困难的非指定多项式（NP-hard）的寻找最佳旅游路线的优化问题[8]，或者在特定限制条件下的最优解问题[30,31]。此外，启发式的遗传算法[27]和人工神经网络算法[28]也被用来解决此类优化问题。后来，旅游路径规划的重点不再是一个单纯的计算问题，聚类算法[30]和推荐算法[32,33]等智能算法开始被用到传统的路径规划中，这类方法通常可以一定程度上对用户的数据进行分析，利用用户的一些个人行为，比如认知，行为体验和情感体验等[34]，或者旅游本身就有的特征，比如时间、花费、距离、天气、交通以及景点的人气等信息[35,36]。 从而比传统的方法能够更加多层次的反映用户的需求，并且对于用户将会选择的旅游路线进行一定程度上的预测[37,38]。

简而言之，许多现存的旅游路线规划技术是用自动化推荐技术去解决规划问题，大大的简化了用户的复杂行为。此外，传统的方法大部分针对的是单一用户，因此都是点到点模型或者环形模型。这种模型通常把每个 POI（用户兴趣点）作为模型的基本结构和操作对象，每个 POI 的出度和入度不超过 1。这意味着，对于参与旅游的团体来说，每个用户无法对规划的路线进行分裂，旅游路线的分裂意味着在团体旅游中，团体可能在某些特定的 POI 处下分裂成两个或多个小团体，分别独立进行一部分的线路，然后在特定 POI 进行路线合并。传统的旅游路线规划方式显然无法实现这种方式。自动化的方式由于提取的参数需要预先设定，依赖于算法的正确性而无法利用人的智慧。因此，无法动态的根据用户复杂意愿变化而变化。俗话说“众口难调”，传统的方法根本无法充分的反映所有用户的旅游意愿。

2.1.2 众包旅游路线规划方法

随着众包技术的出现与发展，众包技术的成功吸引了越来越多研究者的关注。作为一种能够充分利用人类智慧的协作方式，对于完全自动化的路线规划方式来说无疑是一种重要的补充。相关研究表明，一个经过精心选择的群体所产生的群体智慧能够媲美甚至打败专家的专业知识[9,10]。利用群体智慧来进行路线规划，不仅仅能够将群体智慧聚焦，而且也能反映组内的个人需求。因此，相关研究者已经开始将众包技术引入了旅游路线规划之中[10,39]。通过将众包的协作方式引入协同旅游路线规划之中，利用一些设计好的工作流程，来组织参与者的行为。使得团体用户能够充分发挥自己的聪明才智，表达自己的操作意愿。

然而，众包旅游路线规划方式的研究重点在于协作的组织形式以及工作流程，比如 to-do-list 能够使得群体用户聚焦在旅游需求上[10]，三轮工作流对于协同旅游路线规划的影响[39]，以及旅游经验的分享能够帮助用户对于景点和行程有着更加全面的认识[40]。并未从底层技术上进行改变，依然沿用现有的计算机技术进行协同路线规划，工作效率和用户体验没有深层次的改变。

2.2　数据一致性维护的方法

为了隐藏网络延迟带来的影响，提高用户访问和操作数据的效率，一般在用户的本地站点会保留一份数据的副本，作为多个用户之间的共享资源 [19]。当多个用户试图并发访问和编辑被共享资源时，如果不通过相应的算法对冲突操作进行检测和消除，最终会造成共享资源的数据不一致。此外，这类模型通常除了遵守一致性原则之外，为了便于用户的使用，操作之间还会维持因果关系和满足用户的操作意愿，也就是俗称的 CCI 模型（Convergence, Causality preservation and Intention preservation，收敛、因果保持和意愿保持）[41]。这意味着除了各个站点间副本最终一致之外，操作的因果顺序在所有站点一致，并且用户产生操作时的意愿，能够在实际执行操作的时候反映出来。

2.2.1　消息通信协议和操作关系检测机制

对于依赖时间戳来检测操作之间因果关系的 OT 和 AST 算法而言，通信协议以及操作关系检测算法是他们必须的一部分。最开始普遍使用基于向量的时间戳，适用于 P2P 的网络环境[12,19]。向量时间戳是一个长度为 N 的向量，每一维代表某一个站点当前执行到了第几个操作。每一个操作附加一个向量时间戳来表示操作产生时的状态信息。此类时间戳的好处在于实现简单便于理解，不依赖特定的中央服务器。然而对于用户数目敏感，一旦用户数目上升或者处于动态变化中，将极大的影响系统性能。因此，在用户数目处于动态变化并且存在大量用户参与协作的 web2.0 环境中，传统的向量时间戳及其相应的通信协议不再能够满足日益增长需求。因此，后面出现了集中式网络结构下的通信协议和操作关系检测机制，比如 Jupiter[25]，NICE[42]，Codoxword[43]，TIPS[26]以及 Hydra[20]。他们不再依赖对于用户数目敏感的向量时间戳，而是采用标量时间戳，也就是通过中心服务器序列化操作的方式来实现通信协议，从而产生了相应的操作关系检测机制。然而，Jupiter，Codoxword 以及 TIPS 他们都是针对操作转换算法特地设计的，并不具备较强的通用性和移植的可能

性。虽然 Hydra 的通信协议针对地址空间转换算法，但是他的场景针对移动环境下的实时评论，因此其他站点的操作需要移动端主动轮询拉取，而非服务端实时推送，会消耗更多的资源，性能受到影响。

除了依赖时间戳的传统通信协议之外，也存在一类完全不使用时间戳的一致性算法，如 WOOT[44]，TreeDoc[23]，RGA[45]以及 LOGOOT[46]等，此类算法通常被称之为 CRDT。这类算法通常是通过统一分配的标识符来定位操作的目标位置，完全避开了传统一致性算法中如果不判断操作之间的因果关系，就无法正确定位操作的目标位置的问题。由于完全不依赖时间戳，因此更加适用于 P2P 环境。由于该算法的稳定性以及效率几乎完全依赖于标识符的选取和处理，如何选取好的标识符以及相应的对标识符的处理算法是该类一致性算法是所重点关注的问题。

2.2.2 一致性算法支持的数据模型和操作模型

除了通信协议和操作关系检测机制，数据模型和操作模型也是一致性维护中关键的一部分。最初，一致性算法的出现为了是解决共享文档的并发编辑所造成的不一致问题[12]，这类共享文档通常都是简单的线性模型。一般来说，这种线性模型指的是在一维线性空间上存在着的连续的节点，每个节点通常代表一个具有实际意义的对象（比如字母，数字，符号等等），每一个节点可以通过相应的序号来访问，该类数据模型通常支持两种基本的原子操作，删除节点和添加节点。而此类模型，在很长一段时间都是各类一致性算法研究所关注的重点[12,19,23,44,45,47]。后来，随着协作形式的深入，更多的数据模型被相关研究者发现了，相应的一致性问题被很好的定义和解决，相应算法也被逐步提出，其中包括表格协作处理[48]，对于通用标记语言的支持[49]，3D 辅助设计[14]，XML 文档[21]等等。而 CRDT 的侧重点主要还是在线性模型或者转化之后的线性模型上进行探索，研究的侧重点在于提高相应的操作效率，支持的数据模型还不多。尽管在数据类型和操作类型的支持上，OT 和 AST 都进行了各自的探索并且已经取得了相应的成果。然而，协同旅游路线规划问题，本质上是在图结构中进行一致性维护的过程。依赖关系较为复杂，而过去的研究成果，并不能够直接的用在协同规划旅游路线图当中。在图上进行协同编辑，是一致性算法一直未能很好处理的领域之一。因为图是一个比较宽泛的概念，想将一致性算法运用到图结构当中，需要对图结构有清晰的理解和定义，并且和具体的旅游场景相结合，才能更好的解决这个问题。

2.3　本章小结

本章主要分两部分介绍了与本文研究相关的技术发展情况。第一部分主要介绍了旅游路线规划技术的发展，首先介绍了旅游路线规划问题的模型一般会抽象为相应路径模型，然后从传统的带权图的优化问题，发展为自动化路径推荐算法，包括聚类算法和推荐算法等数据发掘算法。后来，人们认识到优化问题和自动化路径规划的局限性，采用了基于众包技术的旅游路线规划方式，解决了自动化路径推荐算法所无法灵活处理的用户偏好等信息的问题。第二部分主要介绍了一致性技术支持的通信协议以及数据模型。通信协议主要内容为操作关系的检测机制，首先介绍了基于向量时间戳的通信协议和操作关系检测机制，这最为传统的机制由于对于用户的数目非常敏感，因此后来被基于中心服务器的标量时间戳通信协议取代，该协议对于用户的数目不敏感，因此非常适合存在大量用户的 web2.0 环境下。此外，存在一种完全不依赖时间戳的 CRDT 算法，该算法的标识符能够替代时间戳的作用，标识符的选取和处理是该算法研究的重点之一。最后介绍了现存的一致性算法所支持的数据模型和操作模型，尽管现在已经支持了多种模型，可是对于图结构模型的支持理论还在探索之中。

第三章　旅游路线规划行为分析与建模

不同于优化算法以及自动化旅游路线规算法的思路，本文所提出旅游路线规划算法研究的重点是以用户为中心，让一组用户能够协同编辑旅游路线的算法。本章在第一部分首先给出两个例子，分别是一个旅游规划任务，分配给一组有经验的用户去解决。以及一组共同旅游的用户，协同进行旅游路线的规划。从这两个例子分别给出群体旅游路线规划系统的必要性以及相应的需求，并针对需求提出相应的数据模型和操作模型。第二部分针对数据模型和操作模型，给出旅游路线规划协作平台的架构。并对工作方式和交互过程做出简要的说明。为了适应用户数目动态变化以及高响应的特点，采用集中式服务架构和本地保存副本的方式。

3.1　协同旅游路线规划需求研究

如果去一个陌生的地方旅游，我们通常会在网上查找相关的攻略和资料，也可能会通过向曾经去过该地或者了解该地的人打探情况。针对这样的需求，相关的门户网站顺势而生，比如 wikitravel，猫头鹰旅游，蚂蜂窝等等。用户在这些网站上撰写相应的旅游攻略和注意事项等等，吸引了大批用户。显然，如果这些有经验的用户能够参与到新人旅游路线的规划之中进行协助，他们丰富的经验和互动，比起单纯的文字和图片的帮助更大，如果他们能够参与到旅游规划的协作之中，满意的旅游路线规划会更加容易产生出来。然而如果存在多个用户对于旅游计划进行编辑时，无疑需要对他们的操作进行合并，如果完全采取手动的方式合并，响应速度太慢，而较差的用户体验，比如合并成本较高，合并耗时过长等等。无疑会导致有经验用户的沉默或者离开而失败。因此，对于协同旅游路线规划平台来说，高的操作响应速度非常重要。

此外，如果旅游的主体是一组用户时，每个人都会有自己的想法和旅游意愿。有时候他们可能会决定暂时“分道扬镳”。考虑这样的场景，有六名用户从北京来到上海旅游，其中有三名是复旦的校友，另外三名是上海交通大学的校友，他们来到浦东国际机场后，三名选择回复旦看望同学，另外三名选择回上海交通大学看望同学，最后他们在虹桥机场一起回北京。而在现实中，团体旅游往往在大方向上保持一致，而部分情况下产生差异，也是十分常见的现象。这种区别，也是我们的协同旅游规划平台所需要研究的特点之一。

传统的旅游路线规划方式有两个特点，第一，无论旅游的主体是单个用户还是一个团体，跟上文提到的旅游模式的线性模型类似，总是假设他们路线是按照一定的先后顺序经过一系列 POI（兴趣点）。意味着对于某一个时刻或者时间段，对应着唯一的 POI。第二，使用传统的协作方式来进行旅游路线规划的代价比较高，因为协作需要用户进行大量的沟通，多人试图对旅游路线规划进行编辑时，也会导致大量的操作合并所带来的额外时间花费，因此一般来说，即使是团体旅游的情况，实际进行规划的人数也不会很多。此外我们的研究注意到，对于实际的团体旅游行为中，可能会出现团体分裂的情况，在同一时刻或者同一时间段，团体分为两批或者多批去访问不同的 POI，这意味着传统的时间段和地点一一对应的关系被打破，如图时间和地点存在着一对多的关系。这对于一致性算法的强项线性模型提出了新的挑战，因为根据这样的原则，团体的实际旅游路线可能是一个有向图结构，图中会有一个 POI 到多个 POI 的情况，也会出现多个 POI 到一个 POI 的情况。针对这样的情况，以及高响应的要求，我们需要提出新的数据模型和操作模型，从而适应新的需求带来的挑战。

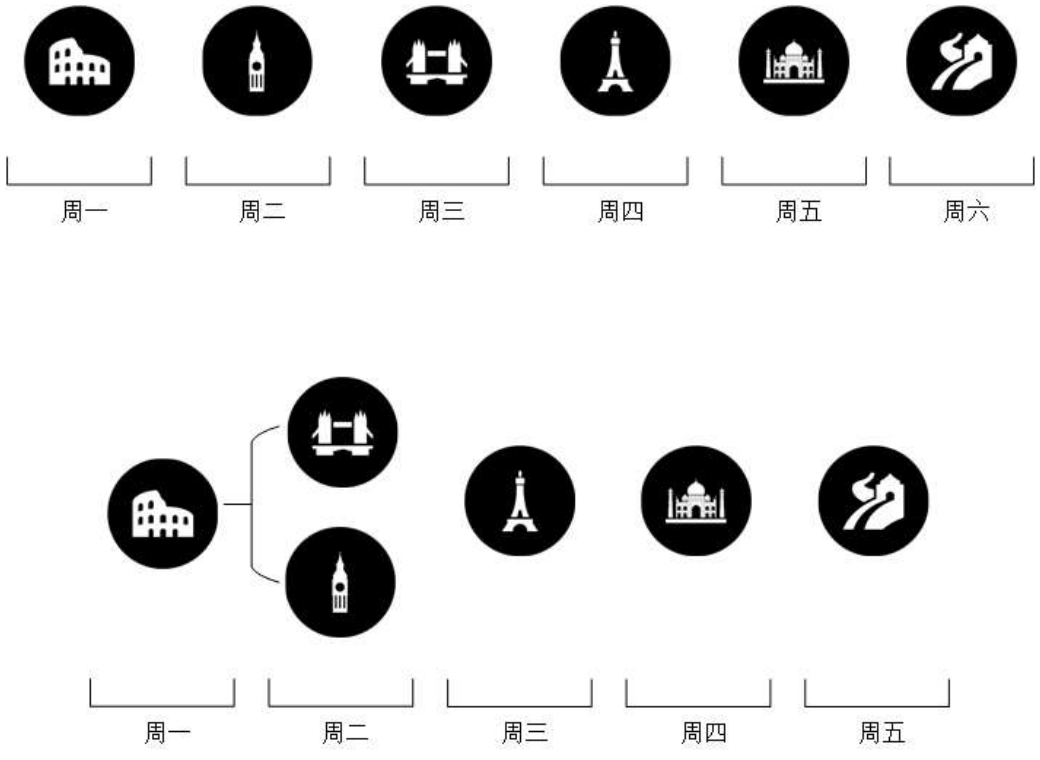


图 3-1：传统的时间跟空间的一对一关系和时间跟空间的一对多关系

3.2 数据模型和操作模型

因此，我们把协作形式聚焦于共享地图上的编辑，我们为用户在地图上设计了五种基本操作，添加 POI，删除 POI，通过有向线段连接两个 POI，删除两个 POI 之间的有向线段，以及对 POI 信息的更新。五种基本操作所组成的序列在地图上的结果抽象为旅游规划图（Itinerary Graph, IG）。严格来说，我们是这样定义 IG 的。IG（PS,ES）主要由两部分组成，PS（POI sequence）是一个 POI 序列，而 ES（edge sequence）是一个有向边的序列。他们的每一个单元都可以通过序列中位置信息进行访问，在我们这个系统中被称为索引（index）。具体来说，POI 和连接 POI 的有向边 Edge 分别是这样一个多元组。

- POI:<UIDoP, latlng, title, content> 其中 UIDoP 指的是 POI 的全局标志符，因为随着用户的操作以及一致性控制算法的影响，POI 的 index 会发生变化，因此 POI 和 index 并没有一一对应关系，而全局标识符和相应的 POI 存在一一对应的关系，关于全局标识符的具体作用将会 在下一章给出。Latlng 是 POI 的地理位置信息（经纬度），通过该信息将会确认该 POI 在地图上的位置。Title 指的是该 POI 的名称，content 是关于该 POI 的一些描述信息等，由用户给出。
- Edge<startPOIndex, endPOIndex, content> startPOIndex 指的是连接操作产生时出发 POI 的 index，而 endPOIndex 指的是目的 POI 的 index。 Content 指的是关于该有向边的描述信息，由用户给出。

基于这两个基本数据模型，我们可以给出相应的原子操作模型。

- **AppendPOI（latln, title, content）：**将一个 POI 依据经纬度信息放置在共享地图上，包括了标题信息和内容信息。实际上，这个节点会被放置在 POI 序列的最后。
- **DeletePOI (index)：**根据 index 信息删除相应的 POI。此外，所有依赖该 POI 的有向边均被移除。表示该节点因为某些原因不再可以被访问。
- **Connect (startPOIndex, endPOIndex, content)：**创建一个从 startPOIndex 所代表的 startPOI 到 endPOIndex 所代表的 endPOI 的箭头，表示 POI 之间的访问先后顺序。这个有向边将会被添加到边序列的最后。
- **Disconnect(index):** 删除 Index 所代表的有向边。意味着该访问顺序由于种种原因被取消。

- **UpdatePOI(index, content):** 根据 Index 的指定的 POI，修改对应 POI 的内容。意味着对于该 POI 的描述存在着误差。

直观上来说，着五个原子操作和数据模型可以生成上文所说的复杂的旅游路线图结构，帮助用户进行路线规划，其他复杂操作均可以看作是基于这五个原子操作组合形成的。

3.3　协同旅游路线规划的框架

为了便于更多的用户参与到协同旅游之中，协同旅游路线规划系统采用浏览器/服务器架构，如图 3-2 所示，由一个特定的中央服务器和任意数目的参与规划的用户组成，每个用户的本地均保存一份路线规划的副本，中央服务器负责管理和转发用户的操作。当一个用户开始参与协同旅游路线规划时候，首先从服务器上获取其他用户产生的历史操作，从而构筑本地副本，有了本地副本，用户的本地操作可以在副本上立刻执行而不依赖服务器的响应和网络状况。用户执行完毕后将自己的操作附送到服务器端，服务器会给每个操作分配一个序号然后将附加序号的操作信息

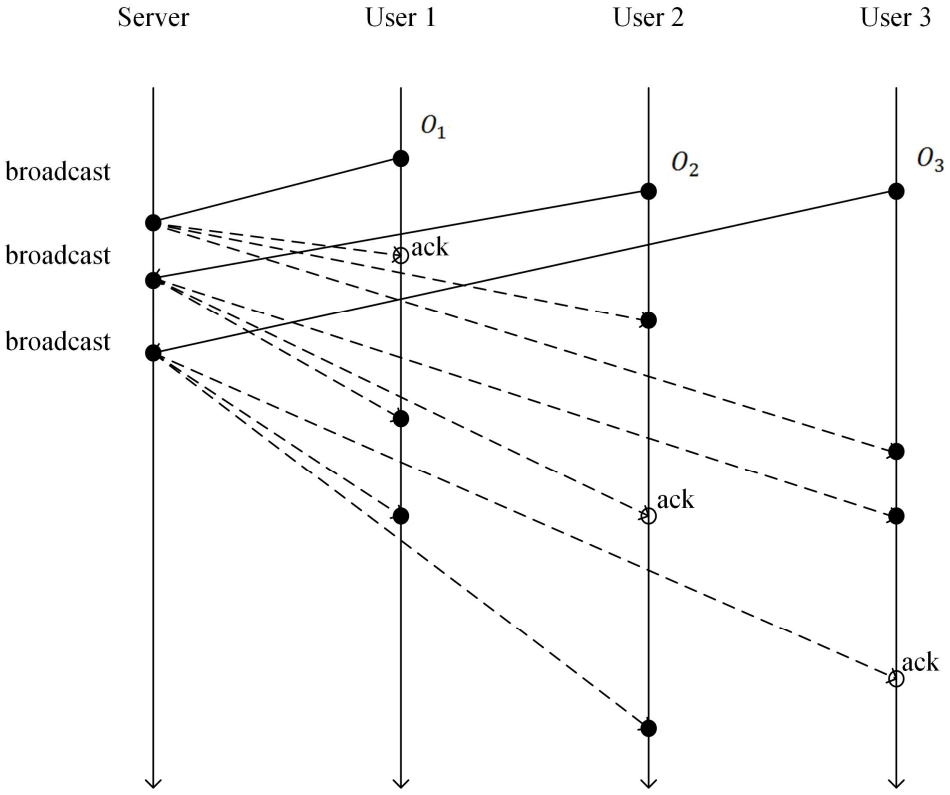


图 3-2：协同旅游规划系统工作时序图

广播给所有用户，其他用户在接收到该操作后通过运行相应的一致性维护算法执行操作，而原用户只需要确认该操作，提取相应的序号信息即可。

从上面的描述可以看出，不同于传统的服务器“客户端拉取”的同步工作模式，为了提高消息的传输速度和避免不必要的性能开销，我们采用的“服务器推送”的异步消息模型去主动告知用户消息的到来而不依赖用户显式的请求。客户端接收到操作后，通过调用基于相应的算法，将副本变为操作能够正确执行的状态。接下来，操作可以在该状态下执行，执行完毕后将副本变为所有接收到的操作以及本地操作执行完毕的状态。

3.4 并发操作的冲突情况

从上文提到的原子操作以及系统框架，我们可以得出如表 3.3 的冲突矩阵。冲突 1，2，3 是跟 POI 序列有关的冲突，因为他们会影响 POI 序列。并发的 AppendPOI 操作和 DeletePOI 操作会导致 index 的改变。由于操作在本地立刻执行，通过可能延迟的网络发送给其他用户的特点，如果我们不对 POI 序列的状态进行转化，则会导致错误的产生。比如，如果 P 序列中的结果为{复旦大学，上海交通大学}，两个用户同时试图删除复旦大学，产生操作 deletePOI(1)。如果不对文档做任何的改变，执行完两个 deletePOI 的结果是 P 序列为空，显然是违反了用户操作意愿的。而两个用户同时产生 appendPOI，由于不同站点执行的顺序不一致，也会引起两个目标 POI 顺序不一致的问题。7，8，9 是跟 E 序列相关的冲突操作，冲突产生的情况和 1，2，3 类似，只是发生在 E 序列当中。10 是一个特殊的冲突，只发生在两个或多个存在时，并且这些 UpdatePOI 的 index 所代表的 POI 相同。通常来说这种冲突在 OT 中会采用“多版本单显式”（Multi-version and Single-display, MVSD）技术解决[50]。而在我们的系统中采用一些特殊而巧妙的方式去解决。4，5，6 是由 deletePOI 引起的，因为 deletePOI 会同时影响 PS 和 ES。对于类似于删除这样的操作，一致性技术进行了相应的探索[12,19]，然而还未对同时影响两个结构的情况给出解决方案。额外的操作需要执行，从而去解决由删除引起的一致性问题。

表 3.3：冲突矩阵

Operation type	*AppendPOI*	*DeletePOI*	*Connect*	*Disconnect*	*UpdatePOI*
AppendPOI	*1*	*2*	⊙	⊙	⊙
DeletePOI		*3*	*4*	*5*	*6*
Connect			*7*	*8*	⊙
Disconnect				*9*	⊙
UpdatePOI					*10*

3.5 本章小结

本章首先介绍了关于协同旅游规划需求的研究，针对相应的需求提出了环境下的数据模型和操作模型。并且给出了针对该模型的相关框架，对于该框架的工作流程进行了简要的说明，并指出了支持多名用户进行协同规划的原因和好处。此外，并发操作会导致冲突，对于冲突操作的产生情况也给出了冲突矩阵，详细的介绍了了产生的原因以及对应的含义。

第四章　消息通信协议

在一致性维护问题中，消息通信时常扮演着重要的角色，本章基于上一章所提出的数据模型，操作模型以及工作框架，将操作分为本地操作和远程操作两类。详细的分析这两类操作之间的关系，然后给出相应的操作关系判定方式。利用操作之间的关系（因果先后，并发）对地址空间进行处理，从而正确的执行操作。同时从理论上进行相关关系判别的正确性证明，并详细给出协同旅游规划模型中一致性算法的具体执行步骤。

4.1 操作关系的定义

通常来说，一致性维护算法的目标并不仅仅维护各个副本的一致性。还需要判断操作的因果关系和操作意愿。如果一致的结果完全不是用户想要的结果，也毫无意义，具体来说，一致性维护算法需要满足以下三个条件[41]。

- **收敛性（Convergence）：**在当每个用户执行完所有用户产生的所有操作之后，每个用户的副本的内容是一致的。
- **因果保持（Causality preservation）：**操作O_2是在另外一个操作O_1执行完之后才产生的，那么在所有站点，O_2都在O_1执行完之后执行。
- **意愿维护（Intention preservation）：**任意一个操作产生的最终效果，在所有站点均保持一致。

收敛性和因果保持很好理解，几乎对于所有的一致性维护算法而言都是普遍适用的。然而，意愿维护中提到的操作效果，跟具体的操作模型相关。操作模型不同，则操作效果也不一样，对于传统的线性结构来说，操作效果指的是

1. 插入的相对位置的一致性。操作O_1在P_1处插入节点N_1，O_2在P_2处插入节点N_2，如果$P_1 < P_2$。那么对于操作O_1和O_2在所有站点的插入位置P'_1和P'_2 也满足$P'_1 < P'_2$.
2. 删除位置的一致性。操作O_1删除节点N_1，那么在所有站点。操作O_1删除节点都是N_1而不是其他节点。

在协同旅游路线规划任务中除了传统操作的效果维护，我们还需要针对旅游路线图结构的特点，提出新的操作效果：

旅游路线图的特殊意愿：对于每一个 Edge 而言，他的存在依赖两个 POI 的存在，起始 POI 和结束 POI，如果其中任意一个 POI 被用户删除，那么依赖该 POI 的 Edge 也会被随之删除。

在地址空间转换算法中，因果关系的判断是非常重要的一部分[19]。只有判断了操作之间的关系，才能确保算法的正确进行，并且满足 CCI 模型。首先给出操作之间的两种关系的定义。操作关系的检测重点在于并发关系的判断，如果是因果先于关系则不用依赖并发控制算法。

因果先于：定义因果关系→，表示存在操作O_1和O_2，O_2在O_1执行完之后才执行，那么$O_1 \rightarrow O_2$。

并发关系：定义并发关系||，如果存在两个操作O_1和O_2，它们既不属于$O_1 \rightarrow O_2$，也不属于$O_2 \rightarrow O_1$，那么它们的关系是$O_1 || \ O_2$。

4.2 异步工作模式的操作关系判别机制

为了掩盖网络延迟对于操作造成的影响，从而实现快速的本地响应，很多时候一致性维护算法会采取在本地保留工作副本的方式。本地操作可以立即在本地副本上执行然后在发送给服务器，然后通过服务器广播给其他用户。传统的地址空间转换算法基于“客户端拉取”的同步工作方式，在移动客户端上，这无疑是一种非常理想工作方式，并且相应方式已经取得了巨大成功[20]。目前来说“服务器推送”是一种更加流行的方式，并且具有性能上的优点[51]。首先，因为跟“客户端拉取”方式不同，他并不需要一个“请求频率”或者其他触发机制去从服务器获取最新操作，而这种触发机制往往对于系统的性能会有一定的影响。“服务器推送”机制使得操作的传播更加实时而不需要无意义的等待时间，因为服务器接收到操作之后经过简单的处理，可以立即广播给其他用户，不需要存储在服务器上等待用户经过一定的频率之后获取（不考虑网络传输中数据丢失和数据传输乱序的情况）。这两个优点使得基于“服务器推送”的工作方式能够实现更好的性能。如图 4-1 给出了两种机制工作的时序图。同步方式一个请求对应一个响应，而异步方式就算没有请求也会接收到服务器的响应。本文接下来将给出基于“服务器推送”的异步工作模式下的

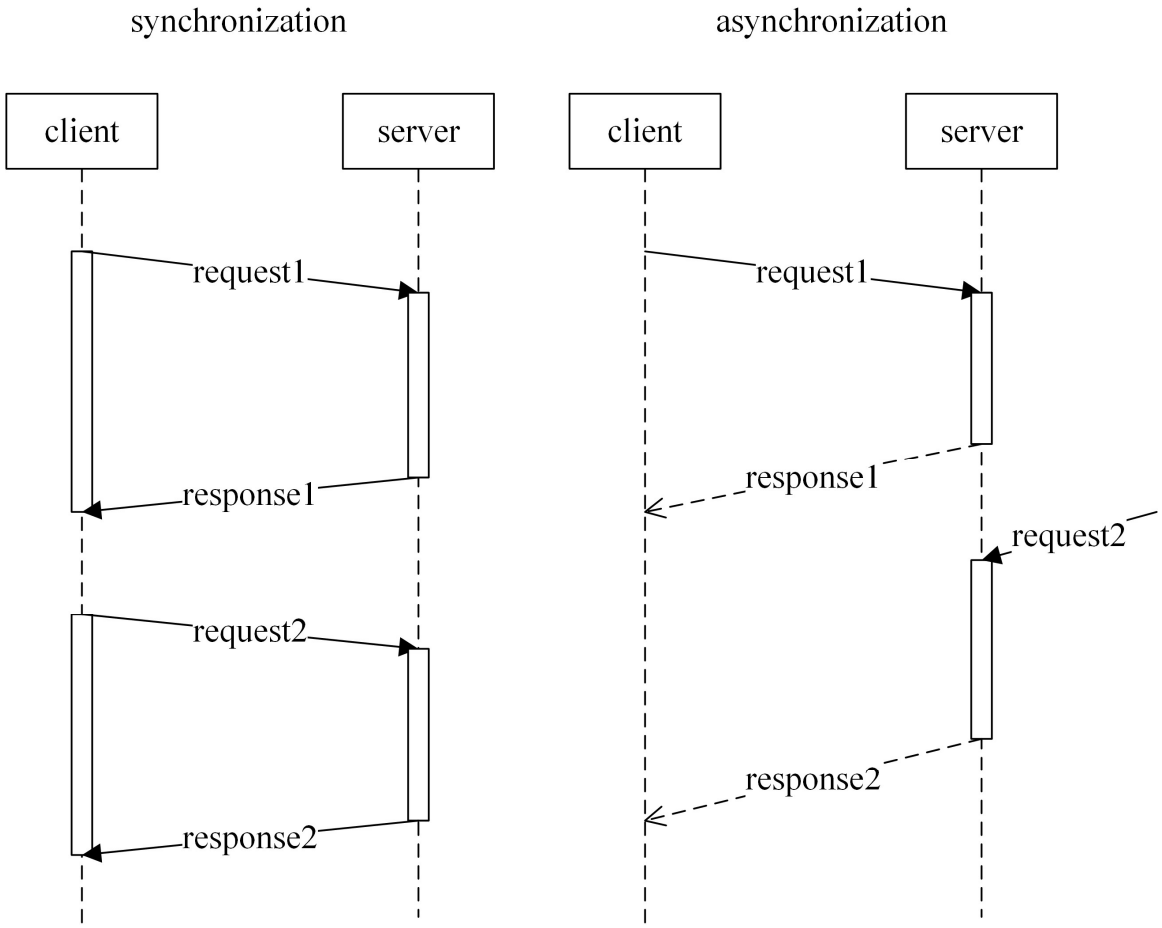


图 4-1：同步工作模式（左）和异步工作模式（右）

操作关系判断机制。

在服务器上，每个接收到的操作会被分配一个连续自增的“服务器接收编号”（server receive number，以下简称为 SRN）。为了便于理解，我们假设服务器端存在一个虚拟操作历史队列（VSHB）去存储所有接收到的操作。而在客户端，有两个操作历史队列（LHB），一个是本地的操作历史队列，存储所有本地产生的操作。另外一个是远程操作历史队列(RHB)去存储所有接收到的服务器推送的远程操作。所有的操作在产生之时都会附加一个时间戳（TS），这个时间戳定义如下所示：

$$TS = < user, opcnt, lastUpdateSRN, SRN >$$

User 用来唯一表示操作的产生者，opcnt 表示操作产生时 LHB 的大小。LastUpdateSRN 表示操作产生时 RHB 中所有操作的 SRN 和 LHB 中已知的 SRN 的最大值。SRN 表示该操作在 VSHB 中的序号，该值初始为空，直到服务器返回服务器分配结果才进行相应的赋值。图 4-2 给出了同步过程。当一个操作 O 被用户 1 创造时，一个时间戳TS_0同时被创建并且附加在操作 O 上，TS_0的前三个属性(user, opcnt,

lastUpdateSRN）可以直接通过本地站点提供的信息获取。这个时候，LHB 和 RHB 中的所有操作均因果先与该操作，因为之前的操作均先与该操作执行完毕。包含了操作信息 O 和时间戳信息TS_0的消息会被发送到服务器上。原本为空的 SRN 会在服务器上根据 VSHB 中的序号分配SRN_i，操作的确认信息（ack）会被发送给原作者，确认信息中附加SRN_i,的信息。然后原作者的 SRN 不再为空。Ack 消息发送的同时，该操作也被广播给所有参与协作的用户，比如图中的 user2。该操作会被放在 user2 的 RHB 当中，注意到 RHB 的所有远程操作均是按照 SRN 升序排列的。在 RHB 中的操作O_r，

1. 满足O_r.SRN <= O.lastUpdateSRN 条件的操作，是因果先于该操作的操作（RHB 中的绿色部分）。
2. 满足O_r.SRN > O.lastUpdateSRN，并且O_r.user = O.user 以及 O_r.opcnt < O.opcnt 的操作，是因果先于该操作的操作（RHB 中的黄色部分的一部分）。

RHB 中的其他的操作与该操作是并发关系。 在 LHB 中的操作O_l可分为三种：

1. O_l的 SRN 为空，那么显然跟操作 O 是并发关系，因为他在 VSHB 中一定排在操作 O 的后面，分配一个比操作 O 更大的 SRN，该 SRN > O.lastUpdateSRN.
2. O_l中的 SRN 不为空，并且满足O_l.SRN > O.lastUpdateSRN. O_l和操作 O 也是并发关系。
3. O_l中的 SRN 不为空，并且满足O_l.SRN <= O.lastUpdateSRN. O_l因果先于 O。

其中 1，2 是图中的蓝色部分，3 是图中的绿色部分。我们可以从上面的讨论中归纳出以下结论：

$$TS_1 \rightarrow TS_2: (1)if \quad TS_1.user = TS_2.user \quad TS_1.opcnt < TS_2.opcnt;$$

$$(2)\mathrm{TS}_1.SRN \; != NULL \; and \quad TS_1.SRN < TS_2.lastUpdateSRN$$

$$TS_1 \parallel TS_2: neither \quad TS_1 \rightarrow TS_2 \; nor \quad TS_2 \rightarrow TS_1$$

根据上面的操作之间因果关系判定法则，新的回溯（Retracing）算法可以归纳如下，遍历相应的地址空间（PS 或者 ES），根据时间戳 TS，检查每个节点上面附加的操作信息，如果操作的时间戳因果先于时间戳 TS，那么获取该操作记录，如果最后无操作记录，则将该节点置为无效，如果有 Disconnect 操作,则也将该节点置为无效，如果是 deletePOI 操作，除了将该节点置为无效，还需要将连接的到该 POI 的边也置为无效，因此回溯 PS 需要在回溯 ES 之后。除了这三个操作之外，还有三个操作 appendPOI, connect 和 updatePOI。他们都是将节点置为有效，其中 updatePOI 是保留 TS 的全序最大的操作，也就是 MVSD（多个版本单一显示），具体怎么判断时间戳的全序将在下一小节给出。具体过程如算法 1 所示：

算法 1 Retrace(Sequence，TS) //Sequence 表示文档序列，可能是 PS 或者 ES，其中 TS 是指时间戳

```
1.  For each node in Sequence do:
2.      操作序列 OL ← {节点上所有因果先于或者等于 TS 的操作}
3.      节点置为无效
4.      If OL 为空 then
5.          继续循环
6.      Else
7.          If OL 中有 DeletePOI 操作 then:
8.              将跟该节点相连的边置为无效，继续循环
9.          End if
10.         If OL 中有且仅有 AppendPOI, Connect 和 update 操作 then
11.             将该节点置为有效，内容为 TS 全序最大那个操作的目标内容
12.         End if
13. End for
```

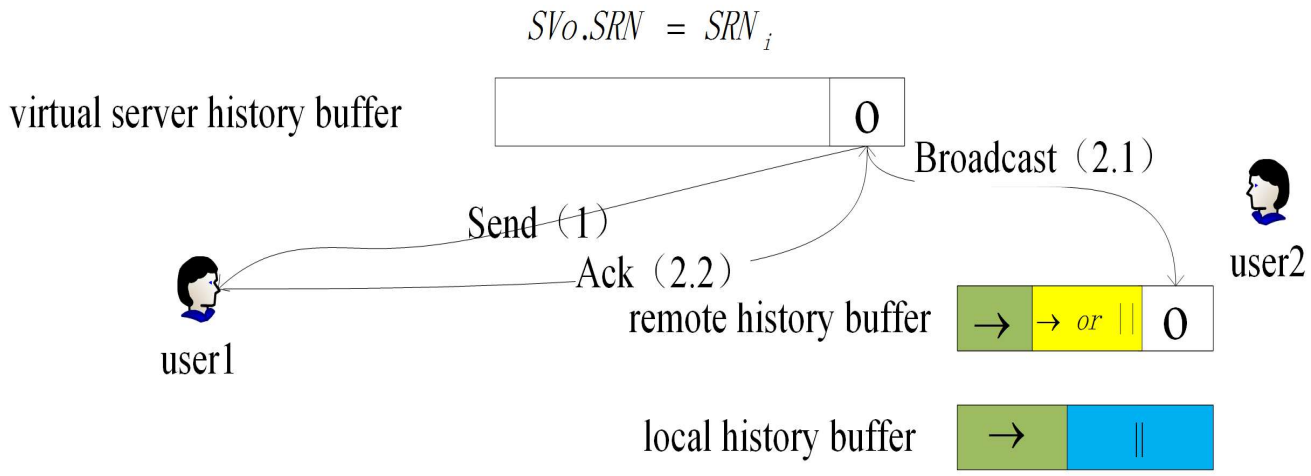


图 4-2：同步流程示意图

4.3 异步工作模式中的客户端工作流程

当用户加入一个协作会话时，如图 4-3 的流程图所示，该协作会话的所有历史操作将会被发送给新加入的用户，从而形成一个副本。该副本是其他用户协作之后的成果，当该会话还没有用户操作产生时，则副本为空。接下来，当本地产生操作

时，产生的操作可以在本地副本上立刻执行而不会被阻塞，因为此时本地所有已经执行过的操作都因果先于该操作，执行完毕后将生成一个包含<user, opcnt, lastUpdateSRN>的时间戳 TS，其中 SRN 为空，user 为该站点标识符，opcnt 为操作产生时 LHB 的大小，lastUpdateSRN 为操作产生时 RHB 和 LHB 中所有操作中出现的最大 SRN。然后将操作信息和时间戳 TS 一起作为消息 MES = <OP, TS>添加到 LHB 尾部，然后发送到服务器端。SRN 等待服务器进行分配后的 ACK 消息确认，因此需要先将 MES 添加到尾部，在发送，防止确认的时候 LHB 中无该消息。

因此，可以将 LHB 看出是一个队列，当本地操作产生时，队列的长度增加，尾指针后移，表示操作入队。当接收到一个 ACK 消息时，头指针所指的 MES 被确认，缺失的 SRN 赋值，头指针后移，表示出队。随着 ACK 消息的接收，opcnt 跟 SRN 可以产生一一对应的关系，从而完成后续操作关系的判断。由于 ACK 消息是异步的，因此并不需要阻塞等待该返回结果。前面也给出了 SRN 为空时的操作关系判断方法。如果没有本地操作产生，则等待其他用户操作到来，如果有其他用户的操作到来，则调用 AST 算法执行其他用户的操作。然后将该远程消息添加到 RHB 的尾部，然后继续判断是否有本地操作产生，重复该流程。注意到 LHB 和 RHB 中的 MES，都是按照 SRN 增加的顺序排列的。因此本地操作产生时，opcnt 和 lastUpdateSRN 可以轻易的得出。

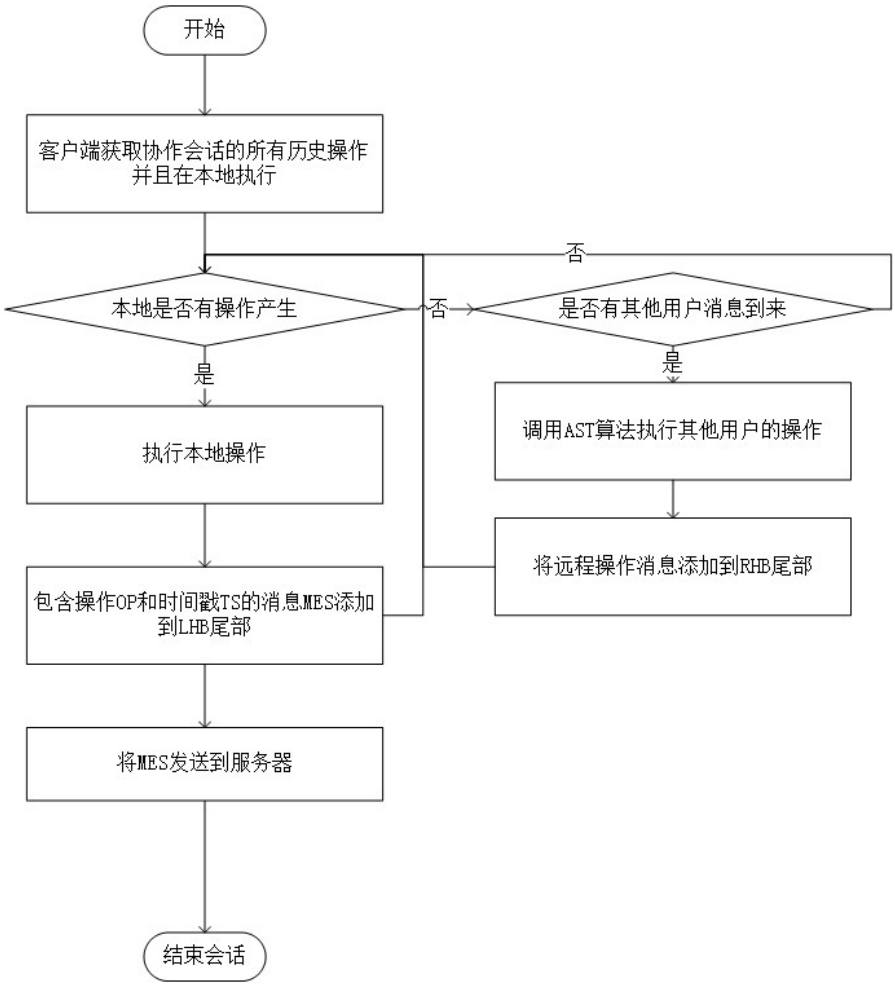


图 4-3：客户端工作流程示意图

4.4 有向图中一致性维护

为了满足协同旅游规划中的 CCI 模型，传统的地址空间转换算法被改写从而适用于这个场景。传统的 AST 核心控制算法主要包含三个步骤[19]：

（1） 第一次回溯：根据操作包含的时间戳信息，将文档恢复到操作能够正确执行时的状态，也就是操作产生时的状态。

（2） 执行操作：根据回溯之后的文档确定操作的位置，执行操作。

（3） 第二次回溯。将文档更新到最新状态。

由于我们的数据模型主要由两个序列组成，分别是由 POI 构成的序列 PS 和表示访问顺序的有向边序列 ES。在 PS 上进行的删除操作会影响 ES，因此回溯过程需要分两步回溯。如算法 2 所示，先将 ES 根据 MES 的 TS 进行回溯，再回溯 PS，因为 PS 中的删除操作会对 ES 产生影响。

算法 2 Control (MES， ES， PS， TS)　//MES 表示操作信息，ES 表示有向边构成的序列，PS 表示 POI 构成的序列，TS 表示相应时间戳

1. Retrace (ES, MES.TS)
2. Retrace (PS, MES.TS)
3. 执行操作 MES.OP
4. 将 MES 添加到 RHB 末尾。
5. TS.lastUpdateSRN ← MES.SRN
6. Retrace (ES, TS)
7. Retrace (PS, TS)

Disconnect 和 UpdatePOI 只需要在相应的序列中找到相应的元素进行修改即可，通过 Retrace 算法能够唯一的确定目标节点，而 AppendPOI 和 Connect 是在序列的末尾添加节点，这个操作跟传统的“插入”操作类似[19,41]，需要在插入位置中的一连串无效节点中寻找实际插入位置。在地址空间中的寻找实际插入位置的算法需要进行相应的变化，才能满足新的需求。由于传统的向量时间戳通过相应的转换算法 Torder 可以映射到一个全序集当中，使得每个操作的时间戳满足

1. 存在两个向量时间戳SV_1和SV_2，如果$Torder(SV_1) \leq Torder(SV_2)$，并且$Torder(SV_1) \geq Torder(SV_2)$，那么$Torder(SV_1) = Torder(SV_2)$。实际在分配向量时间戳的时候绝不会给两个操作分配相等的时间戳（全序关系的反对称性）。
2. 存在三个向量时间戳SV_1，SV_2和SV_3，如果存在$Torder(SV_1) \leq Torder(SV_2)$，并且$Torder(SV_2) \leq Torder(SV_3)$，那么必然满足$Torder(SV_1) \leq Torder(SV_3)$（全序关系的传递性）。
3. 存在两个向量时间戳SV_1和SV_2，要么$Torder(SV_1) \leq Torder(SV_2)$，要么$Torder(SV_1) \geq Torder(SV_2)$（全序关系的完全性）

本系统算法采用标量时间戳TS $< user, opcnt, lastUpdateSRN, SRN >$取代了传统的向量时间戳，为了保证 range-scan 算法的正常运行，同样需要将 TS 通过相应的 Torder 方法转换到全序集中。通过上面的描述我们可以发现，SRN 本身就是一个全序序列。我们只需要通过对TS_i投影到 SRN 上得到TS_i(SRN)，所有的TS_i(SRN)就能构成一个全序集合。

$$\mathrm{TS_i.SRN} < \mathrm{TS_j.SRN} \Rightarrow \mathrm{Torder(TS_i)} < \mathrm{Torder\left(TS_j\right)} \tag{5.1}$$

然而，跟时间戳并发关系判定所面临的问题一样，SRN 并不是每个操作产生之后就存在的，在服务器分配 SRN 之前，他的值一直为空。同样，我们可以根据其他三个的值来简介的判断他们的全序关系，当$\mathrm{Torder}(\mathrm{TS}_1) < \mathrm{Torder}(\mathrm{TS}_2)$，得到如下的三种情况满足条件。

1. $\mathrm{TS}_1.user = \mathrm{TS}_2.\mathrm{user}$ 并且$\mathrm{TS}_1.opcnt < \mathrm{TS}_2.\mathrm{opcnt}$。
2. $\mathrm{TS}_1.SRN \neq \mathrm{NULL}$ 并且 $\mathrm{TS}_1.SRN = \mathrm{NULL}$
3. $\mathrm{TS}_1.SRN \neq \mathrm{NULL}$，$\mathrm{TS}_2.SRN \neq \mathrm{NULL}$ 并且 $\mathrm{TS}_1.SRN < \mathrm{TS}_2.SRN$。

根据上面的定义，range-scan 算法就能通过全序关系确定节点的实际前驱节点和最终插入位置。

DeletePOI 有着自己特殊性，因为它会同时影响 PS 和 ES。Connect 的两个参数 startPOIndex 和 endPOIndex，实际对应的 POI 是需要根据 Connect 的 TS 确定的，那么 DeletePOI 在试图删除 Edge 的时候，需要按照 ES 中每个创建 Edge 的 Connect 操作的 TS 来回溯 PS，从而确定当前的 Edge 是否会被该 DeletePOI 删除，回溯的次数跟 Connect 的总操作次数相同，随着协作的进行 Connect 操作数目的上升，无疑会花费大量的时间，导致 DeletePOI 操作的时间复杂度慢慢上升。为了解决这个问题，节约不必要的回溯次数，我们给每个 POI 分配一个 UIDoP，在本地站点中 POI 和 UIPoD 存在一一对应关系，不同站点的 UIDoP 的关系不做进行强制要求，从而避免额外的通信。我们可以仿照 SRN 的方式进行分配 UIDoP 或者其他方式来分配，只需要保证一一对应关系即可。 在满足一一对应的前提下，我们可以通过 UIDoP 来找到 POI，在 Connect 的时候，Edge 可以添加两个属性<startPOIUIDoP,endPOIUIDoP>来隐式指定实际的 POI，从而无需对整个 Edge 进行遍历。

4.5 本章小结

本章的第一部分介绍了协同旅游操作系统中的操作关系进行了定义，便于后面对于操作判定算法的说明，后面由此详细介绍了基于“服务器推送”的操作关系的判定机制。第三部分在该机制的基础上，结合旅游规划任务的特点，给出了客户端的工作流程。最后对全序关系给出了定义和计算方法，并且介绍了旅游路线规划中一致性维护的关键算法。

第五章　算法改进和效率分析

传统的地址空间转换算法，在两次 retrace（回溯）过程中花费了大量的时间，而最终大部分节点的有效性没有发生变化。针对这一情况，本章将给出相应的改进算法，完全避免了 retrace 的过程，从而大大提高算法的运行速度。对于传统的控制算法的修改和完善将会被给出，我们将实现这个原型系统，并给出原型系统的介绍，并搭建相应的测试环境，通过算法性能的比较，分析算法的执行效率。

5.1　基于唯一标识符的模型改进

从上文可以看出，UIDoP 的运用避免了大量的回溯过程。因此，不妨将唯一标识符的思路扩大到所有节点，并且重新定义节点之间的关系。过去，我们将节点的关系通过 index 来表明，随着操作数目的上升和文档长度的增加，节点的 index 一直在变化，为了找到对应的节点，我们往往需要根据时间戳回溯之后，才能根据 index 找到相应的节点，因此大量的时间被用在了回溯上。如果我们给每个节点分配一个唯一的标识符（identifier），并且通过 identifier 来查找节点，这个时候就没有回溯的必要了，基于这样的思路，我们将重新定义数据模型和操作模型

5.1.1 改进的数据模型

IG 中 PS 和 ES 两个线性序列包含一系列的节点，每个节点可以根据唯一标识符去找出来对应的节点。主要包含的依然是 POI 和 edge 节点。 POI 和 edge 被重新定义为这样的五元组节点 node <identifier, data, nextId,TS_{ins}，TS_{del}，TS_{upd}>，其中：

- identifier 是用来区分每个节点的唯一标识符。
- data 为节点储存的内容，比如是 POI 还是 edge，POI 包含的基本信息（经纬度，标题，地址等信息）或者 edge 包含的基本信息（描述访问顺序的信息）。
- nextId 为后继节点标识符，通过该标识符可以查找后继节点。
- TS_{ins}是创建该节点的操作的时间戳。
- TS_{del}是删除该节点的操作中全序最小的时间戳。
- TS_{upd}是修改该节点的操作中全序最大的时间戳

接下来我们将定义节点之间的连接关系，我们用<表示节点之间的逻辑“前驱-后继”关系，用$<_r$表示节点之间的实际“前驱-后继”关系，具体形式如下：

1. $n_1 < n_2 \Leftrightarrow n_1$是$n_2$的逻辑前驱，表示在用户产生创建$n_2$操作的时候，所能看到的前驱节点是$n_1$。
2. $n_1 <_r n_2 \Leftrightarrow n_1$是$n_2$的实际前驱，表示在实际执行创建$n_2$操作之后，$n_1$的 nextId 是$n_2$的 identifier。

首先我们会创造对用户隐藏的节点n_{start}和n_{end}，并且满足$n_{start} <_r n_{end}$，然后我们才可以在这两个节点的基础上进行操作。基于以上的定义，我们的模型依然是可以看做线性序列，如图 5-1 所示。PS 和 ES 分别对应两个这样的线性序列及其组织关系。

5.1.2 改进后的操作模型

改进后的操作模型主要由三种原子操作组成，分别是 Insert，Delete 和 Update，具体的定义以及操作影响如下。

1. Insert(id_{pre}, n_{new}, TS):在id_{pre}所表示的节点n_{pre}后面添加一个节点n_{new}，操作上下文可以表述为如下：
   a) 前置条件：$n_{pre} < n_{next}$
   b) 后置条件：$n_{pre} < n_{new} < n_{next}$
2. Delete(id_{del}, TS):表示删除id_{del}所表示的节点n_{del}，操作上下文可以表述为如下：
   a) 前置条件：$n_{pre} < n_{del} < n_{nex}$
   b) 后置条件：$n_{pre} < n_{next}$
3. Update(id_{upd},newdata, TS):表示更新id_{upd}所表示的节点n_{upd}中的 data，操作上下文可以表述为如下：
   a) 前置条件：n_{upd}.data = olddata
   b) 后置条件：n_{upd}.data = newdata 当且仅当 TS 比该节点其他非删除操作的 TS 大。

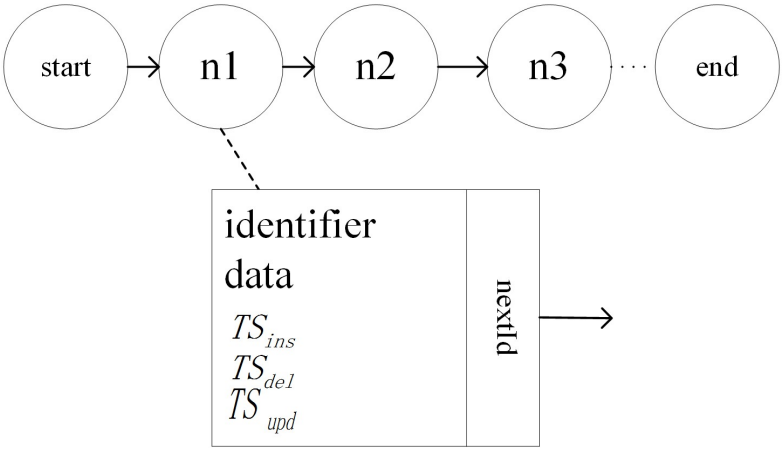


图 5-1：改进后的线性序列

5.2　无标记回溯的一致性维护算法

为了实现高的本地响应速度，本地操作应当能够立刻执行。但是当操作传递到其他站点时，其他站点的文档状态可能不是该操作产生时候的状态，直接执行该操作可能会导致错误或者每个用户的副本结果不一致，这些都是我们一致性维护算法所需要解决的问题，传统的地址空间转换算法依赖“标记-回溯”过程，才能将目标文档的状态恢复到操作产生时的状态，避免了操作错误和不一致结果的出现。然而随着文档的增大，“标记-回溯”成了影响系统性能的瓶颈，然而传统的地址空间转换算法又不能离开“标记-回溯”过程，因为 index 信息依赖“标记-回溯”之后确认。

然而我们通过观察回溯过程后发现，添加和更新操作并不会修改原有节点的有效情况，删除操作也只修改一个节点和与其相关的节点有效信息。大部分并发操作引起的一次回溯产生的状态变化在二次回溯后会复原。因此，很多节点的有效情况变化并不是有效的，我们完全可以避免。所以，利用改进后的模型，我们通过标识符的方式对原有的线性结构用单向链表结构重新组织。有两种结构可以是通过标识符来查找节点，分别是比较稳定的添加删除复杂度为 O（logn）的平衡树结构，或者不稳定但复杂度为 O（1）散列表结构，由于避免了复杂度为 O（n）的“回溯”过程，性能要比传统的地址空间转换性能好，具体的性能分析我们将在后面给出。

算法 3 给出插入操作的具体执行过程。从上面的描述可以知道，由于现在节点不再通过 index 定位，而是通过唯一标识符在 IG 中找出来。因此我们可以通过因果前驱节点的标识符，直接找到因果前驱节点n_{pre} （行 1），避免了 retrace 的过程，而且找到的n_{pre}必然满足$n_{\text{pre}}.TS_{\text{ins}} \rightarrow n_{new}.TS_{\text{ins}}$。这一点很好证明，因为如果用户

在产生这个操作的时候，必然看到了n_{pre}这个节点的存在（否则操作不合理），也就是说这个节点是有效的，产生这个节点的操作必然因果先于当前的 insert 操作，因此他们之间的时间戳存在明显的因果关系。但是，节点的逻辑前驱和实际前驱往往并不一样。通过传统的地址空间转换算法中的 range-scan 过程可知，逻辑前驱后面可能存在着一系列无效节点，这部分节点有可能是跟该操作并发的 Insert 操作产生的节点，也有可能是被删除导致无效的节点，他们都有可能成为n_{new}的实际前驱，如果确定n_{new}的实际前驱节点，依然需要通过 range-scan 算法来找到（行 2），这里的 range-scan 算法需要根据新的链表结构做一些调整。因为这个先后顺序是通过 nextId 确定，而不能直接通过 index 取得。然后，我们对新节点的两个时间戳进行赋值，其中$TS_{\text{ins}} = TS$，而$TS_{\text{del}} = TS_{\text{infinite}}$（行 4-5）.其中$TS_{\text{infinite}}$表示所有除了$TS_{\text{infinite}}$之外的其他时间戳直接都满足：$TS \rightarrow TS_{\text{infinite}}$。跟单向链表增加元素类似，修改相应节点的 nextId 可以调整线性序列（行 6-7），而不需要通过“数组移位”的方式（插入一个节点，需要后面的节点往后移动一位腾出地方）。最后在 IG 中存储该节点，并且将节点置为有效。

算法 3 Insert (preId， n_{new}，TS) //preId 表示插入节点的因果前驱节点，n_{new}表示待插入的节点，TS 表示操作执行时的时间戳。

1. 通过 preId 在 IG 中找出相应的节点n_{pre}
2. $n_{\text{realPre}} \leftarrow$ range-scan(a_{pre}, TS)
3. 通过n_{realPre}的 nextId 在 IG 中找出节点n_{next}
4. $n_{\text{new}}.TS_{\text{ins}} \leftarrow TS$
5. $n_{\text{new}}.TS_{\text{upd}} \leftarrow TS$
6. $n_{\text{new}}.TS_{\text{del}} \leftarrow TS_{\text{infinite}}$
7. n_{realPre}. nextId $\leftarrow n_{\text{new}}.identifier$
8. n_{new}.nextId $\leftarrow n_{\text{next}}.identifier$
9. 在 IG 中保存a_{new}并且将它置为有效节点

根据类似的方法，我们可以对 delete 操作和 update 操作进行相应的改进。这两种操作可以根据标识符直接找到目标节点，并且根据全序关系进行更新，其中 delete 的TS_{del}设置为所有并发操作中全序最小（或最大）的 TS，而 update 的信息（data）设置为全序关系中最大（或最小）的信息。注意到以上三种新的操作，并不影响目标节点之外节点的有效无效信息，回溯所带来的负面影响完全消除。以前每个节点会附加所有跟该节点相关的 insert 和 delete 操作，我们这边一个节点只需要存储与操作相关的时间戳信息即可，大大节省了存储资源。具体过程如算法 4 和算法 5 所示。

算法 4 Delete (identifier，TS)　//identifier 表示目标节点，TS 表示操作执行时的时间戳。

1. 通过 identifier 在 IG 中找出相应的节点n_{delete}
2. **If** Torder(TS) < Torder($n_{delete}.TS_{\text{del}}$) then:
3. 　　$n_{delete}.TS_{\text{del}} \leftarrow TS$
4. **End if**
5. 将n_{delete}置为无效节点

算法 5 Update (identifier，n_{new}, TS)　//identifier 表示目标节点，n_{new}表示新的节点，TS 表示操作执行时的时间戳。

1. 通过 identifier 在 IG 中找出相应的节点n_{update}
2. **If** Torder(TS) > Torder($n_{update}.TS_{upd}$) then:
3. 　　$n_{update}.TS_{upd} \leftarrow TS$
4. 　　$n_{update}.\text{data} \leftarrow n_{new}.data$
5. **End if**

由于并发操作的存在，逻辑前驱节点和实际前驱节点可能并不一样，因此需要 range-scan 算法来确定实际前驱节点的位置，传统的 range-scan 算法依赖 retrace 的过程来确定 range-scan 算法所需要的两个参数：逻辑前驱节点和逻辑后继节点。然后 range-scan 才能在这两个节点之间找出实际前驱节点。改进后的 Insert 可以通过标识符直接找到逻辑前驱节点，避免了 retrace 来找到逻辑前驱节点的过程。此外，将 retrace 判断节点是否有效的方法引入 range-scan 中，可以直接通过逻辑前驱节点的后继无效节点（并发操作产生的节点）信息，找出实际前驱节点。Range-scan 的过程如算法 6 所示。首先找到逻辑前驱节点，遍历它的后继节点中的无效节点，也就是不是因果先于待执行插入操作的时间戳的后继节点序列。当找到第一个时间戳的全序比 TS 的全序要大的节点，将目标节点赋值给它（行 5-7）。跟插入排序类似。然而，用户可以通过因果操作，使得一段并发操作并不按照全序由小到大排列，这个时候需要通过判断这部分由于因果操作所导致的逆序序列，并将操作合理的插入逆序序列中使得插入的操作在合适的位置（行 8-10）。主要通过这两步操作进行实际前驱节点的查询。注意到文档在初始化的时候由两个节点n_{start}和n_{end}，他们的TS_{ins}为特殊的时间戳TS_{zero}，对于所有除了TS_{zero}之外的时间戳 TS，都有$TS_{\text{zero}} \rightarrow \text{TS}$，因此保证了的 range-scan 插入的范围不会超过文档的范围。最后将目标节点返回，让 Insert 算法正确插入到无效节点序列中。

算法 6 Range-scan (n_{pre}, TS) //n_{pre}表示逻辑前驱节点，TS 表示操作执行时的时间戳。

1. $n_{\text{realPre}} \leftarrow$ null
2. $n_{\text{cur}} \leftarrow$通过$n_{\text{pre}}$.nextId 在 IG 中找出对应的节点
3. $TS_{\text{cur}} \leftarrow n_{\text{cur}}.TS_{\text{ins}}$
4. **While** !($TS_{\text{cur}} \rightarrow$ TS) do
5. **If** Torder(TS) < Torder(TS_{cur}) and n_{realPre} = null then
6. $n_{\text{realPre}} \leftarrow n_{\text{Pre}}$
7. **End if**
8. **If** Torder(TS_{cur}) < Torder(TS) and n_{realPre} != null and $TS_{\text{cur}} \rightarrow n_{\text{realPre}}.TS_{\text{ins}}$ then
9. $n_{\text{realPre}} \leftarrow$ null
10. **End if**
11. $n_{\text{pre}} \leftarrow n_{\text{cur}}$
12. $n_{\text{cur}} \leftarrow$通过$n_{\text{pre}}$.nextId 在 IG 中找出对应的节点
13. $TS_{\text{cur}} \leftarrow n_{\text{cur}}.SV_{\text{ins}}$
14. **End while**
15. **If** n_{realPre} = null then
16. Return n_{pre}
17. **Else**
18. Return n_{realPre}
19. **End if**

5.3 算法结果一致性的证明

如果只有一个用户在编辑共享地址空间中的文档，这种极端情况下跟单机的文档编辑应用并无区别。在这种情况下，显然不会造成文档的不一致，因为全局的所有操作之间存在严格的因果链，只有当一个操作在所有副本执行完之后，下一个操作才执行。不失一般性，如果多个用户在编辑共享地址空间中的文档，每个用户在产生操作到在所有副本执行完的这段时间内，其他用户不产生操作。那么这种情况跟一个用户在编辑共享空间的情况一致。根据第四章的提到的新的通信协议我们可以发现，具有因果关系的两个操作在所有站点的执行顺序跟因果顺序一致，因此在本系统中，具有因果关系的操作并不会违反 CCI 模型。此外，删除操作由于唯一标

识符的存在，也不会存在二义性的情况。因此，一致性问题主要解决的是插入操作所带来的不一致的情况。如果是两个并发插入操作O_1和O_2，他们的逻辑前驱节点一致，而他们在各个站点的执行顺序不一样，插入的位置会造成不一致。协同编辑中造成副本不一致的根本原因是在共享文档中插入顺序不一致。通过判断全序关系和因果关系可以确定两个并发插入操作之间产生节点的位置关系。

5.3.1 初始状态一致的 d 个操作

令S_i表示站点接收到 i 个服务器的操作广播信息（包括 ack 消息）所形成的文档状态。考虑有一共 d 个操作$O_1,O_2 \ldots O_d$，如果他们的时间戳 TS. lastUpdateSRN 一致，令$|S_i|$ = i 表示形成文档状态S_i所需要执行的操作个数，那么 TS. lastUpdateSRN = $|S_i|$，意味着这 d 个操作是在文档状态为S_i的时候产生的所有操作。$S_i + \Delta d$表示在状态S_i的时候，每个站点将待执行操作序列Δd分为两部分$\{L_n, R_m\}$，其中L_d表示站点在S_i到S_{i+d}产生的本地操作序列$l_1,l_2 \ldots l_n$，R_d表示其他站点产生的操作序列$r_1,r_2 \ldots r_m$，并且满足 m + n = d。注意，对于每个站点L_d，R_d的集合不一样。此外，根据新通信协议，这两个序列中的操作均按照全序由小到大排列，因此 Torder(l_1) < Torder(l_2)<… < Torder(l_n)，Torder(r_1) < Torder(r_2)… < Torder(rm)。站点按照先执行L_d序列后执行R_d序列的顺序执行操作，有 $S_i + \Delta d = S_i + L_n + R_m$。我们给出如下定理：

定理 1：给定操作状态S_i和在S_i一共产生的 d 个操作$O_1,O_2 \ldots O_d$序列Δd，那么对于每个协作站点在S_i执行完 d 个操作后的文档一致，满足$S_{i+d} = S_i + \Delta d$。

证明如下：

显然，除了删除操作不影响操作效果之外，如果两个或多个并发操作O_1,O_2的 lastUpdateSRN 一样。而他们的逻辑前驱节点都不一样，他们的操作效果是完全无关的。因此，我们只需要将Δd按照逻辑前驱节点一样来进行分组然后分别考虑即可。为了说明简单，不妨设Δd中的每个操作均为插入操作，并且逻辑前驱一致。

情况 1，Δd中的任意两个不同的操作O_1,O_2，满足$O_1.TS_{ins} \ || \ O_2.TS_{ins}$。

产生这种情况的情景是在S_i的时候，每个站点产生 0 个或者 1 个操作，这些操作形成Δd满足Δd中的任意两个不同的操作O_1,O_2满足$O_1.TS_{ins} || \ O_2.TS_{ins}$，他们的逻辑前驱节点相同，我们按照这些操作的全序关系插入排序，无论操作的执行顺序如何，他们的执行效果均是在逻辑前驱节点后按照全序关系由小到大排序。不妨假设$\Delta d = \{O_1,O_2 \ldots O_d\}$，如果Torder($O_1$) < Torder($O_2$) < … < $Torder(O_d)$，令 N(O_1)表示操作O_1插入的节点，根据算法 6，无论哪个站点，他们的顺序关系满足

$N(O_1) <_r N(O_2) <_r \ldots <_r N(O_n)$。显然，在这种情况下$S_{i+d} = S_i + \Delta d$。并且我们发现当两个操作是并发操作时，操作顺序不影响执行效果，由于其中R_d的每个操作都和L_d中的操作并发，我们可以得到定理 2：

$$S_i + L_d + R_d = S_i + R_d + L_d$$

情况 2，Δd中的任意两个不同的操作O_1,O_2，要么满足$O_1.TS_{ins} \mid\mid O_2.TS_{ins}$，要么满足$O_1.TS_{ins} \rightarrow O_2.TS_{ins}$并且存在一系列的$O_{i1}$，$O_{i2} \ldots O_{in}$使得$N(O_1) <_r N(O_{i1}) <_r N(O_{i2}) <_r \ldots <_r N(O_{in}) <_r N(O_2)$。

当起始状态相同时，来自任意两个不同站点的操作必然属于并发关系。而这种情况发生的是在S_i的时候，每个站点产生 0 个或者多个操作，并且如果某个站点产生了 2 个或者多个操作，那些操作都是在上一个操作产生的节点后面添加节点。注意到如果一个站点产生了 2 个或者多个操作，只要操作都是在上一个操作产生的节点后面添加节点，使得$O_1.TS_{ins} \rightarrow O_{i1}.TS_{ins} \rightarrow O_{i2}.TS_{ins} \rightarrow \cdots \rightarrow O_{in}.TS_{ins} \rightarrow O_2.TS_{ins}$，且$N(O_1) <_r N(O_{i1}) <_r N(O_{i2}) <_r \ldots <_r N(O_{in}) <_r N(O_2)$。因为对于两个操作如果满足$O_1.TS_{ins} \rightarrow O_2.TS_{ins}$，那么$Torder(O_1) < Torder(O_2)$,所以形成的序列依旧能够保持按照全序升序排列，$Torder(O_1) < Torder(O_{i1}) < Torder(O_{i2}) < \cdots < Torder(O_{in}) < Torder(O_2)$。因此结论跟情况 1 一致，不再赘述。

情况 3，Δd中的至少有一对不同的操作O_1,O_2，使得$O_1.TS_{ins} \rightarrow O_2.TS_{ins}$，并且$N(O_2) <_r N(O_1)$。

产生这种情况的情景是在S_i的时候，每个站点产生 0 个或者多个操作，并且如果某个站点产生了 2 个或者多个操作，其中至少存在一对操作满足，他们的 preId 相同(相同的两个可以不是O_1，O_2)。操作的效果是导致在同一个 preId 对应的节点的后继节点并不按照全序由小到大排序。也就是$N(O_2) <_r N(O_1)$，但是$Torder(O_1) < Torder(O_2)$。由于这种情况所造成的线性序列不再满足单调递增特点。我们将这种序列称之为全序逆序序列（TRS）。按照算法 6，一个这样的序列中的所有操作由一个用户产生，并且满足$O_1.TS_{ins} \rightarrow O_2.TS_{ins}$。一个用户可以生成多个 TRS，甚至 TRS 中还可以嵌套 TRS。处理 TRS 的方法是将所有 TRS 合并，用序列中全序最小的节点替代该 TRS，递归该过程直至不存在任何 TRS。此时，剩下的节点满足全序单调递增规律。我们设TRS_i中全序最小的节点为TRS_i.min，假设最后剩下 m 个 TRS 合并后的节点，那么最后结果满足$TRS_1.min < TRS_2.min < \ldots < TRS_m.min$，$Torder(TRS_1.min) < Torder(TRS_2.min) < \ldots < Torder(TRS_m.min)$。其他非 TRS 中的节点$O_1$,$O_2$,…$O_n$和所有$TRS_i.min$满足按照全序从小到大排序，意味着如果存在$N(O_1) <_r TRS_1.min <_r N(O_2) \ldots <_r TRS_i.min <_r N(O_n)$那么必然满足

$Torder(O_1) < Torder(TRS_1.\min) < Torder(O_2)$ … $< Torder(TRS_i.\min) < Torder(O_n)$

由于 TRS 中的节点是同一个用户产生并且是因果操作，他们的顺序很容易确定，因此按照TRS. min的位置关系可以还原整个 TRS。由于本质上该情况的执行效果可以通过 TRS 合并的方式规约化为在逻辑前驱节点后按照全序关系由小到大排序，因此$S_{i+d} = S_i + \Delta d$。具体例子可参照图 5-2 给出的示意图。根据上述三种情况的讨论，定理 1 得证。并由此推出定理 3:

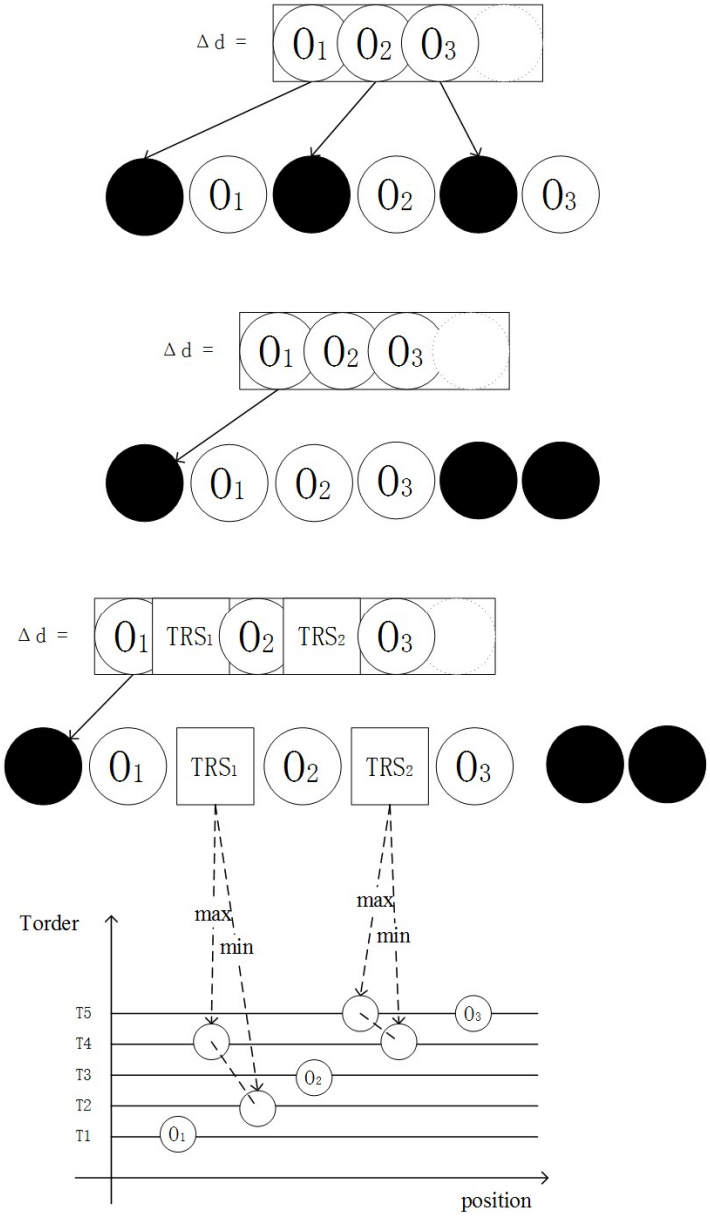


图 5-2：初始状态一致的 d 个操作的执行结果示意图

定理 3：给定操作状态S_i和在S_i一共产生的 d1 个操作O_1,O_2…O_{d1}序列Δd1，在S_{i+d1}一共产生的 d2 个操作O_1,O_2…O_{d2}序列Δd2那么对于每个协作站点在S_i执行完 d 个操作后的文档一致，满足$S_{i+d1+d2} = S_i + \Delta d1 + \Delta d2$。

由定理 1 可知$S_{i+d1} = S_i + \Delta d1$，$S_{i+d1+d2} = S_{i+d1} + \Delta d2$，由此推出$S_{i+d1+d2} = S_i + \Delta d1 + \Delta d2$

5.3.2 一共 N 个操作的执行结果

不妨将定理 1 拓展到一个完整的工作流程当中，从工作开始到工作结束一共产生 N 个操作，这 N 个操作按照我们的消息通信机制进行数据的分发，工作运行的结果满足如下定理。

定理 4：在一次协同会话中，每个站点执行完会话过程中产生的全部 N 个操作后，副本是完全一致的，即$S_{0+N} = S_0 + \Delta N$，其中S_0是空文档，S_{0+N}表示执行完 N 个操作之后的文档。

证明如下：不妨设所有站点在状态S_0产生n_0个操作，S_1产生n_1个操作，以此类推，一直到S_{N-1}产生n_{N-1}个操作，并且满足$N = \sum_{i=0}^{N-1} n_i$。 根据通信协议可知操作的广播顺序为Δn_0，Δn_1…Δn_{N-1}(可能为空)，因此对于每个站点 si，执行顺序满足：

$$S_{0+N}[si] = \Sigma_{i=1}^{N}(L_i[si] + R_i[si]) \quad (5.1)$$

$L_i[si]$和$R_i[si]$分别表示站点 si 在S_{i-1}到S_i产生的本地操作序列和接收到的远程操作序列（序列可能为空）。而$S_{0+N}[si]$，注意区别开S_{0+N}，前者表示某个站点的状态，后者表示全局的文档状态。同理可得，执行完前n_0个操作的状态满足：

$$S_{n_0}[si] = \Sigma_{i=1}^{n_0}(L_i[si] + R_i[si]) \quad (5.2)$$

根据通信协议可知，上面公式中每个R_i属于Δn_0，L_i中一部分属于Δn_0，另一部分属于Δn_1，不妨设这个分界操作为$L_{n_0'}$,那么L_1，L_2 … $L_{n_0'}$属于Δn_0，余下的属于Δn_1。根据定理 2，可对式子 5.2 进行整理后得：

$$S_{n_0}[si] = \Sigma_{i=1}^{n_0} R_i[si] + \Sigma_{i=1}^{n_0'} L_i[si] + \Sigma_{i=n_0'+1}^{n_0} L_i[si]$$

根据定理 1 可知:

$$S_{n_0} = \Sigma_{i=1}^{n_0'} L_i[si] + \Sigma_{i=1}^{n_0} R_i[si]$$

表示如果每个站点去除了操作序列$\Sigma_{i=n_0'+1}^{n_0} L_i[si]$ 所达到的状态一致。而：

$$S_{n_0+n_1} = S_{n_0} + \Sigma_{\mathrm{i}=n_0'+1}^{n_0} L_i[\mathrm{si}] + \Sigma_{\mathrm{i}=n_0+1}^{n_0+n_1'} L_i[\mathrm{si}] + \Sigma_{\mathrm{i}=n_0+1}^{n_0+n_1} R_i[\mathrm{si}]$$

以此类推，由于最后到Δn_{N-1}的后面没有其他后续操作，则：

$$S_{\mathrm{N}}=S_{\mathrm{N}-n_{N-1}} + \Sigma_{\mathrm{i}=(\mathrm{N}-n_{N-1})'+1}^{\mathrm{N}-n_{N-1}} L_i[\mathrm{si}]+\Sigma_{\mathrm{i}=(\mathrm{N}-n_{N-1})+1}^{\mathrm{N}} L_i[\mathrm{si}] + \Sigma_{\mathrm{i}=(\mathrm{N}-n_{N-1})+1}^{\mathrm{N}} R_i[\mathrm{si}]$$

再结合定理 3 可推出：

$$S_{\mathrm{N}} = S_0 + \Sigma_{\mathrm{i}=1}^{\mathrm{N}-1} \Delta n_i = S_0 + \Delta \mathrm{N}$$

由此得证。

5.4　算法效率分析

根据[19]可知，传统的地址空间转换算法的复杂度开销主要在回溯过程和操作执行过程，回溯算法的复杂度是 O(n*d)。其中 n 表示文档的大小，d 表示为每个节点平均附加的操作数量。根据操作的不同，操作执行的复杂也有所不同，删除操作只需要修改对应节点的有效位信息，所以操作复杂度为 O(d)。而插入操作由于 range-scan 算法运行的过程，需要对操作目标位置范围的 m 个并发节点进行遍历确定实际插入位置，因此复杂度为 O(m)。通常来说 m 不会特别大，因此算法的瓶颈主要在于回溯过程，随着插入操作数目的上升，文档规模越来越大，操作的执行效率将会收到非常大的影响。

因此[19]提出了通过利用红黑树来改进传统的地址空间转换算法，利用树的结构性质，将插入操作的复杂度优化为 O(logn + m)，删除操作优化为 O(logn)。而时间开销最大的回溯算法，通过对长度为 h 的并发操作序列的检索，把第一次回溯的算法复杂度降低为 O(d * h + h * log(n))，d 和 h 的数量通常比较小。第二次回溯的过程仅仅需要将第一次回溯过程种变更的有效信息位恢复，然后处理新节点的标记位。因此第二次回溯的复杂度为 O(h+d)。综上所述。插入操作的改进复杂度为 O(d * h + h * log(n) + logn + m + h + d)，删除操作的改进复杂度为 O(d * h + h * log(n) + logn + h + d)。

在我们的改进算法中，由于注意到两次回溯之后其实并不改变节点的有效位情况，因此我们直接省略了回溯的过程，因此只需要考虑插入和删除两个操作所引起的复杂度开销。 回顾一下改进算法里的插入操作，首先找到操作的逻辑前驱节点，然后在逻辑前驱节点的后续并发操作节点序列中找出实际插入位置，最后完成节点的插入。删除操作和更新操作比较简单，直接找到目标位置，对节点进行相应的更

新即可。我们可以发现算法的复杂度开销主要在于找到目标节点上。由于每个节点都有唯一的标识符 identifier，我们可以通过多种方式来组织文档结构从而提高查找的效率。比较稳定的方式是通过红黑树等二叉搜索树结构来组织文档结构，我们以红黑树为例，对改进算法的复杂度进行分析，可知红黑树的查找复杂度和插入复杂度都为 O(logn)[52]。因此我们可以得知如果通过红黑树来组织节点，找到逻辑前驱节点的算法复杂度为 O(logn)，在并发操作节点序列中寻找实际插入节点复杂度为 O(m*logn),因为每个节点只保存下一个节点的标识符,因为查找下一个节点的过程同样需要 O(logn)，保存节点的复杂度则为 O(logn)。删除和更新操作只需要 O(log n) 找到目标节点进行相应信息的跟新即可，由于我们并不在每个节点下附加所有对于该节点的操作，因此也没有额外的复杂度开销。综上所述，插入操作的时间复杂度为 O(logn + m*logn + logn), 删除以及更新操作的时间复杂度均为 O(log n)。由于逻辑前驱节点的后继无效节点序列长度 m 是由并发操作影响的，所以 h >= m。因此可以得出在一定情况下基于红黑树的改进算法的复杂度开销要低于基于红黑树改进的传统地址空间转换算法。

然而，我们同样可以通过散列表的方式来组织文档结构，通过良好定义的散列表，查找和插入的复杂度均为 O(1)[52]。因此改进算法的插入和删除操作的时间复杂度分别降低为 O(1 + m*1 + 1)和 O(1)，时间复杂度明显大幅度的下降。但由于该算法性能依赖于散列表的选择，实际的运行效率可能不稳定。四种方法的复杂度对比情况如 5-3 所示。

表 5-3：四种地址空间转换算法的复杂度对比

算法	插入操作	删除操作（更新操作）
传统方法	O(n*d + m + d + n)	O(n*d +1+ n)
传统方法的红黑树改进	O(d * h + h * log(n) + logn + m + h + d)	O(d * h + h * log(n) + logn + h + d)
改进方法的红黑树实现	O(logn + m*logn + logn)	O(log n)
改进方法的散列表实现	O(1 + m*1 + 1)	O(1)

5.5　原型系统实现

基于改进的地址空间转换算法我们实现了协同旅游路线规划的原型系统 Clip。我们后续也会通过对该原型系统以及基线系统的性能测试来验证改进算法的优势。本小节将着重介绍原型系统的模块构成以及设计理念。

如图 5-4 所示，Clip 主要有两部分组成，服务端和客户端。服务端比较简单，主要进行客户端发送过来的操作的 SRN 分配以及操作的广播，将操作发送给所有协同旅游任务的参与者。客户端相对来说比较复杂，主要有用户界面（user interface）。定义了用户的基本操作（AppendPOI, DeletePOI, Connect, Disconnect, Update）。接下来是服务层，服务层对地址中间转换算法所支持的操作（插入，删除以及更新操作）进行相应的封装，并将封装好的服务提供给用户界面调用。接下来的两部分分别是是 AST 引擎（AST-ENGINE）模块以及共享工作空间（shared workspace）模块。服务层封装好的一致性算法运行在 AST 引擎上，得到正确的文档结果，然后将文档结果反映到协同旅游规划任务的共享工作空间当中展示给用户。此外，AST 引擎可以接收远程操作，并且将本地操作发送到 Clip 服务器上进行广播。

不难发现，基于此结构下的系统可以很容易的进行模块的替换，无论是采用改进后的地址空间转换算法，还是传统的地址空间转换算法，只需要将 AST 引擎部分进行替换即可，同样，如果我们需要将地址空间转换算法应用到其他应用场景中，只需要对用户界面，服务层和共享工作空间进行相应的改变即可。接下来我们将会介绍改进后的 AST 引擎的工作模块。

如图 5-5 所示，改进后的 AST 引起主要由三部分组成，操作历史缓存区（history buffer），通信管道（pipe）以及数据模型（data model）。服务端接收用户界面的操作并将其转换成对应的 AST 引擎支持的操作，放入本地操作历史缓存区，通信管道接收服务器广播的操作，放入远程操作历史缓存区。操作历史缓存区将自己的所有待执行操作依次交给数据模型去运行一致性维护核心算法，确保操作正确的执行，执行后的结果反映给共享工作空间，渲染成所相应的形式。操作历史缓存区和通信管道是通信协议的重要组成部分，通信管道只要支持服务器推送的数据传输模式即可，一致性算法的主要运行在数据模型上。

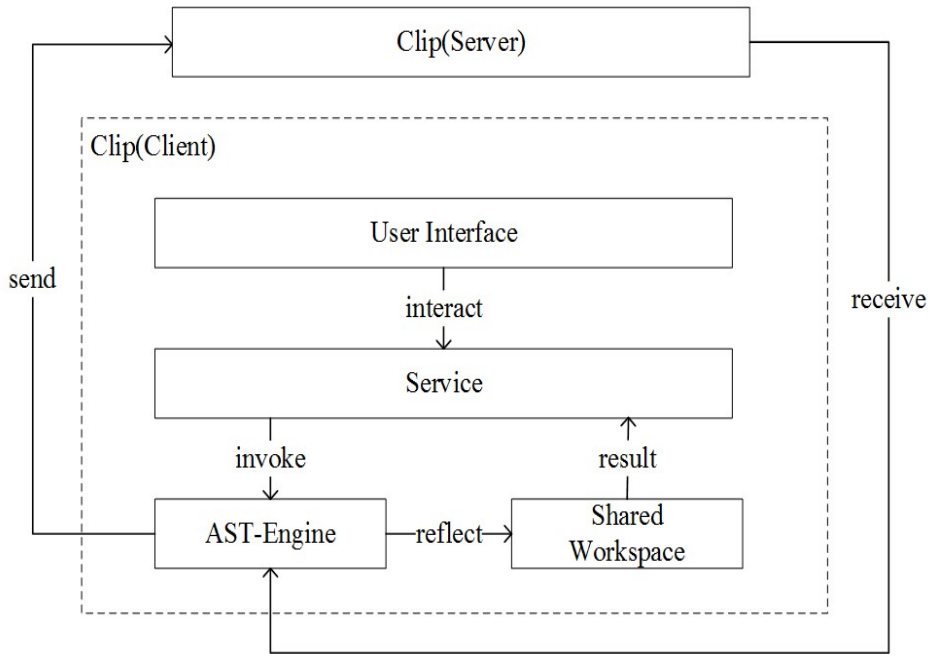


图 5-4：Clip 的模块构成

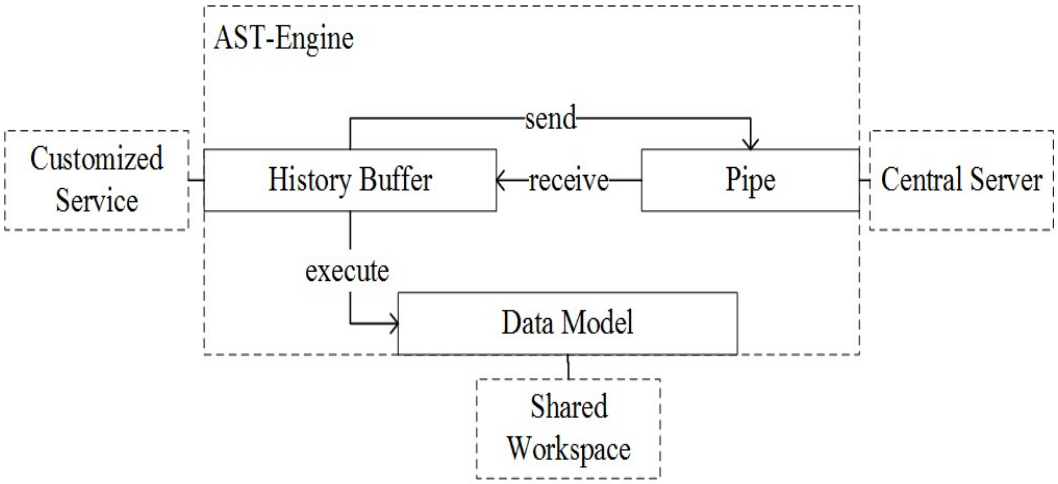


图 5-5：AST 引擎的模块构成

5.6 实验评估

5.6.1 实验设计

基于原型系统 Clip 我们设计了 AST-Engine 的 2 对组件来进行实验，分别是传统地址空间转换算法的数据模型组件以及改进地址空间转换算法的数据模型组件，异步工作模式的通信管道以及同步工作模式的通信管道。Clip 服务器端的后台系统

主要由 J2EE 实现，通过 WebSocket 来实现网络通信，客户端的系统主要由 JavaScript 实现，采用 B/S 结构来组织服务。主要从以下三个方面来评估 Clip 的系统性能。

1：本地操作的响应速度以及传统和改进的地址空间转换算法的性能差异。

2：同步工作模式和异步工作模式对于服务器吞吐量的影响。

3：同步工作模式和异步工作模式对于操作传输时延的影响。

AST-Engine 运行在协同旅游路线规划所定制的服务组件和共享工作空间组件之中。因此，对于用户来说，他们的输入输出效果一致，只是实际运行的算法效率和工作模式不同。协同旅游路线的服务端发布在运行在虚拟机中的 Apach Tomcat 上，该虚拟机配置了 X5660 2.8GHz 处理器，内存 4G，安装的操作系统为 Windows server 2008 R2。客户端的处理器是 I5-2400 3.1GHZ，主存 8G，运行的操作系统是 windows 10。评估实验是通过模拟用户操作来进行的。部分实验通过 selenium[53]随机产生的用户的合法基本操作来完成对于共享工作空间的编辑。每个在共享空间中的操作被服务层捕获，并且转化成调用相应的封装好的 AST-Engine 的操作，反映到数据模型上，然后通过定义好的通信管道发送给中心服务器。所有的操作由中心服务器进行统一的调度和分配，我们采用如下的方法测试一致性算法的运行效率：首先依据一定的原则产生大量的随机操作，然后操作会按照定义的结构流向数据模型和通过通信管道进入服务器被处理和转发。我们将会对这个工作流程中所关心的算法效率，服务器吞吐率以及操作的传播时延进行评测和分析。

5.6.2　实验步骤和结果分析

改进地址空间转换算法和传统地址空间转换算法的性能比较

由于在共享工作空间的文档规模较小的时候，无论是传统的地址空间回溯算法还是改进的地址空间回溯算法。单个操作的响应时间都太小以至于无法衡量。因此，为了使得对算法的性能有比较清晰的认识，我们定义 TRT（total response time，总共响应时间）。TRT(n)表示执行 n 个操作一共所需要的时间，来替代单个操作所需响应时间。$\mathrm{TRT(n)} = \sum_{i=1}^{n} op_i$，其中$op_i$表示在 i-1 的文档规模下，单独执行第 i 个操作所需等待的操作响应时间。由于一致性维护算法在本地副本运行不依赖网络，因此操作完全避免了网络延迟可能带来的影响，操作依次执行不需要额外的等待时间。因此 TRT(n)实际上就是本地副本从空文档开始依次执行 n 个操作所需等待的响应时间之和，如图 5-6 所示，红色的线是传统地址空间转换算法操作个数 n 和 TRT(n)的变化关系，蓝色的线是改进后的地址空间转换算法的操作个数 n 和 TRT(n)的变化关系。正如表 5-3 的时间复杂度的理论分析所表明的一样。相比于对于文档规模的变

化较为敏感的传统的地址空间转换算法，文档规模变化对于改进后的地址空间转换算法的操作响应时间的影响很小，2000 个操作在几十毫秒以内就能全部执行完毕，算法的性能提高非常明显，接下来的测试中发现 4 秒内能执行完 30000 个操作。

同步工作模式和异步工作模式对于服务器性能的影响

首先我们给出同步工作模式和异步工作模式的定义：

同步工作模式：按照事先设置好的请求频率，根据上一次同步操作的截止时间戳（lastUpdateSRN）的信息，客户端主动发起对于其他用户在上一次截止时间戳到目前为止发送到服务器的操作的拉取。

异步工作模式:对于所有参与协作的用户而言，每当他们有一个操作被发送到了服务器上，服务器会立刻将该操作广播，推送给所有参与用户，而无需客户端主动拉取相应的未同步操作。

最初的地址空间转换算法所依赖的向量时间戳是支持两种工作模式的[19]，不过耗费的空间情况对于参与的用户数目大小比较敏感，不太适用于 web2.0 的网络应用，而后面发展的标量时间戳尽管消除了对于用户数目敏感的不足之处[20],但是他支持的是同步的工作模式。本文提出的服务器的工作算法支持两种工作模式，而且采用标量时间戳，对于用户的数目也不敏感。由于同步工作模式和异步工作模式对于系统的性能影响各有特点，接下来我们将会模拟这样的一个场景，从这个场景中分析不同工作模式的性能特点。

服务器处理用户请求的效率

服务器需要对用户发送过来的请求所附带的时间戳中空缺的 SRN 进行分配，这也是服务器的主要工作部分之一，我们将服务器对操作处理所花费的时间，定义为操作的处理时延（processing time，以下简称为 PT）。我们对 PT 进行了相应的测试，如图 5-7 所示，服务器处理操作所需的时间非常少，从图中不难发现 10000 个操作的处理时间为 4000 毫秒，平均下来一个操作需要 0.4 毫秒。相对于网络的延迟来说基本忽略不计。而且处理时间与操作的数目呈线性上升趋势。

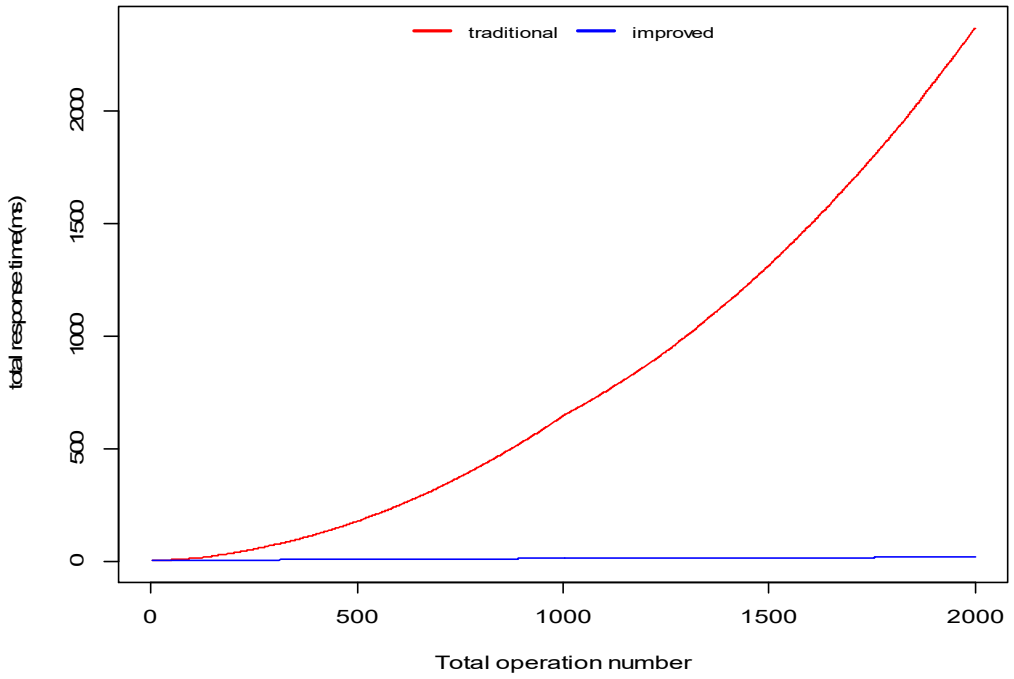


图 5-6：改进算法和传统算法对于文档规模变化的影响关系以及操作响应速度

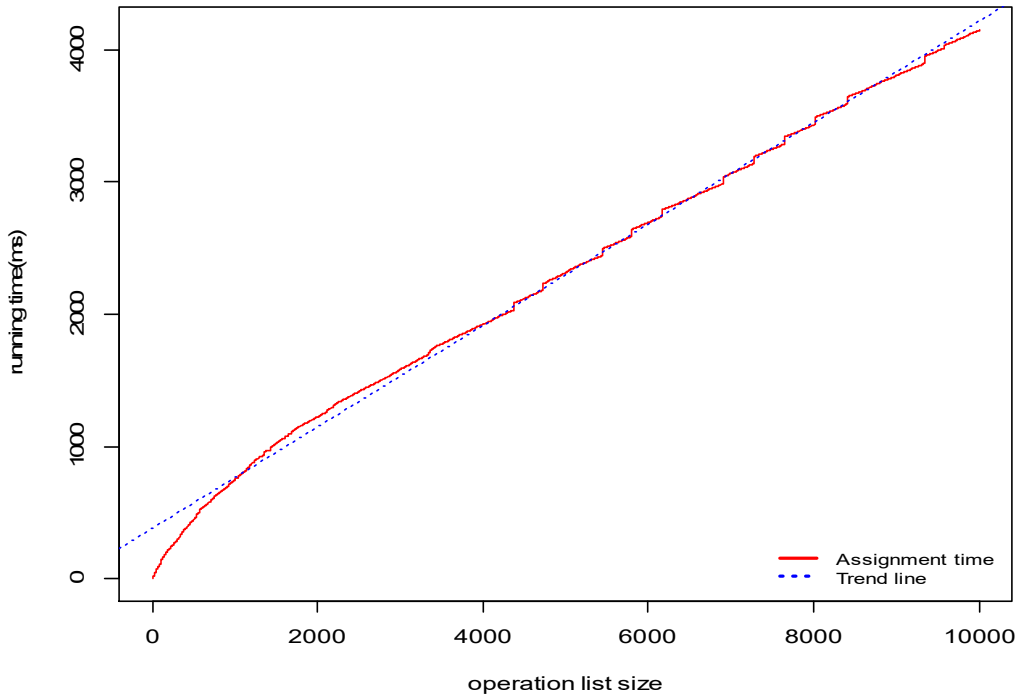


图 5-7：服务器分配 SRN 的算法运行时间和操作数目的关系

操作传输时延的影响

我们将考察同步和异步工作对于操作传输时延的影响，我们将随机产生 200 个操作，每个操作按照一定的频率对服务器发起服务请求，进行数据的同步和交换。首先我们定义操作的传输时延为一个操作从产生到被另外一个客户端接收所经过的时间。对于同步工作模式而言，一次请求包括将从上一次请求到本次请求时间区间本地产生的操作发送到服务器端，并且从服务器端拉取其他待同步操作这两个部分。对于异步工作模式而言，一次请求只需要将本地待发送的操作发送到服务器上即可，然后这些操作会再被服务器处理完毕之后进行异步的广播给所有用户。我们设定请求产生的频率分别为 400 毫秒、600 毫秒、800 毫秒和 1000 毫秒，在这些请求频率下考察不同工作模式对于传输时延的影响。从工作模式的特点我们可以看出，同步工作模式的传输时延（synchronization time delay，以下简称为 STD）主要由这几部分组成，客户端到服务器的网络传输延迟（network latency,以下简称为 NL），服务器对于操作的处理时延 PT,等待获取时间（waiting time，以下简称为 WT）组成，最后再从服务器发送给用户所需经历的网络延迟 NL.具体计算公式如下：

$$\mathrm{STD} = \mathrm{NL} + \mathrm{PT} + \mathrm{WT} + \ \mathrm{NL}$$

其中 NL 和 PT 的时间跟主机性能和网络环境有关，上面也给出了特定主机性能下对于 PT 的影响。WT 跟请求频率（request frequency，以下简称为 RF）有关，并且满足$0 \le \mathrm{WT} \le \mathrm{RF}$。异步工作模式的传输时延（asynchronization time delay，以下简称为 ATD）由于避免了 WT，所以 ATD 的计算公式为：

$$\mathrm{ATD} = \mathrm{NL} + \mathrm{PT} + \ \mathrm{NL}$$

图 5-8 表示 200 个操作每隔相应的 RF 发出所造成的传输时延（propagation delay）的变化情况。由于 NL 和 PT 的相对与 RF 来说几乎可以忽略不计。因此对于 STD 和 ATD 而言，主要影响传输时延的因素只剩下 WT，异步工作模式不受 WT 影响，因此跟请求频率无关，远远的低于同步工作模式下的传输时延，而传输时延接近请求间隔。从这个实验可以看出异步的工作方式下操作频率对于传输时延的影响几乎没有，而且没有额外的时间开销，工作性能更加优越。

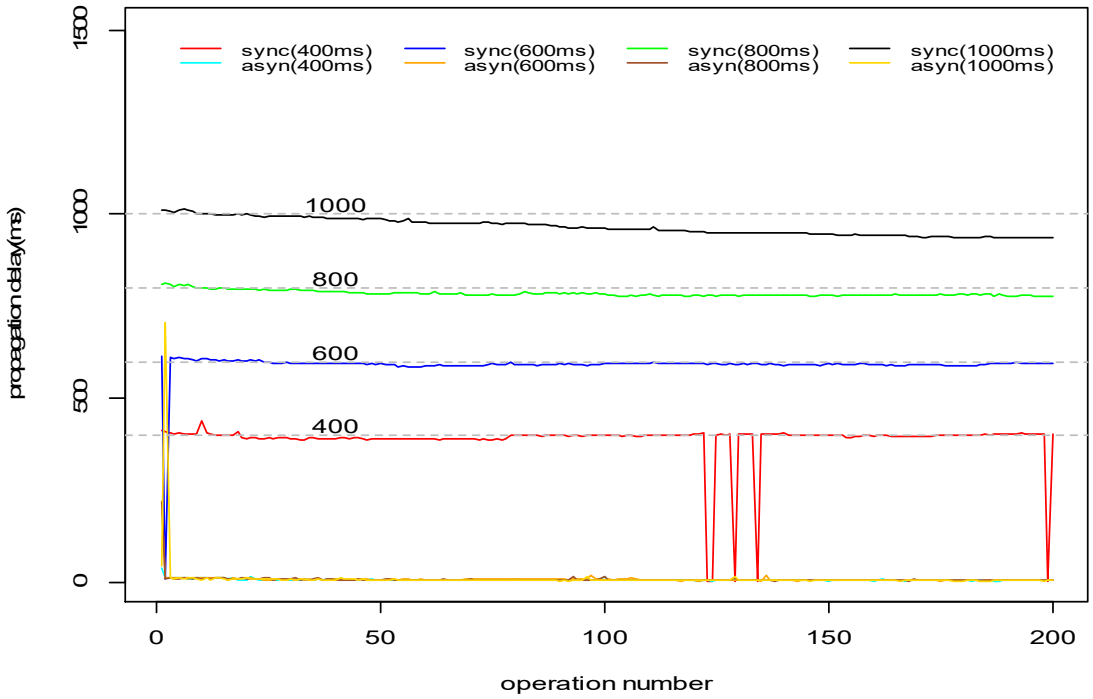


图 5-8：操作的传输时延和请求频率的关系

服务器吞吐量

为了考察高频操作下的服务器性能情况。我们定义服务器性能指标单位为每秒处理完毕的操作个数（request per second,以下简称为 RPS），表示单位时间内能够处理的操作数目，我们设定请求频率均为为 10 毫秒/次，同时有 5、8、10 和 12 个用户参与，因此理论上每秒到达服务器的操作为 500 个（1000/10*5）、800 个（1000/10*8）、1000（1000/10*10）以及 1200（1000/10*12），每个用户一共发送 1000 个操作。结果如图 5-9 所示。一开始吞吐量还未稳定，操作还没有稳定到达服务器端，因此服务器端的 RPS 值较低，随着时间的推移以及来自客户端请求数目的增加。服务器端的 RPS 趋于稳定，接近理论能够到达的最大 RPS，但是随着用户数目的上升，并发请求的增多，RPS 变得越来越不稳定。因为并发量的上升对服务器资源造成更大的消耗（线程切换和调度，共享资源的抢占）会造成系统的不稳定，实验环境下系统所能够稳定工作的最大 RPS 在 1000~1200 左右。由于过度消耗主机资源，并发量过高会严重损害服务器的性能，导致各项指标急剧下降。

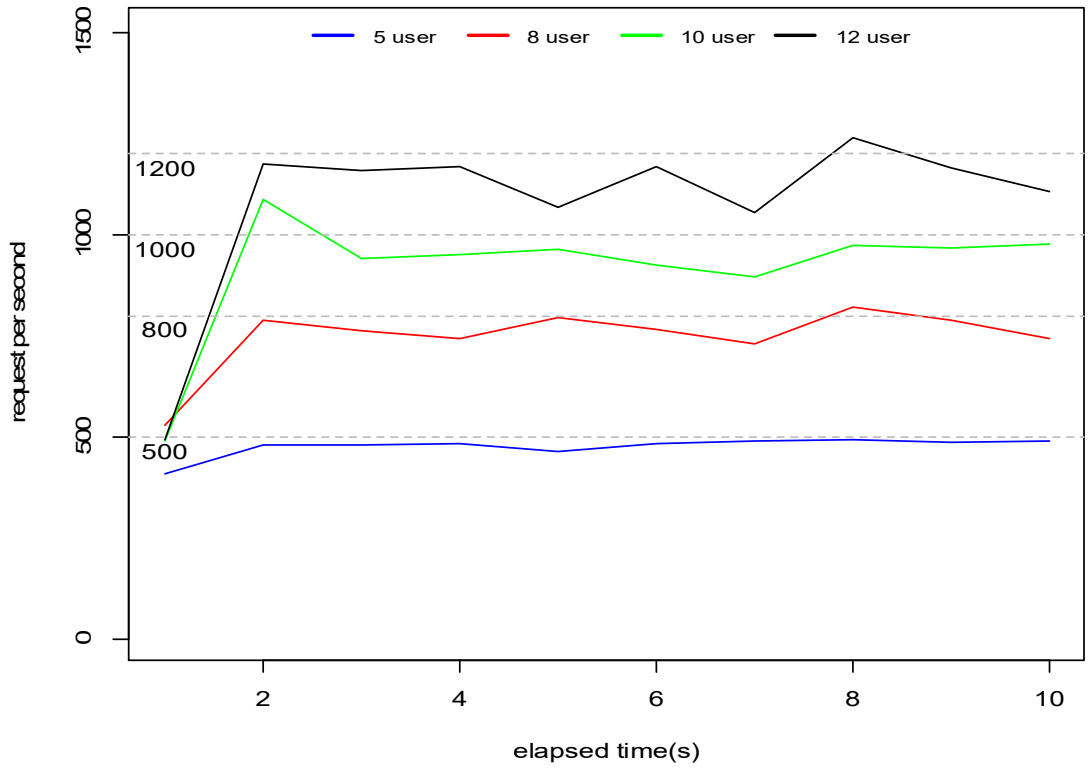


图 5-9：服务器吞吐量跟请求频率和用户数目的关系

5.7 小结

本章主要提出了地址空间转换算法的改进算法，包括新的数据结构，一致性维护算法。然后基于改进的算法给出了正确性的证明，证明了在新的通信协议的基础上一致性算法能够满足副本的一致性。此外，从算法层面对现存的集中地址空间转换算法进行复杂度上的分析，给出了复杂度的对比表格，从理论上证明了改进算法的性能优越性。然后，提出了 Clip 的系统架构，给出了模块划分方法。便于实时协同编辑算法的推广和应用。最后在 Clip 架构上实现了协同旅游路线规划系统，实现了跟数据模型和通信协议相关的两对组件，并且在这些组件上设计了相应的试验并且进行了相应的试验，通过对实验结果的展示，分析。证明了改进后的地址空间转换算法强大的性能优势。然后对服务器的性能进行了测试，并且对比同步和异步两种工作模式进行，证明了该方法能够适用于 web2.0 环境下的基于旅游路线图的大规模协作应用。

第六章　总结和展望

6.1　总结

随着经济水平的提高以及交互式网络应用的日益发展，越来越多的用户加入在线快速协作任务之中，享受群体智慧和互联网技术的双重便利。针对各种协作场景的应用也随之出现，比如众包平台，问答社区，在线办公系统以及维基百科编纂等等，并且成为一种新的趋势。然而同样带来了相应的技术问题和性能挑战。协作所带来的操作合并开销与信息交流开销，都制约了协作的快速准确的推进。此外，现有的协作框架所支持的数据模型和操作模型也非常单调，无法满足用户对于新的交互模式日益增长的需求。我们针对团体旅游路线规划系统这一在线实时协作应用给出了相应的一致性操作模型和算法，为协同旅游路线规划任务提供实时的交互体验。这在协作系统中具有重要的意义，因为传统的协作往往代价较高，难以快速的完成一个特定任务。我们针对协同旅游路线规划这一在线协作系统场景给出了完整的解决方案。这一解决方案能够适用于现在 web2.0 应用流行的集中式架构，为大规模多用户的快速实时协作任务提供有效的支持。总的来说，本文的研究内容以及贡献主要包括三个方面，具体如下。

(1) 通过研究旅游路线规划任务的国内外研究趋势，结合团体旅游体现出来的特点和需求，对团体旅游路线规划任务进行相应的建模。抽象出相应的数据模型和操作模型，从而首次将传统的地址空间转换技术应用于协同旅游规划任务的场景中，将计算机辅助技术以及群体智慧结合在了一起。并发操作引起的冲突能够被快速的自动消除，提供快速准确的本地响应。

(2) 提出一个支持异步消息推送和同步消息拉取的操作通信协议。不同于传统的同步信息拉取机制或者依赖向量时间戳的同步机制。该协议采用标量时间戳，消除了向量时间戳所带来的由于用户数目变化所带来的额外开销，并且支持异步消息推送模型和同步消息拉取模型两种工作模型。基于该网络通信协议能够识别并发操作和因果操作，辅助基于操作关系判定的一致性算法的进行，能够适应用户人数动态变化的大规模协作应用，拓宽了地址空间算法支持的网络通信协议。两种工作模式各有自己的性能特点，通过实验给出了相关说明。

(3) 提出了基于标识符改进的地址空间转换算法。改进算法利用标识符来定位节点，消除了传统的地址空间转换算法定位节点依赖回溯的弊端，克服了由于

回溯所造成的性能开销，完全摆脱了两次回溯的过程。算法效率大大提高，本地响应速度因此大大提高。

6.2 展望

结合本文现有的工作可以发现，在快速协作交互应用中，依然有很多富有挑战亟待解决的问题值得深入研究，需要进一步的探索。

1. 基于新的通信协议和改进的地址空间转换算法设计出相应的垃圾回收机制。尽管改进后的算法效率有了很大的提高，但是由于算法的运行依然会导致无效节点被保留下来，久而久之会对系统的性能有一定的影响，合适的回收机制能够保持系统的性能稳定性，而改进后的算法对于垃圾回收机制有了新的要求，值得进一步的探索。
2. 服务器对于客户端发送过来的操作信息的管理和分发的研究和探索。由于提出的通信协议需要在服务器上对操作进行全局的序列化，因此服务器不能并发的处理操作消息，处理的效率不高，服务器性能收到影响。此外，我们可以从试验中发现网络 IO 也是制约系统性能的一个重要因素。如何将操作进行合并和压缩，减少网络 IO 的调用，都值得进一步的研究从而提高系统的性能。
3. 团体旅游路线规划中或者其他一致性维护算法所支持的应用中的操作意愿的变化所引起的问题也需要进一步研究。在一致性维护工作中最难以定义和解决的无疑就是与操作意愿相关的问题了。目前的一致性维护研究中还没有给出对于意愿动态变化的场景中的一致性维护算法的描述和正确性证明，这需要对于操作意愿进一步的研究和对意愿和操作的关系更深一步的认识。
4. Clip 对于旅游规划方式的贡献程度的进一步探明。本文提出了一种新的基于地址空间转换技术的协同旅游规划方式，然而缺乏相应的真实数据和精心设计的实验去证明这种规划方式的成功率和正确率等等，可以设计进一步的用户实验或者场景模拟去进行案例分析以及效果分析，从而衡量对于旅游规划方式的具体贡献程度。

参 考 文 献

[1] China Outbound Tourism Statistics in 2015[EB/OL]. [2017/1/2]. https://www.travelchinaguide.com/tourism/2015statistics/outbound.htm.
[2] Lew A, McKercher B. Modeling Tourist Movements: A Local Destination Analysis[J]. Annals of Tourism Research, 2006,33(2):403-423.
[3] Xiang Z, Wang D, O'Leary J T, et al. Adapting to the internet: trends in travelers' use of the web for trip planning[J]. Journal of Travel Research, 2014:75406077.
[4] Arif A S M, Du J T, Lee I. Exploring tourists' collaborative web search: implications for system design, 2013[C]. Australian Computer Society, Inc..
[5] TouristEye | Plan your getaways and trips with our web and mobile apps[EB/OL]. [2017/1/2]. http://www.touristeye.com/.
[6] Travel Itinerary Software & Trip Planner Platforms | Travefy[EB/OL]. [2017/1/2]. https://travefy.com/pro.
[7] Wikitravel - The Free Travel Guide[EB/OL]. [2017/1/2]. http://wikitravel.org/en/Main_Page.
[8] Tsiligirides T. Heuristic methods applied to orienteering[J]. Journal of the Operational Research Society, 1984:797-809.
[9] Mysore A S, Yaligar V S, Ibarra I A, et al. Investigating the" wisdom of crowds" at scale, 2015[C].
[10] Zhang H, Law E, Miller R, et al. Human computation tasks with global constraints, 2012[C]. ACM.
[11] Saito Y, Shapiro M. Optimistic replication[J]. ACM Computing Surveys (CSUR), 2005,37(1):42-81.
[12] Ellis C A, Gibbs S J. Concurrency control in groupware systems, 1989[C]. ACM.
[13] 邵斌. 高效的操作转换一致性维护方法研究[D]. 复旦大学, 2010.
[14] Sun C, Others. Dependency-conflict detection in real-time collaborative 3D design systems, 2013[C]. ACM.
[15] Sun C, Wen H, Fan H. Operational transformation for orthogonal conflict resolution in real-time collaborative 2d editing systems, 2012[C].
[16] Sun C, Xia S, Sun D, et al. Transparent adaptation of single-user applications for multi-user real-time collaboration.[J]. ACM Transactions on Computer-Human Interaction, 2006,13(4):531-582.
[17] Xu Y, Sun C, Li M. Achieving convergence in operational transformation: conditions, mechanisms and systems, 2014[C].
[18] Ahmed-Nacer M, Ignat C L, Oster G, et al. Evaluating CRDTs for real-time document editing[J]. DocEng 2011 - Proceedings of the 2011 ACM Symposium on Document Engineering, 2011:103-112.
[19] Gu N, Yang J, Zhang Q. Consistency maintenance based on the mark & retrace technique in groupware systems, 2005[C]. ACM.
[20] Xia H, Lu T, Shao B, et al. A Partial Replication Approach for Anywhere Anytime Mobile Commenting: CSCW '14, New York, NY, USA, 2014[C].
[21] Yang J, Wang H, Gu N, et al. Lock-free consistency control for web 2.0 applications, 2008[C]. ACM.
[22] Li D, Li R. An Admissibility-Based Operational Transformation Framework for Col-

laborative Editing Systems[J]. Computer Supported Cooperative Work (CSCW), 2010,19(1):1-43.
[23] Preguica N, Marques J M, Shapiro M, et al. A commutative replicated data type for cooperative editing, 2009[C]. IEEE.
[24] Sun C, Ellis C. Operational transformation in real-time group editors: issues, algorithms, and achievements, 1998[C]. ACM.
[25] Nichols D A, Curtis P, Dixon M, et al. High-latency, low-bandwidth windowing in the Jupiter collaboration system, 1995[C]. ACM.
[26] Shao B, Li D, Lu T, et al. An operational transformation based synchronization protocol for web 2.0 applications, 2011[C]. ACM.
[27] Tasgetiren M F, Smith A E. A genetic algorithm for the orienteering problem, 2000[C]. IEEE.
[28] Wang Q, Sun X, Golden B L, et al. Using artificial neural networks to solve the orienteering problem[J]. Annals of Operations Research, 1995,61(1):111-120.
[29] Golden B L, Levy L, Vohra R. The orienteering problem[J]. Naval research logistics, 1987,34(3):307-318.
[30] Bolzoni P, Helmer S, Wellenzohn K, et al. Efficient itinerary planning with category constraints, 2014[C]. ACM.
[31] Lu E H, Lin C, Tseng V S. Trip-mine: An efficient trip planning approach with travel time constraints, 2011[C]. IEEE.
[32] Brilhante I, Macedo J A, Nardini F M, et al. Where shall we go today?: planning touristic tours with tripbuilder, 2013[C]. ACM.
[33] Hsieh H, Li C, Lin S. Measuring and recommending time-sensitive routes from location-based data[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2014,5(3):45.
[34] Yahi A, Chassang A, Raynaud L, et al. Aurigo: An Interactive Tour Planner for Personalized Itineraries, 2015[C]. ACM.
[35] Lathia N, Capra L. How smart is your smartcard?: measuring travel behaviours, perceptions, and incentives, 2011[C]. ACM.
[36] Munar A M, Jacobsen J K S. Motivations for sharing tourism experiences through social media[J]. Tourism management, 2014,43:46-54.
[37] Lian D, Xie X, Zheng V W, et al. CEPR: A collaborative exploration and periodically returning model for location prediction[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2015,6(1):8.
[38] Muntean C I, Nardini F M, Silvestri F, et al. On Learning Prediction Models for Tourists Paths[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2015,7(1):8.
[39] Rafidi J. Real-time trip planning with the crowd, 2013[C]. ACM.
[40] Aizenbud-Reshef N, Barger A, Guy I, et al. Bon voyage: social travel planning in the enterprise, 2012[C]. ACM.
[41] Sun C, Jia X, Zhang Y, et al. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems[J]. ACM Transactions on Computer-Human Interaction (TOCHI), 1998,5(1):63-108.
[42] Shen H, Sun C. Flexible notification for collaborative systems, 2002[C]. ACM.
[43] Sun D, Sun C. Context-based operational transformation in distributed collaborative editing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2009,20(10):1454-1470.
[44] Oster G E R, Urso P, Molli P, et al. Data consistency for P2P collaborative editing,

2006[C]. ACM.
[45] Roh H, Jeon M, Kim J, et al. Replicated abstract data types: Building blocks for collaborative applications[J]. Journal of Parallel and Distributed Computing, 2011,71(3):354-368.
[46] Weiss S E P, Urso P, Molli P. Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks, 2009[C]. IEEE.
[47] Andr E L, Martin S E P, Oster G E R, et al. Supporting adaptable granularity of changes for massive-scale collaborative editing, 2013[C]. IEEE.
[48] Palmer C R, Cormack G V. Operation transforms for a distributed shared spreadsheet, 1998[C]. ACM.
[49] Davis A H, Sun C, Lu J. Generalizing operational transformation to the standard general markup language, 2002[C]. ACM.
[50] Sun D, Xia S, Sun C, et al. Operational transformation for collaborative word processing, 2004[C]. ACM.
[51] Pohja M. Server push with instant messaging, 2009[C]. ACM.
[52] Cormen T H. Introduction to algorithms[M]. MIT press, 2009.

发表论文和科研情况说明

攻读硕士期间发表的学术论文

[1] Yang D, Lu T, Xia H, et al. Making itinerary planning collaborative: An AST-based approach[C] //IEEE, International Conference on Computer Supported Cooperative Work in Design, IEEE, 2016: 257-262.

[2] Zhang, P., Gu, H., Gartrell, M., Lu, T., Yang, D., Ding, X., & Gu, N. (2016). Group-based Latent Dirichlet Allocation (Group-LDA): Effective audience detection for books in online social media. Knowledge-Based Systems, 105, 134-146.

攻读硕士期间参与的科研项目

[1] 移动协同工作中基于AST的一致性维护技术研究, 国家自然科学基金项目, 项目编号61272533, 2014.09-- 2016.5.

[2] 上海市残疾人辅助器具资源中心项目：重症残疾人无障碍自理智能化控制与康复提醒系统, 2014.09-- 2015.12.

致　谢

时光荏苒，三年的复旦大学研究生生活终于快要来到终章。在这三年的寒窗岁月里，遇见了许许多多的人，经历了形形色色的事。这一切都历练了我，使我增加了阅历、丰富了经验、提高了能力，提笔至此，回想起这三年的岁月不仅感慨万千。在论文即将划上句话的时刻，请允许我对你们献上崇高的敬意！

首先，非常感谢我的导师卢暾教授。卢老师在这三年里对我的深切期待和一直以来的关怀，使我受益良多。在科研学习上，卢老师严谨考究，对于理论知识和动手能力的结合十分重视，这种精神深深的影响着我，大大提高了我的逻辑能力和代码水平。此外，卢老师勤奋认真的精神也让我受益良多，卢老师经常对我的研究方向和研究内容给出一些富有深远影响的建议和提示，让我在科研之路上能够走的更加从容。培养了我踏实认真的学习态度，并锻炼了自主学习的能力。在生活中，卢老师对于我们也十分的关怀，积极的帮助我们解决生活上的各种困难，同时以身作则，我从卢老师的言行中学习到了许多做人的知识和准则，指引我今后的人生道路。

其次，实验室的老师和同学们也给我提供了非常大的帮助，我怀着崇高的敬意对他们表示我深深的感谢。感谢对顾宁老师和丁向华老师对于我论文的指导意见，没有他们的对于论文细节上的一些帮助，这篇论文的完成将会更加的艰难。非常感谢刘铁江老师对于我们科研和工作上的督促和帮助。非常感谢远在微软的邵斌，夏欢欢师兄对于论文思路和方法的启发和帮助。在我对于理论和认知出现偏差的时候，谢谢你们及时纠正我，并且提供正确的解决思路，帮助我顺利的完成论文。感谢许家华、吴娥英、严鑫、罗东亮和程沛五位同学伴我三年同行，同时感谢实验室的兄弟姐妹们共同营造的温暖和谐的学习氛围，每次在学习或者研究累了的时候，想起你们又充满了干劲。

我还感谢我的父母和女朋友，谢谢你们忍受我心情低落时候的抱怨，并一直给我鼓励，让我顽强走到了现在。你们始终以润物细无声的爱，滋润着我，伴随着我成长。其次，亲人和朋友对我的信任和支持，一直是我坚强的后盾，让我昂扬向前。

最后，感谢百忙之中抽空参与评阅和答辩的老师们，感谢你们的指导和建议让我能够将三年来的学习和研究凝结为这份沉甸甸的答卷。略书不能悉吾意，衷心的祝福大家明天会更好！

复旦大学
学位论文独创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文中除特别标注的内容外，不包含任何其他个人或机构已经发表或撰写过的研究成果。对本研究做出重要贡献的个人和集体，均已在论文中作了明确的声明并表示了谢意。本声明的法律结果由本人承担。

作者签名：_____ 日期：

复旦大学
学位论文使用授权声明

本人完全了解复旦大学有关收藏和利用博士、硕士学位论文的规定，即：学校有权收藏、使用并向国家有关部门或机构送交论文的印刷本和电子版本；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、缩印或其它复制手段保存论文。涉密学位论文在解密后遵守此规定。

作者签名：_____ 导师签名：_____ 日期：