学校代码： 10246
学　　号：15210240030

硕 士 学 位 论 文

（学术学位）

地址空间转换算法的优化研究

Research on Optimization of Address Space Transformation Algorithm

院　　系：　计算机科学技术学院
专业学位类别（领域）：　计算机软件与理论
姓　　名：　张健豪
指 导 教 师：　卢　暾　副教授
完 成 日 期：　2018 年 3 月 5 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

移动互联网的发展已经影响了人类的生活方式，互联网中的应用变得多元化，接入互联网的终端设备丰富多样。同样，协同应用在人群中的普及度也越来越高，协作编辑应用提供的 Anywhere Anytime 的服务更是方便了人们利用碎片化时间办公的需求。由于使用场景的增多，移动设备的网络状况不稳定，硬件资源有限等问题对协同编辑算法的设计带来了诸多挑战。如何使得协同编辑算法既能够支持各种使用场景，操作执行效率又高效，成为了协同领域研究者们共同需要面对的问题。

在二十多年协同编辑算法的发展过程中诞生了一系列的算法，其中地址空间转换（Address Space Transform，AST）是一种高效并且易于证明其正确性的算法。地址空间转换算法，自 2005 年提出以来被用在 Web2.0、树形文档编辑、协同路径规划等诸多场景下。面对更加广泛的算法应用场景和各式各样的硬件环境，原始算法的设计已经不能满足新场景下的性能要求，因此本文针对地址空间转换算法从字符串和移动端场景下分别提出了两种优化方法，具体的工作与贡献如下：

1. 提出了支持字符串操作的地址空间转换算法（AST-String，ASTS），优化了字符串环境下地址空间转换算法的操作执行性能。在传统协同编辑算法的使用场景下，用户执行操作的对象往往是字符串，现有以单个字符为基础的地址空间转换算法模型需要将字符串拆分，并无法直接支持例如复制-粘贴这样常见的用户操作。ASTS 通过在操作产生时将节点合并，随后根据需要分裂节点的方法加快了算法在字符串环境下的执行效率，有效的降低了内存的使用量，并减少了网络传输量。
2. 提出了基于部分复制技术的地址空间转换算法，解决了移动环境下用户所使用的终端设备存在内存和网络带宽受限的问题。通过在现有的协同编辑系统中应用部分复制技术，用户仅仅需要从服务器下载自己感兴趣的副本至本地即可完成编辑，另外服务器也仅仅向客户端同步必要的操作，前者节省了客户端内存后者节省了网络传输量。部分复制技术的提出为 AST 在更多资源有限的硬件设备上的应用提供了可能。
3. 实现了本文所提出的优化算法并完成实验，对实验结果进行评估与分析。通过将本文提出的以字符串为操作单位的模型和原始单个字符为操作单位的模型相比较，从实验数据中得出在一般字符串操作环境下，本文提出的 ASTS 存在明显的效率优势。另外，针对部分复制技术，本文实现了支持部分复制的地址空间转换算法观察到相比较客户端持有全副本，客户端持有部分副本能够大幅减少客户端操作执行时间，大幅节省客户端内存使用，并且没有增

加过多的服务器内存开销。实验结果体现了部分复制技术在中心化系统中的应用能够有效的利用有限的客户端硬件资源。

关键字：协同编辑；地址空间转换算法；字符串操作；部分复制技术；移动协同应用；计算机支持的协同工作

中图分类号：TP3

Abstract

The development of mobile Internet has influenced the way of human life, both the Internet applications and access to the Internet terminal equipment are diverse. Similarly, the popularity of collaborative applications getting higher and higher, and the collaborative application provides Anywhere Anytime service is more convenient for people to use the fragmentation in office. The design of collaborative editing algorithm have brought many challenges due to the increase of usage scenarios and the instability of network conditions in mobile devices. How to make the collaborative editing algorithm both comprehensive and efficient has become a common problem that researchers need to face.

In last two decades, a series of collaborative editing algorithms have been developed. Address Space Transformation algorithm is an efficient and easy-to-prove algorithm. Address space Transformation algorithm, referred to as AST, has been used in Web2.0, tree document editing, collaborative itinerary planning and many other scenarios since 2005. In a wider range of application scenarios and a variety of hardware devices, the design of the original algorithm can no longer meet the performance requirements. Therefore, this paper presents two optimization methods for AST, the specific work and contributions are as follows:

1. Proposing to support the string operation of the address space transformation algorithm (AST-String, ASTS) which is optimize the algorithm performance in string environment. In traditional usage scenarios of collaborative editing algorithms, the user often performs the operation on a string object. Existing AST model needs to split the string into single characters and can't directly support such common user operations such as copy-and-paste. The ASTS support node with string and split operation. These improvements speed up the algorithm in the string environment, implementation of the efficiency of effectively reducing the memory usage, and reduce the amount of network traffic.
2. Proposing partial replication technology on AST which is turn to make full use of the mobile memory and network bandwidth. By applying partial replication technology to existing collaborative editing systems, users simply download their own interested copy from the server to complete editing, and the server synchronizes only the necessary actions with the client, saving the client memory and also reduce the network traffic. The proposed partial replication technology for AST provides the possibility on the hardware device in more limited resources.

3. Implementing the optimized algorithms on AST and evaluating the experiment result for both above methods. By comparing the proposed model with string operation and the original single character operation, we conclude that there are obvious efficiency advantages of ASTS in string environment. In addition, for the partial replication technology, this paper implements the address space transformation algorithm that supports partial replication. It is observed that holding a partial replica on the client can significantly reduce the operation execution time and save the client memory usage, Moreover, it does not add too much server memory overhead. This fully demonstrates that the application of partial replication technology in a centralized system can effectively utilize limited client hardware resources.

Keywords: Collaborative editing; Address space transformation algorithm; String-wise operation; Partial Replication technology; Mobile collaborative applications; CSCW

Chinese Library Classification: TP3

第一章 绪论

自 1989 年第一个协同编辑算法[1]在 SIGMOD 会议发表以来，许多关于协同编辑算法的工作陆续发表在 CSCW、GROUP 等国际会议中，算法的发展主要集中在两个方向，第一如何提高算法效率[41], [42][43], [44]，第二如何将算法应用到新的领域中[45], [46]。随着各种各样改进方法和新算法的提出，算法时间空间复杂度已经十分出色了，当前性能最优的算法[9], [32]都能够在对数甚至常数时间内完成一次操作。然而，随着算法的应用场景越来越广，算法应用的运行设备也有各自不同的性能特点，需要针对用户实际的使用和设备的性能需求来做特定的算法优化。面对算法模型的越来越复杂功能越来越多，如何在优化效率的同时尽可能使得算法设计简单和易于理解也是一个重要的研究问题。本章将先介绍本文的研究背景，介绍现今移动应用大环境下对算法模型的影响，接着介绍现有协同编辑算法中面临的问题，最后介绍本文的主要工作。

1.1 研究背景和意义

近年来硬件技术飞速提升，手持硬件设备在多方面取得了改革性的进步，以 iPhone 为代表的智能机的产生正式取缔了 PC 机成为互联网的主要接入终端，由此移动互联网的潮流到来了。当前正处在这个潮流的巅峰时期，手机几乎成为了我们生活中必不可少的一部分。另一方面，表、腕带、眼镜这些传统的配件也逐渐智能化，智能设备已经充斥于我们生活中的每一个角落。硬件设备的发展同时也推动了软件的发展，应用的功能变得丰富多样，手机也远远超出了他本应该承担的职责，游戏机，笔记本，导航仪等等丰富的功能都以 APP 的形式体现在一个终端设备上。

实时协同编辑算法作为 CSCW 研究领域中重要的一部分，自 PC 时代就已经产生，1989 年 Ellis 发表了 dOPT 算法[1]也是当前最为流行的 OT 算法[4]的原型。虽然当时算法模型上还存在一些缺陷，却为 CSCW 领域的研究者们打开了一扇新的大门。随后，一批研究者都投入到协同编辑算法的研究，他们所采用的思想不尽相同，可以统称为操作转换。20 多年来，协同编辑算法从当年仅仅用于支持小范围的文本协作，到现在可以支持多终端，跨平台，大规模的协作，协作的对象也不仅仅局限于文字，同时可以支持表格[27]、格式化文本[22]、3D 结构[28]、图形结构[34]等等。

随着研究的不断进展，无论是 OT[1], [2]、AST[32]还是 CRDTs[6]都对于算

法效率和应用场景做出了很大的改进。然而，随着新的环境的产生和新的需求的提出，这样的改进将一直持续下去。另一方面，随着移动平台的兴起大部分的系统都采用了中心化的结构，原本客户端软件实现的功能都以服务的形式在云平台上提供。在这种模式下，应用部署在服务器上，客户端仅仅承担应用中的一部分职责，而其余的则由服务器完成。协同编辑算法也是如此，刚刚提出所使用的 P2P 结构也逐渐被中心化结构所取代，标量时间戳，操作序列同步等技术对于协同编辑算法在移动终端设备上的效率有明显的改进。

针对不同的环境，协同编辑算法的性能要求也不一样，本文讨论的性能优化分为两方面：针对用户操作的性能优化技术与针对移动环境的性能优化技术。用户在实际使用协同编辑应用中会产生连续的操作序列，往往操作对象在文档中的位置也是集中的，对于在相邻位置上连续的插入或删除操作称为字符串操作。如何使得协同编辑算法能够在尽量简洁的情况下支持字符串操作成为当前研究者们主要研究的问题[10][11][14][15]。另外，随着移动平台的普及，使得许多移动应用也运行在移动环境中，例如 Google Doc[47]、One Drive[52]等协作应用分别推出了在移动平台上的 APP，用户利用闲散时间办公也成为了可能。由于移动环境下的网络传输延迟高且不稳定、设备内存小，因此对于移动协同编辑算法提出了更高的挑战，本文的后半部分将会讨论如何采用部分复制技术解决移动协同编辑算法中的性能问题。

1.2 协同编辑算法中的优化问题

协同编辑领域中的优化问题一直是研究者的主要研究方向之一[41][42][43][44]，优化问题往往分为两类，面向算法的优化[41][42]和面向应用（用户）的优化[43]。前者致力于将算法应用到更多更广的领域中，后者则能提高应用与算法的适配程度和用户的体验。由于智能终端设备及其应用生态环境的快速形成，原始协同编辑算法在全新的应用环境与模式下出现了许多不足。以地址空间转换算法为例，就存在以下两大急需解决的问题：

字符串操作环境下的性能问题 地址空间转换自提出已经超过十年了，研究者们将其成功应用到了数种数据结构的协同编辑应用中。然而，地址空间转换算法对于字符串操作一直没有模型层面的支持。在原始算法中，字符串操作是单个操作的集合，模型会将用户的连续操作拆分，并且每个操作拥有单独的时间戳。这样大大降低了算法对于内存和网络带宽的使用，并且降低了算法的响应速度。因此如何从模型层面上支持字符串操作，使其能够更加快速的响应字符串操作同时节省内存和网络带宽的使用是极为迫切的需求。

移动终端下的硬件资源瓶颈 移动设备具有内存小、计算速度慢等特点。在

协同编辑领域中，随着编辑内容所占用内存空间越来越多，内容格式越来越复杂，移动设备客户端的内存使用成为了系统中的一个瓶颈。在传统分布式系统领域中，部分复制技术能够提高客户端资源率，在协同编辑系统中，部分复制技术的使用不仅仅能够解决内存使用问题，还能够提高本地操作的响应速度。因此，如何使地址空间转换算法在算法层面上支持部分复制技术是一项十分有意义的研究工作。

1.3 本文主要工作

针对上节阐述的研究问题，本文在地址空间转换算法的基础上提出了具体的解决方案。地址空间转换算法是协同编辑领域中的经典算法之一，由 Ning Gu 等[41]提出，该算法具有操作执行高效，易于证明等特点，因此本文以其作为优化工作的基础算法。地址空间转换算法在 P2P 结构和中心化结构上都有对应的版本，前者为原始算法，后者则在 Huanhuan Xia 等[45]和 Dayi yang 等[46]中提出，后者是前者的改进并且去除了耗时的地址空间回溯操作。本文对于上述两类算法分别做了算法层面的优化。

本文的研究内容如下：

1. 提出了支持字符串操作的地址空间转换算法模型 ASTS。ASTS 通过将相邻的用户操作合并，传输和存储时时间戳与字符串相对应，从模型层面上支持了字符串操作。ASTS 的提出解决了原本用户常见操作如复制粘贴、查找替换响应速度慢，资源消耗大的问题，同时也大大降低了网络带宽的使用。
2. 提出了中心化地址空间转换算法中的部分复制技术。通过服务端存储地址范围表动态的维护每个客户端所拥有的副本范围，支持了操作能够同时在部分副本与全副本中的正确执行。相对于全复制，部分复制技术有效的减少了算法对于客户端内存的消耗，同时为地址空间转换算法在便携式设备和嵌入式设备上的应用提供了可能。
3. 完成了上述地址空间转换算法的优化方法实验与分析，评估了算法的性能。一方面，从字符串操作环境和单个字符操作环境两个方面分别对 AST 和 ASTS 进行实验分析，验证了 ASTS 能够有效改善算法在字符串环境下的操作执行性能。另一方面，从内存使用，操作响应速度两方面对支持部分复制技术的 AST 和全副本技术的 AST 进行比较，验证了部分复制技术能够有效减少客户端对于内存的使用，并改善客户端的操作执行性能。

1.4 本文组织结构

本文主要分为七个部分：

第一部分为绪论：首先阐述了本文的研究背景和意义，接着介绍了协同编辑算法近年来遇到的问题，接着介绍了本文的研究工作，最后介绍本文的组织结构。

第二部分为相关原理与技术：从传统的协同编辑算法开始介绍，到字符串模型下的相关工作，再到最后部分复制在其他协同应用中的相关工作。

第三部分为支持字符串操作的地址空间转换算法，首先介绍了模型和算法的定义，其后分别介绍了 AST 三大模块（控制算法，回溯算法，操作执行算法）在字符串操作环境下的改进方式，最后介绍了算法优化和 Undo 操作的支持。

第四部分介绍了基于全局唯一标识符的中心结构地址空间转换算法，首先介绍了基础数据结构的定义，其后介绍了 AST 三大模块的对应实现，最后讨论了系统编辑算法中的排序规则。

第五部分介绍了地址空间转换算法中的部分复制技术，首先介绍了部分复制技术下的基础数据结构定义，其后分别介绍了操作同步和副本同步的基本流程，最后介绍了算法的实现和复杂度。

第六部分是实验设计与分析，分别针对上述两种优化方法介绍了实验设计与分析和实验过程和结果分析。

第七部分为总结与展望，首先总结对于本文的研究工作做了总结，其后对于未来可能的研究方向做了介绍，为未来的研究工作提供了思路。

第二章 相关原理与技术

系统编辑算法的提出最初是为了支持多个用户实时编辑同一份文档时，操作能够实时响应并且算法能够自动处理操作之间的冲突。在协同编辑算法的研究过程中研究者们提出了不少优化方法，每种优化方法对应于一种思想，思想通常能够作用于多个不同种类的算法甚至能够作用于跨领域的多个不同算法。本文提出的优化方法也不例外，不仅仅适用于地址空间转换算法，在其他协同编辑算法中也可以适用，国内外也包含了不少这样的相关工作与技术。

本章首先会介绍传统的协同编辑算法，其中重点会介绍 AST[41]。接着，会针对现有的协同编辑算法中支持字符串操作的模型进行介绍。最后，还会介绍分布式系统中其他使用部分复制系统的案例，并且重点介绍 Docx2go **错误!未找到引用源。**、Hydra[45]这两个应用了部分复制技术的协同编辑算法。

2.1 传统协同编辑算法

协同编辑算法根据算法思想的不同大致可以分成三大种类，根据提出时间的前后依次是操作转换算法（Operation Transformation，OT）[1][2][3]、地址空间转换算法（Address Space Transformation，AST）[32]和可交换复制式数据类型（Conflict-free Replicated Data Types，CRDTs）[6]。操作转换技术大多通过维护历史操作，并且根据本地历史操作对远程操作进行转换使其正确的在文档中执行。地址空间转换的思想则相反，他通过记录操作产生时刻的地址空间，远程操作在执行前先进行地址空间回溯，以此确保远程操作的正确执行，其后再将地址空间回溯到原来状态。可交换复制式数据类型主要通过确保操作在全局的排序关系，这样支持了操作可以在各个站点以任意的顺序执行，由于排序规则是全局唯一的因此最后文档也能确保最终一致。

本文提出的优化主要针对地址空间转换算法（AST）[41]。相对于其他两种协同编辑算法，AST 的算法结构清晰易于证明。从提出以来，许多研究者对其做了相关的效率改进和应用场景拓展工作。这些工作主要包括采用因果先序向量来使得地址空间转换算法应用于大型协同编辑应用场景之下[43]，在 Web2.0 的环境下，将地址空间转换算法应用于 XML 树形结构的一致性维护问题上[42]，并且同时支持了简单的事务操作[42]，移动评论树形结构的中心化地址空间转算算法和树形结构的部分复制技术[45]。这种部分复制技术的思想和本文的类似，然而区别在于这种部分复制技术是应用层面的，而本文的则是算法层面的，另外中

心化的地址空间转换算法的提出也为本文的工作打下了必要的基础。

最早应用在协同编辑上的算法为操作转换算法，该名称是一类算法的集合。其中包括 dOPT[1]、adOPT[2]、GOT[3]、GOTO[4]、ABT[5]等。同样，CRDTs 作为最近十年才提出的协同编辑算法集合主要应用在 P2P 的协同编辑场景下，其中具有代表性的包括 WOOT[6]、LOGOOT[7]、TreeDOC[8]和 RGA[9]。

2.2 支持字符串操作的协同编辑算法模型

为了统一定义操作和简化模型定义，最初研究者在设计协同编辑算法模型时都将操作的力度定义为字符。然而，协同编辑算法在发展的过程中不少研究者们发现，单个字符的操作模型并不能高效的处理用户的操作。用户在使用协同编辑操作软件时常常会以字符串为单位，无论是中文还是西文都会以若干个字符来组成一个有意义的词组或单词。采用现代中文输入法输入时，常常以词组或者短语为单位这对于以字符为单位的算法模型来说十分的不自然。

因此，在 OT 和 CRDTs 领域中都有研究者对于算法进行改进，使其支持字符串操作。接下来分别举例介绍：

首先对于 OT 算法，[19]是一项十分重要的工作，文中作者提出了一种全新的操作转换方法来支持字符串操作。同时，此文的另一项重要贡献是提出了一种一致性维护的框架，（Convergence, Causality preservation, and Intention preservation，CCI）模型，在其后关于协同编辑的研究工作[32][38]中被广泛的采用。

另一种支持字符串操作的 OT 算法是（Admissbility-Based Transformation with Strings，ABTS）[26]，他是基于（Admissbility-Based Transformation，ABT）[5]，一种已经证明正确的 OT 算法。在这项工作中，作者对于 OT 特有的 IT（Inclusion Transformation）和 swap 两个算法进行了改进使其能够支持字符串操作。另外，作者介绍了 ABTS 的框架可以用于一系列的 OT 算法使其能够支持字符串操作。作为较早提出的字符串协同编辑模型，ABTS 的通用性和正确性都是毋庸置疑的，唯一不足的是操作执行效率依赖于本地历史操作队列的长度。

另外 CRDTs 领域也有相关研究者做字符串模型相关的工作，如基于 WOOT[6]算法改进的一种方法[10][11]。文章中作者提出了一种 CRDT 方法同时支持字符串操作和选择 Undo 操作，文章中作者也采用了分裂的方式来支持跨越或在节点中间的操作。另外，通过将分裂后的节点通过指针形式连接成一个双向链表，使得模型能够方便的支持 Undo 操作。值得一提的是，这篇文章提出的方法是唯一一个能够同时支持 Undo 和字符串操作的协同编辑模型。作为最近提出的方法之一，在各方面做得近乎完美，然而远程插入操作的执行效率为并发插入操作的平方，还存在可以优化的空间。

关于 CRDTs 领域的研究还包括 LOGOOT-String[12]，一种基于 LOGOOT[7]的字符串操作模型。LOGOOT 是一种简洁易懂的 CRDTs 算法，通过生成全局唯一的标识符然后根据标识符排序。对于这种方法的改进也十分的简洁，虽然作者提出了一些改进手段，可无限增长的标识符依然是这个算法的重要缺陷。

另外，还存在一些和本文提出方法同时期提出的支持字符串操作的方法，他们的算法模型的设计也十分的出色。第一个是关于对 RGA[9]（一种目前最高效的 CRDTs 算法）的改进，作者采用平衡树的形式组织原本线性的文档结构，提出了一种叫做 Block-wise RGA[9]的方法，同样采用节点分裂的方法来处理发生在节点中间的操作。另一种方法也是针对 RGA 算法，叫做 RGASS（RGA Supporting String）[15]。

本节枚举并介绍了当前支持字符串操作的协同编辑算法模型，可以发现对于传统的单字符模型进过一定的改进都能够很好的适配字符串操作。AST 算法也是一种高效的协同编辑算法，并且具有很广的应用范围，在随后的章节中会设计一个针对 AST 的字符串操作支持方案。

2.3 部分复制技术在协同系统中的应用

中心化结构也称作客户端-服务器结构是当前大多数互联网应用采用的系统架构，其中协同编辑应用也不例外。在中心化结构中，客户端和服务器往往拥有不同的资源，在系统中也扮演者不同的角色。算法可以分别针对客户端和服务器的特点做对应的优化，因此也产生了一系列的中心化协同编辑算法。如最初出现的 Jupiter[37]、TIPS[38]到后来的 Hydra[24]，虽然基于不同的算法体系，但他们都按照中心化结构的特征对算法进行了改进。

部分复制技术在中心化结构中是一个常用的优化技术，在分布式系统中存在广泛的应用。原理就是由服务器存储全部副本，客户端根据实际需求向服务器请求部分副本进行编辑修改然后将操作同步到服务器。通过这样的原理能够充分的利用客户端有限的内存，并且也考虑到了服务器的资源相对富裕这个架构特点。另外，对于不同的接入设备通过算法不同的参数调整能够使得其拥有更加出色的用户体验。

部分复制在协同应用中的研究并不多，本小节将以此介绍仅有的几个系统。

Cimibiosys[39]是较早期的一个协同应用，它被设计用来同步不同设备之间的数据。针对移动设备 adhoc 的协作模式，它设计了自己的同步协议。在 Cimibiosys 中每个设备都会存在一个过滤器，在同步时，设备仅仅对自己感兴趣的对象进行同步。在 Cimibiosys 中并不支持协同编辑，同步的最小粒度为对象且具有较新版本的对象会直接覆盖旧对象。

Docx2go[40]旨在提供微软 Docx 格式文件之间的协同编辑。针对 Docx 格式中的树形结构，作者设计了骨架树的方式来维护客户端的部分副本。Docx2go 采用了 LOGOOT[7]算法做为同步协议，如前文所述，尽管 LOGOOT[7]算法的时间戳能够直接确定全局的全序关系，但是它以时间戳的长度无限增长为代价，这也成为了系统中的一个缺陷。

Hydra[24]是一个在线的协作移动评论系统，他和本文采用了一样的同步协议。和 Docx2go[40]类似，Hydra 也在客户端以增量的方式维护了一颗骨架树副本来减少内存的使用。作者在文章也提出了一种新的地址空间转换算法同步协议，采用了中心化的结构和标量时间戳。然而，当树的高度为 1 时，采用骨架树的方法就不那么有效了，整个评论结构都会从服务器上下载至客户端。然而在现实生活中扁平化的树形结构十分的常见，一个权威用户发表的微博评论，一个复杂的 html 文档等。另外骨架树的部分复制方法也不能适用于线性结构，在算法层面上并不具有通用性。

在整个分布式系统领域中，部分复制技术十分普遍只不过支持的粒度各有所异。比如，针对整个网站客户端仅仅加载自己感兴趣的网页也可以认为是部分复制技术的应用。本文的后续章节将介绍如何将如此普遍而重要的技术应用于协同编辑算法，并且保证了算法在协同应用中的通用性。

2.4 本章小结

本章首先介绍了传统的系统编辑算法分为三大类，然后逐个介绍了三类算法分别的基本思想，其后介绍了 AST 相关的领域工作。第二部分介绍了字符串环境下，当今协同编辑领域做的相关工作，其中包含 OT 和 CRDTs 领域的若干代表算法。最后本文介绍了部分复制系统在分布式系统中的基本思想，与其在协同编辑领域的应用，着重介绍了 Hydra 这个在应用层面使用部分复制技术的在线协同评论系统。

本章所介绍的内容体现了同一种优化的思想在不同算法上的展现，从相关工作看本文所介绍的地址空间转换算法的优化技术并不是空穴来风，而是借鉴了协同编辑领域甚至是分布式系统领域其他学者的成功案例，并将其思想合理的体现在地址空间转换算法上，这样做的目的是使得不同的优化方法能够适配更加丰富的应用，有效的提高了未来用户在使用协同编辑系统中的体验。

第三章 支持字符串操作的地址空间转换算法

本章介绍主要讨论地址空间转换算法如何通过模型上和算法上的改动来支持字符串操作。本章的第一部分介绍模型的基础知识，第二部分介绍 ASTS 算法实现原理和实现，并且引入核心的节点分裂操作，第三部分通过结合例子证明了算法的正确性，最后一部分介绍了 ASTS 算法的一些优化方法。

3.1 模型定义

广义上的地址空间转换算法可以分为三部分：时间戳与操作结构，副本存储结构也就是通常所说的文档模型和算法部分，前两部分为模型中的数据结构，后一部分为操作执行的算法依据也是维护文档最终一致的排序规则。

3.1.1 时间戳与操作结构

和所有基于 AST 的算法一样，模型中的每个操作都会附上一个时间戳。时间戳用来确定操作之间的关系，和构建全序值，首先给出时间戳的定义。

定义 3.1 时间戳[42] 时间戳是一个 n 维向量，其中 n 表示协作站点的个数。在第 i 个站点的时间戳向量中的第 j 维表示第 i 个站点当前已经执行了 j 站点上的操作数。

AST 中所采用的时间戳为分布式系统常用的时间戳，可以用图 3.1 中简单的例子来给出时序图与时间戳的对应关系，站点在没产生（接收）一个操作之后会将对应站点号的分量递增。

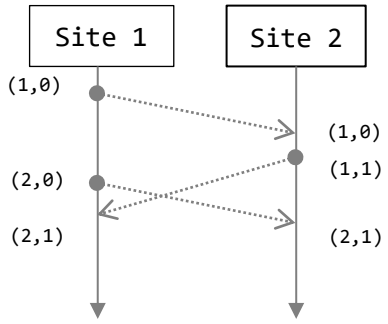

图 3.1 各站点时间戳产生（接受）的变化

一些研究者认为采用向量时间戳可能会导致时间戳大小随着合作者数量增加，从而增加了网络传输量。对于这个问题 DCV[35]或是采用标量时间戳给出了很好的解决方案，也能够和本文提出的算法思想兼容。由于本文讨论的重点并不在此，因此下文中以向量时间戳为例描述算法。

在分布式系统中，操作之间的关系可以分成两类：因果关系和并发关系。对于大多数的并发控制算法都需要根据时间戳来检测操作之间的关系，本文所采用的操作检测关系也和传统的定义一样。

定义 3.2 因果关系[42] 存在两个操作 O_1 和 O_2，O_1 因果先序于 O_2 记做“O_1->O_2”当且仅当满足以下两种情况：（1）O_1 和 O_2 产生在同一个站点，O_1 在 O_2 之前发生。（2）O_1 和 O_2 产生在不同站点，假设 O_1 产生于站点 a，O_2 产生于站点 b，当 O_1 在站点 b 上执行之后 O_2 再产生。

定义 3.2 并发关系[42] 存在两个操作 O_1 和 O_2，O_1 和 O_2 为并发关系记做“O_1||O_2”当且仅当满足，O_1->O_2,O_2->O_1 同时不满足。

因果关系和并发关系通俗的解释就是能够区分先后顺序和不能够区分明确的先后顺序的操作对，典型的因果先序操作对和并发操作对样例可见图 3.3, 其中 A -> B, A -> C, B || C。

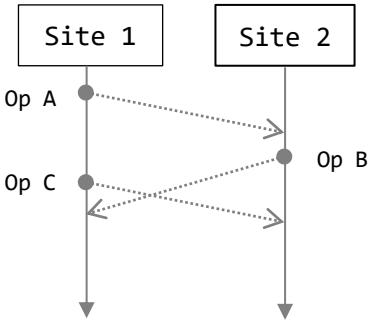

图 3.2 因果先序，并发关系操作样例

3.1.2 副本存储结构

为了便于清晰的展示算法思想并且便于读者的理解，本单元讨论的算法基于 AST 的线性存储结构，但是一样可以用[42]提出的红黑树优化 AST 的性能，使其操作的执行花费降低为对数级别，详细的优化方法将留作后续单元讨论。

和操作定义一样，相较于传统 AST 的节点设计只改变了在节点上支持存储字符串，用户在一次操作所产生的字符串会存储在一个节点上。

定义 3.3 节点 在本地副本中节点是最基础的存储单元，节点由以下几部分组成：

- String 节点中表示的字符串
- Operation List 作用于 string 的操作列表
- Effective 有效标记表示在当前文档状态下该节点已经被删除

每个节点都会对应唯一的一个插入操作，和多个删除操作且这些删除操作互相之间为并发关系。其中的原因是，只有插入操作的过程才能够创建节点，以及用户不能够对统一节点重复执行删除操作，节点的具体结构则在文档模型介绍完后一同给出。

定义 3.4 文档 文档是由节点组成的有序序列。

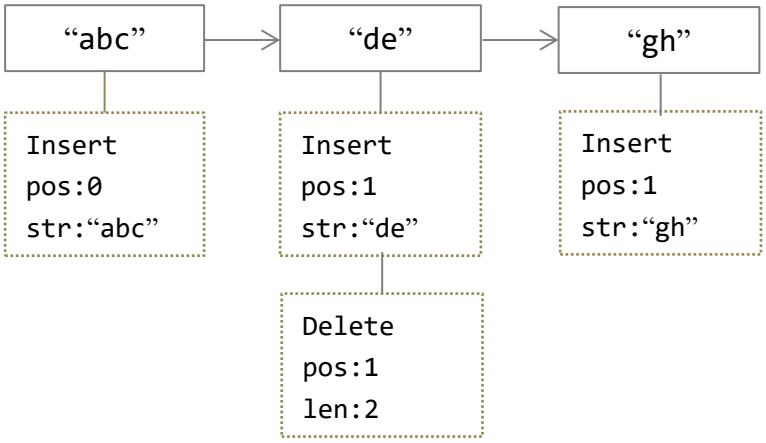

图 3.3 文档结构样例（省略时间戳）

节点按照一定的顺序组织起来，并建立合适的索引方式成为文档，因此文档的存储结构并不仅仅为线性结构，还可以为平衡二叉树和哈希双向链表。上图展示文档样例（省略时间戳）。

3.2 算法定义

算法作为整个地址空间转换的核心部分，地址空间转换算法可以分为三个部分：控制算法，回溯算法，操作执行算法。地址空间转换算法的核心思想是：让操作在他产生的地址空间下执行。控制算法保证了整个算法以“回溯-操作执行-回溯”的流程执行，回溯算法确保操作在执行之前地址空间已经回复到操作产生时候的状态，操作执行算法负责整个节点的排序和数据结构的维护。本文所讨论的支持字符串的地址空间转换算法，在操作和文档结构上虽然发生了变化，但对

于整个算法的流程和核心思想并没有变化，其中控制算法和回溯算法由于和原始的 AST 一致因此本文只介绍一下算法思想，具体的算法细节在 AST 原始论文中。

3.2.1 控制算法

控制算法负责控制 ASTS 的执行流程，对于一个需要执行的操作都需要确保当前文档的状态和该操作产生时的状态一样。因此，控制算法会先调用回溯算法将文档回溯到该操作产生时的状态。接着执行这个操作，最后将文档回溯到原来的状态。

3.2.2 回溯算法

当远程操作到达时，回溯算法用来确保它能够在合适的地址空间中执行。在这一步骤，算法需要根据操作的时间戳和节点的操作信息来计算每个节点上的有效值。通过定义 3.5 可知节点对应唯一的一个插入操作和若干互为并发关系的删除操作，因此对于一个节点可以通过依次判断每个操作是否有效来决定当前节点是否有效。以下给出定义：

定义 3.5 有效值[42] 对于一个远程操作 O，一个节点是有效的，当且仅当满足以下两个条件：（1）节点的插入操作因果先序于操作 O（2）节点操作列表中仅仅包含一个插入操作，或者剩余所有的删除操作都和操作 O 为并发关系。否则这个节点则是无效的。

根据有效值的定义，能够对文档中所有的节点计算出远程操作 O 所对应的有效值，由此回溯算法的大致可分为两个步骤：

（1）依次判断节点操作列表中每个操作的有效性。

（2）根据有效值定义设置节点的有效标记。

3.2.3 操作执行算法

当回溯算法确保了操作执行的地址空间时，操作就可以根据操作执行算法在文档上执行了，这一章节关注于支持字符串操作的算法模型和单个字符的算法模型的区别。

和基础的地址空间转换算法不同的是，节点存储的基本信息是字符串，因此用户的操作位置可能发生在节点的中间。因此本文提出了分裂操作来解决这个问

题，在操作执行之前检查当前操作有效边界的位置。如果其中一个有效边界位于某一个节点的中间，那么就需要分裂这个节点。接着解释一下何为有效边界，对于插入操作，有效边界定义为操作所包含的字符串需要插入的位置，对于删除操作则定义为删除操作在文档中的开始位置和结束位置，对于删除操作而言，无论是开始还是结束位置处在节点中间都需要做分裂操作。以下给出算法的伪代码与说明。

```
算法 1 AST-Execute
Document : D
Operation : o
1.  Function AST-Execute(D, o)
2.      Count = 0;
3.      For Node in D
4.          If Node.effective == 1 Then Count += Node.length
5.          If Count >= o.position Then
6.              If o.type == Insert Then
7.                  Rangescan(D, Node, Count - o.position, o)
8.              End If
9.              If o.type == Delete Then
10.                 Delete(D, Node, Count - o.position, o)
11.             End If
12.             Break
13.         End If
14.     End For
15.End Function
```

算法 1 展示了操作执行的基本流程，其中 D 为文档 o 是需要被执行的操作。首先找到有效边界的开始（第 3-5 行），然后对于插入操作调用 Rangescan 算法（第 7 行），对于删除操作则调用删除操作执行函数（第 10 行）。

```
算法 2 Rangescan
Document : D
Node : NS
shift : start position in the Node
Operation : o
1.  Function Rangescan(D, NS, shift, o)
2.      N <- NS.next.shift
3.      P <- NULL
```

```
4.      While Nodes containing n is not effective
5.          If TOrder(N) > TOrder(o) And p == NULL Then
6.              P <- N
7.          End If
8.          If TOrder(N) < TOrder(o) And N -> P Then
9.              P <- NULL
10.         End If
11.         If N -> o Then P <- N, Break
12.         N <- N.next
13.     End While
14.     If P is NULL Then P <- N
15.     If P is in a Node Then Split(Node, P)
16.     Create a new node and insert in P
17.End Function
```

算法 2 展示了 ASTS 的 Rangescan 算法，首先和原始 AST 一样需要给出 Torder 定义。

定义 3.6 Torder[42] Torder 对于每一个操作是一个唯一的值，给定操作 O1 和 O2 和对应的时间戳 T1 和 T2，有 Torder(O1)<Torder(O2)，当且仅当（1）T1 中每个分量的和小于 T2 中每一个分量的和，或者（2）T1 中每一个分量的和等于 T2 中每个分量的和，并且产生 O1 的站点号小于产生 O2 的站点号。

Rangescan 算法用于找到插入操作的精确位置，和原始 AST 算法有所差别的是如果插入位置是在一个节点之间则需要对节点进行分裂然后再插入（第 15-16 行）。

```
算法 3 Delete
Document : D
Node : NS
start position in the Node : shift
Operation : o
1.  Function Delete(D, NS, shift, o)
2.      P <- NS.shift
3.      If P is in a Node Then Split(Node, P)
4.      If P + o.length is in a Node Then
5.          Split(Node, P + o.shift)
6.      End If
7.      For Node in [P, P + o.length]
8.          Add o on the Node
```

```
9.      End For
10.End Function
```

算法 3 介绍了删除操作的执行算法，在 ASTS 中操作对象是一个字符串，因此删除操作的有效边界可能横跨多个节点。操作对应的有效边界由算法 1 计算好从参数传入，假设初始 P 指向第一个有效边界，第二个有效边界在 P + length 位置。然后依次检查两个有效边界，对需要分裂的节点进行分裂（第 3-5 行）。其后算法遍历所有位于有效边界中间的节点，将删除操作添加到对应节点的操作列表中（第 7-9 行）。

```
算法 4 Split
Node : n
Position : P
1.  Function Split(n, P)
2.      ncopy = n
3.      n.string = sub(0, p-1)
4.      For operation in n.operationList then
5.          operation.string = sub(0, p-1)
6.      End For
7.      ncopy.string = sub(p, n.string.length-1)
8.      For operation in n.operationList then
9.          operation.string = sub(p, n.string.length-1)
10.     End For
11.     Add node ncopy behind n
12.End Function
```

算法 4 展示了节点的分裂。首先需要分裂的节点会进行一次拷贝产生一个副本（第 2 行）。对于原始的节点需要更新节点和节点中操作列表中的操作对象为原始操作对象的长度为 p 的前缀（第 3-6 行）。对于拷贝的节点同样更新节点和节点中操作列表中的操作对象为原始操作对象的长度为 n.string.length - p 的后缀（第 7-10 行）。最后将节点副本插入到当前节点之后。

3.2.4 一个例子

对于上文提出的算法，本节中通过例子进一步的介绍：

如图 3.4 所示，假设存在三个站点，分别为站点 1，站点 2，站点 3，分别有

四个操作 A、B、C、D 操作的具体信息展示在图 4 的操作列表中，本文以站点 1 的视角来展示 ASTS 的算法执行过程。

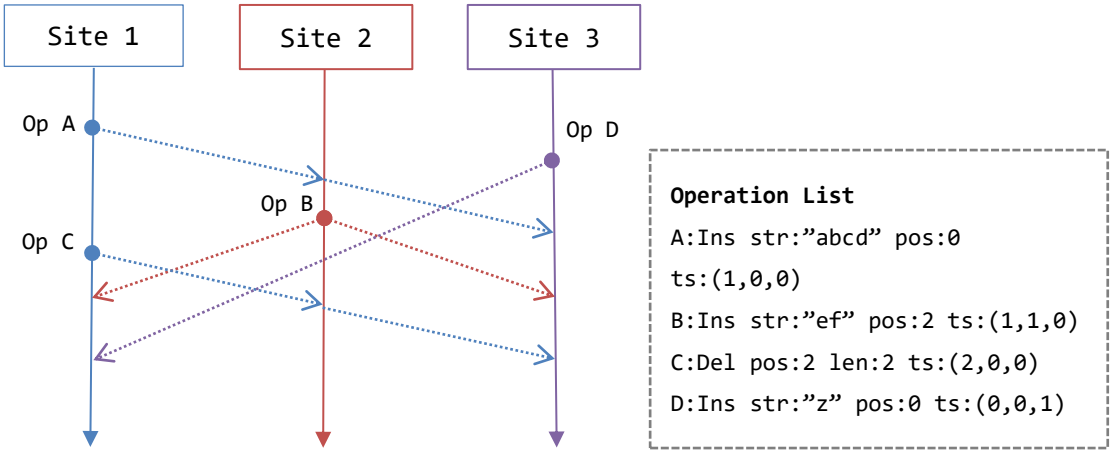

图 3.4 样例中的操作时序图和操作列表

如图 3.5 所示，当站点 1 产生操作 A 后会在文档中建立一个新节点，由于 abcd 的插入是用户的连续行为而且发生在同一个位置，因此被合并成单个操作。其后站点 1 又产生的操作 C 也是本地操作，删除从位置 2 开始的 2 个字符也就是删除“bc”，首先算法会检查操作的有效边界，显然边界 b 前和 c 后都在节点的中间，因此对应节点需要进行分裂操作，分裂操作后的文档拥有三个节点分别为“a”、“bc”、“d”，原本在节点上的操作列表也随着节点的分裂而一同复制，在节点分裂完之后操作执行算法会找到操作位置节点“bc”然后将操作加入其操作列表中，接着站点 1 收到了远程站点的操作 B，在第二个位置插入“ef”，由于是远程操作首先要进行地址空间回溯将操作回退到时间戳(1, 1, 0)时的文档状态，等

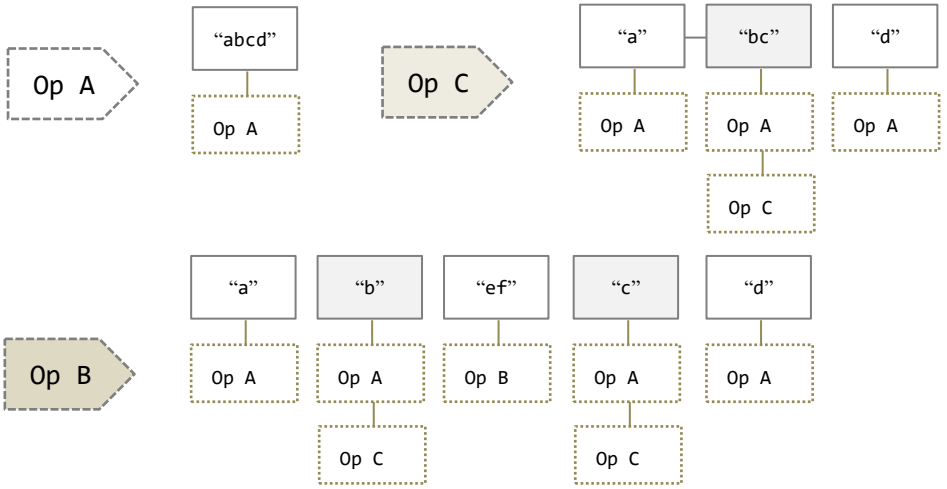

图 3.5 前三个操作执行过程中站点 1 文档结构所发生的变化，操作的执行顺序为 A -> C -> B

价于操作 A 刚刚执行完的文档状态，这里需要注意的是地址空间回溯并不会将原先已经分裂好的节点再合并回去，而是简单抹去操作 C 的效果，而后操作执行算法会找到插入位置在字符“b”后，创建节点并进行插入操作。

最后来讨论操作 D 在站点 1 上执行的情况，首先操作 D 为远程插入操作操作，文档首先进行回溯，此时所有的节点都会变成无效节点，然后操作执行算法会在整个文档范围中进行 Rangescan 查找具体插入位置。这里若是读者对 Rangescan 算法十分了解则会意识到，在单个字符模型中 Rangescan 依靠 Torder 来判断节点插入的位置，在整个文档中任意两个节点的 Torder 都是不同的，而在 ASTS 中由于引入了节点的分裂操作，同一时间同一站点上插入的节点会最终变为多个节点分布在文档中间那么 Torder 值就可能不唯一了，会不会影响 Rangescan 的算法执行呢？答案是显然不会。下文通过证明几个性质来依次解释这个问题:

性质 3.1 分裂后节点之间的节点一定因果后序于被分裂的节点，也等价于中间节点 Torder 值要大于等于两边的节点。

性质 3.2 分裂后的多个节点要么全部包含在 Rangescan 的范围中，要么全都不包含在 Rangescan 的范围中。

证明：这个问题可以采用类似于数学归纳法的思想做一个简单的证明。

初始情况：文本中的节点都未被分裂，此时若有节点需要分裂，只有一种可能就是一个因果后续的插入或删除操作的有效边界在节点中间，此时操作执行完后还是满足上述性质。

每插入一个新节点的情况：从初始情况满足性质 3.1 可以推出新插入节点的 Rangescan 的范围一定也满足性质 3.2，所以后续插入的节点不会因为 Rangescan 边界而插入原先和其并发的分裂节点中。接着只需要考虑，在 Rangescan 范围内的多个 Torder 相同的分裂节点，可以分三种情况讨论，由于有 Torder(中间节点)>=Torder(分裂节点）

Case 1：Torder(插入节点)>Torder(中间节点)>=Torder(分裂节点）

算法执行到区间末尾时插入位置在该范围之后，真个遍历过程中 P 的值不会被更新。

Case 2：Torder(中间节点)>=Torder(分裂节点)>Torder(插入节点)

算法执行到区间末尾时插入位置在该范围之前，算法遇到第一个节点就会停

下来且中间的节点并不会置空原来的值。

Case 3：Torder(中间节点)>Torder(插入节点)>Torder(分裂节点)

算法执行到区间末尾时插入位置在该范围之后，即使前面找到了 P 的位置后边界也会清空 P。

综上所述，新插入的节点也会同时满则性质 3.1 和性质 3.2，性质 3.1 和性质 3.2 同时成立。

证明完毕。

接着讨论操作 D 在站点 1 上的情况，操作 D 的 Torder 为(1, 3)，Torder(Op B)=(2, 2)>(1, 3)>Torder(Op A)=(1, 1)，属于上述 Case3，根据结论新插入的字符“z”会越过所有节点插入在文档的最后，最后的文档结构如图 3.6 所示。

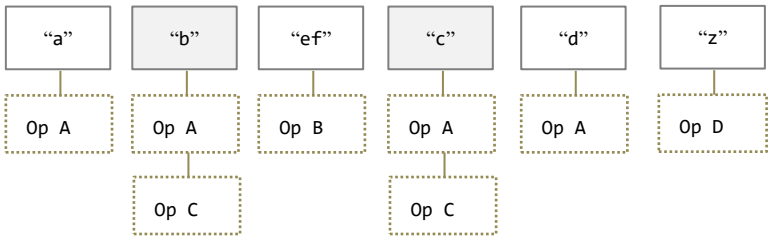

图 3.6 执行操作 D 后站点 1 的文档结构

3.3 算法优化与选择 Undo 操作

上文介绍了 ASTS 算法数据结构和算法部分的所有内容，整个算法都采用线性结构的文档模型，对于操作的执行效率较低功能较少。在 AST 原始论文中也提到了算法的优化和 Undo 操作的支持，在本节中将对于 ASTS 的这两方面技术进行讨论。

3.3.1 算法优化

原始 AST 可以采用平衡树对算法效率进行优化，原理是优化了原先算法回溯过程的时间负责度和文档长度成正比这个缺点，将文档用平衡树组织起来，通过在节点上维护当前子树中有效节点的个数来快速的实现回溯过程。另外，算法还需要维护一个每个站点的操作历史队列，这样就能够倒序的找到所有和当前执行操作为并发关系的操作，再一次从当前操作到根节点更新有效节点值。

由于 ASTS 仅仅改变了操作和文档的结构并没有改变算法思想，因此这种优化思想也可以应用在 ASTS 中。唯一需要注意的是，由于分裂操作涉及到文档结构的变化，因此在分裂的同时需要维护节点上的信息。本文中作者的实现基于一种 SBT 的平衡树，在几种平衡树中 SBT 常数最小且实现较为简洁，具体可以参考原始论文[51]。

节点平均大小指的是一个节点中存储的字符串的平均长度，后文的实验中可以看到节点平均大小是小于操作字符串平均的大小的。由于分裂操作会将一个节点拆分成两个节点，在最坏的情况下 ASTS 会退化为和 AST 一样每一个节点都是单个字符的。这个问题本文还没能给出一个合理的解决方案，然而这并不会对算法在实际使用场景中的效率产生多大的影响，由于用户的操作都是具有语义性的，连续插入的往往为一个短语或一个句子从语义上具有连续性，因此只修改一部分的情况较少，所以上述的最快情况几乎不可能出现。

3.3.2 支持选择 Undo 操作

在文档编辑中，Undo 操作是一个用户频繁使用的操作，几乎所有的文档编辑器都会支持 Undo 操作。在协同编辑中，存在多个用户同时编辑文档的场景，此时会有并发操作的产生传统使用按按时间顺序撤销用户操作并不能够很好的满足用户的需求。因而产生了选择 Undo 操作，选择 Undo 操作通过提供给用户一个过去操作的历史列表让用户可以选择自己需要回退的操作执行回退操作，这样的操作能够很大程度上提高了协同编辑软件的易用性。

过去对于支持 Undo 操作也存在不少研究，最简单的方法莫过于产生一个新的操作能够和原来的操作进行抵消。然而，这种实现方式从模型层面上来讲并不是十分的自然，Undo 操作往往会引入很多不必要的数据存储，因此降低模型的执行效率。地址空间转换算法能够通过简单的改进支持选择 Undo 操作，在 AST 的原始论文中作者有做详细的讨论，这里就不再详述。

对于 ASTS 在新加入了节点分裂操作后文档模型变的和以前不太一样了，同一个操作可能在文档模型的多个节点上，因此第一个需要解决的问题就是如何使用简单的数据结构来快速检索操作分布的节点。这里可以使用一个叫做操作位置邻接表的东西，本文将分布在数据结构中的操作通过一个哈希表索引起来，哈希值通过操作中的时间戳计算出来，通过使用合适的哈希函数能够使操作尽量均匀的分布在哈希表中。当操作执行的时候将新操作插入对应的哈希值链表末尾即可，或者是节点发生分裂的时候通过节点的哈希值能够在常数时间内找到原来的节

点删除它，然后再插入新分裂后的节点，整个过程也是常数的。

通过操作位置邻接表，能够在令人满意的时间内检索一个操作，接着需要解决的问题就是如何存储 Undo 操作。这里作者采用的方法是在每一个操作结构中加入一个 Undo List，每当接收到一个对应操作的 Undo 请求时，Undo 操作将被存储到对应操作 Undo List 的末尾。可以观察到的是对于一个 Undo 操作也可能存在并发的情况，不过这并不影响判断操作是否被 Undo 进行，可以像对待删除操作一样的对待它。最后需要注意的是 Undo 操作本身也是一个操作，Undo 操作本身也可以支持被 Undo，在地址空间回溯算法判断节点是否有效的时候需要设计一个递归算法来遍历所有的 Undo 操作。

3.4 本章小结

本章首先介绍了 ASTS 的相关定义与文档模型，介绍了 ASTS 文档模型中和原始 AST 的异同。然后对于算法部分包括控制算法，回溯算法和操作执行算法做了描述，着重分析了操作执行算法。在操作执行算法中引入 ASTS 独有的分裂操作，能够使得用户操作位置在节点中间是操作能够合理的在文档模型中执行。其后，通过例子的列举详细的介绍了 ASTS 的算法执行过程，其中也简单的证明了在引入节点分裂操作后，虽然文档中会存在两个 Torder 一样的节点，但并不会对整个算法的正确性造成影响。最后，文章介绍了算法的优化方案和算法对于选择 Undo 操作的支持。优化方案中包括如何采用平衡树来加快地址空间回溯和操作执行算法的时间效率，并且说明了模型退化在实际的使用场景下很少会出现。选择 Undo 则通过加入操作位置邻接表和 Undo List 来支持，并且有着十分高效的执行效率。

第四章 基于全局唯一标识符的中心结构地址空间转换算法

地址空间转换算法自提出以来已经有很多优化的方案。不同架构的算法也被纷纷提出，Hydra 和 Clip 中都提出了可以利用全局唯一标识符来定位操作目标节点，从而来达到算法优化的目的。目前为止基于全局唯一标识符的中心结构地址空间转换算法是所有算法中时间复杂度最低的，且正确性也得到了证明。本小节首先讨论全局唯一标识符在算法中的具体作用，其后介绍基于全局唯一标识符的中心结构地址空间转换算法的一些基本概念和定义，为后文介绍部分复制技术的应用做铺垫。

4.1 地址空间回溯与全局唯一标识符

在过去，地址空间回溯是 AST 算法中的重要组成部分，他有着和逻辑相符合的算法思想。地址空间回溯通过消除文档中并发操作的效果来确定新操作的目标节点，这在原本算法中是最耗时的一个步骤。在整个文档结构中，并发操作一般较少（普通的协同编辑场景中可以认为并发操作的数量相对于文档长度为常数），而地址空间回溯则需要遍历所有的文档结构去找并发操作。在原始论文中作者就意识到了这一点并且提出了基于平衡树的优化手段，通过平衡树将文档节点管理起来，用操作队列快速的找出并发操作然后更新平衡树节点上维护的有效值（该节点所对应的子树中的有效节点个数）在对数时间内完成回溯过程。该优化方法对于算法效率有了明显的提升，然而还没能达到最高效的状态。

最近，通过对于地址空间回溯进行了深入的分析发现地址空间回溯过程其实只是帮助操作定位其目标节点。这里举例说明，下图中展示了一个操作从本地执行完到同步到远程站点的过程。可以发现在本地执行时 b 插入在 a 之后，按照传统通过位置信息来定位目标节点的方法，b 处在 1 号位置（节点 a 为 0 号位置）。当站点同步到远程站点之后，副本中可能已经存在已执行的并发操作，这时候 b 同样应该插入在 a 之后。传统算法通过先进行地址空间回溯，隐藏 z 的插入操作的执行效果然后再找到位置为 0 的节点开始 Rangescan 查找实际插入位置。其实，通过采用 Hash 表管理所有的节点就能够在常数时间内找到目标节点，因此，

通过上述分析得出全局唯一标识符能够更加高效的替代地址空间回溯的功能这个结论。

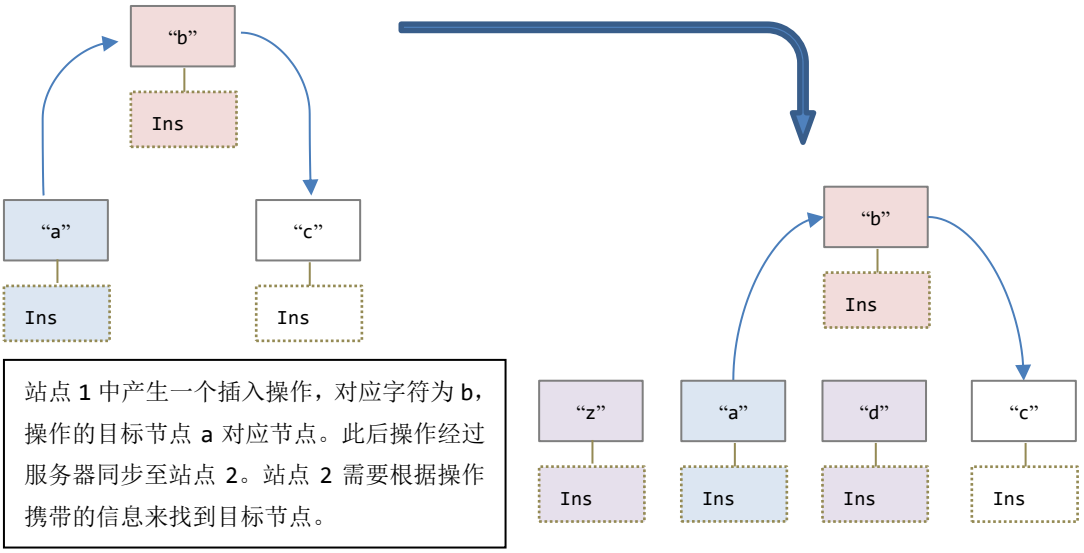

图 4.1 地址空间转换算法中定位节点的过程

4.2 基于全局唯一标识符的模型定义

本小节先对系统的整体框架做一个介绍，然后再给出具体的模型定义。首先，本文所采用的系统为中心化结构，客户端可以为各式各样的终端设备，并且他们具有各自不同的特点。客户端的所有操作都将会上传到中心服务器中，服务器在本地维护一个文档模型，并且将接收到的操作在本地执行，其后，服务器将操作转发到其他客户端中。由于考虑到无线网络的不稳定性，服务器和客户端交互的次数需要尽可能的少以此来减少重传所带来的代价。客户端和服务器的操作交换在本文所介绍的系统中会以操作同步协议的方式进行。如图 4.2 所示，一次请求

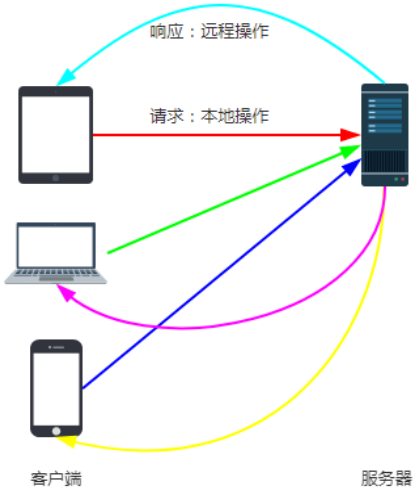

图 4.2 服务器和客户端构建的双路通讯

中客户端会上传本地新产生的操作，服务器则会将新收到的远程操作返回至客户端。

4.2.1 标量时间戳与全局唯一标识符

在 Hydra[45]的基于标量时间戳的地址空间转换算法中给出了一种中心化系统中通用的标量时间戳定义，并且采用了基于操作队列的并发操作检测方法确保了算法的高效。操作到达服务器后会被本地标记时间戳以便服务器之后能够快速的找到需要同步的并发操作，因此对于上述标量时间戳的机制进行简化。

定义 4.1 中心服务器的标量时间戳 中心服务器本地维护一个逻辑时间戳 T，每当发生客户端操作同步时逻辑时间戳自增，同样在操作同步的返回信息中附加此逻辑时间戳。

如图 4.3 所示，服务器管理两个客户端 A 和 B，第一次 B 客户端发起请求是时间戳从 0 变为 1，因此客户端 B 所接受到的操作同步请求中所附加的时间戳为 1。之后，A 客户端发起操作同步请求接收到的时间戳为 2，以此类推。

定义 4.2 全局唯一标识符 客户端 A 的全局唯一标识符 UIA 是一个二元组(S，UCN)，其中 S 为客户端 A 的本地分配计数器，UCN 为全局唯一站点标识符。

全局唯一标识符替代了原来的时间戳，当插入操作产生时客户端会根据本地的全局唯一标识符为操作分配对应的标识符，然后分配计数器自增。由于操作对应于文档结构中的节点，因此，这样保证了文档中的任意一个节点都具有全局唯一标识符。

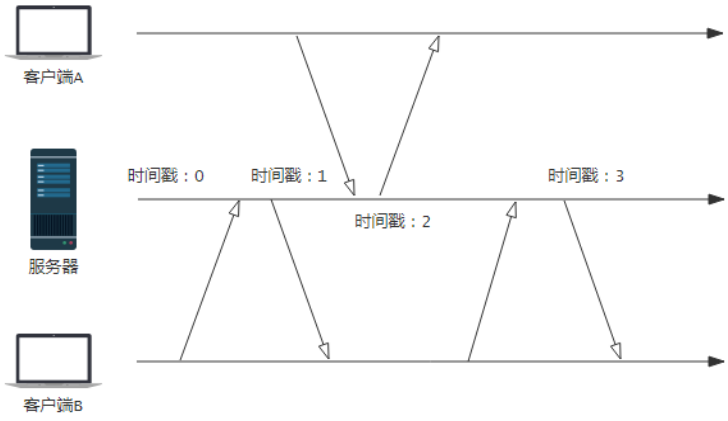

图 4.3 标量时间戳在同步期间的变化

4.2.2 操作结构定义

在地址空间转换算法中操作结构往往包含着两部分信息，第一部分为操作有效信息，其中包括用户操作类型和一些附加的必要信息（如插入操作的字符信息），第二部分则是帮助定位操作目标节点的附加信息。在本文所介绍的基于全局唯一标识符的 AST 模型中，操作所附加的信息即为全局唯一标识符。如下给出操作结构的具体定义：

定义 4.3 操作结构 操作结构为一个三元组[T, D, TN]，其中 T 标示操作的类型包含插入和删除两种，D 为操作所附加的必要信息，对于插入操作包含插入的字符和操作产生节点的全局唯一标识符，对于删除操作则为空。TN 为操作的目标节点，插入操作的目标节点为一个范围分别为操作产生时的前一个有效节点及其后一个节点，删除操作的目标节点则为需要删除的节点。

4.2.3 文档结构定义

为了支持高效的算法执行效率，文档结构也需要做出改变，本文所提出的算法模型中采用哈希表加双向链表的结构。哈希表的键值对应全局唯一标识符，这样能够在牺牲一定存储空间的代价下达到最高的算法执行效率。在[45]所提出的基于标量时间戳的地址空间转换算法中，也采用了类似的文档结构，但是关于操作的存储则是基于队列的。本文对于此文档结构做了改进使其能够和部分复制技术做到兼容。

定义 4.4 节点 文档节点由一个多元组组成[STR, UI, OPLIST, PRE, NXT]，其中 STR 包含节点中的内容，UI 为该节点对应的全局唯一标识符，OPLIST 为作用于该节点的操作列表以数组的形式存在于节点中，PRE 和 NXT 为两个节点类型的指针分别指向节点的前驱和后继。

和上一章介绍的一样节点操作列表中有且仅有一个单独的插入操作和零至多个删除操作，如有两个以上的删除操作那么他们一定互为并发关系。由于这里不存在地址空间回溯的概念因此可以直接认为包含删除操作的节点为无效节点。

定义 4.5 文档结构 文档结构由两部分组成[HT, DOC]，HT 为一个哈希表键值为各个节点的全局唯一标识符，对应的指针指向节点。DOC 为一个双向链表由上述节点结构组成，DOC 的第一个节点 Head 和最后一个节点为 Tail 是文档中的虚拟节点。

在文档中定义虚拟节点是地址空间转换算法模型设计中常用的手段，虚拟节点的加入不仅能够使得每一个操作都有目标节点，而且在 Rangescan 实现时不需要考虑边界情况。虚拟节点仅仅具有全局唯一标识符，并没有对应的操作列表，他们只能作为插入操作的目标节点。并且，从时序关系上来讲，虚拟节点因果先序与系统中的任意一个操作。基于全局唯一标识符的文档结构如图 4.4 所示。

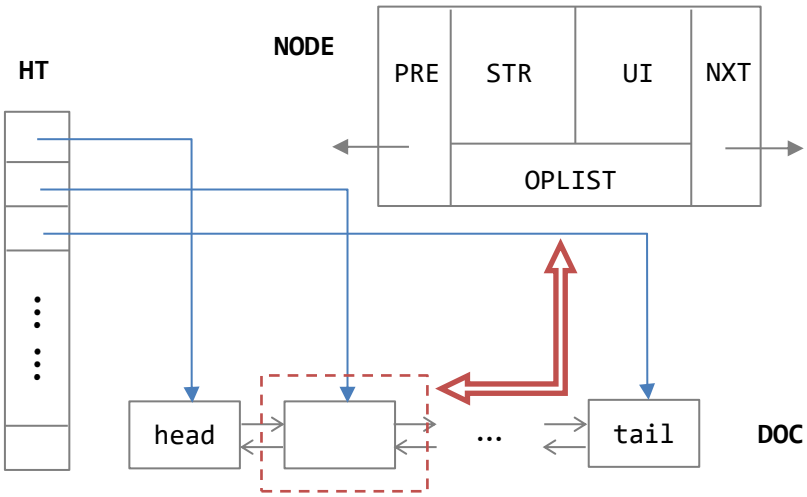

图 4.4 基于全局唯一标识符的文档结构

4.2.4 远程操作的同步

对于一次操作同步来说，服务器需要知道客户端上次同步的时间以此来找出需要同步给该客户端的远程操作。由于部分复制技术的需要在本文中的操作列表绑定在文档节点之上，而在文档结构中又无法快速的找出并发操作。因此，这里需要借助一个临时的操作队列，操作队列中会存储那些还未被客户端同步的操作。一个客户端产生的操作被发送到服务器后，服务器会将其放入这个操作队列中，等待所有客户端都完成同步再将此操作从队列中移除。

定义 4.6 历史操作队列 历史操作队列 Q 是一个双元组[OQ, QN]，OQ 是一个操作队列中的操作会满足以下几点要求：（1）OQ 中的操作按照到达服务器的顺序排列（2）至少还有一个客户端未同步过操作队列尾端的操作。（3）OQ 中的每个操作都会附上到达服务器的操作时间戳 T。QN 也对应一个队列，QN 中的值和操作队列中一一对应为每个操作需要被同步的客户端数量。

历史操作队列用来暂时存储客户端上传的操作，这些操作会在同步到所有客户端后删去。因此，历史操作队列的长度应该等于系统中并发操作的数量，在一般的协同编辑系统中并发操作的数量相对于文档长度可以认为是常数个。

4.3 基于全局唯一标识符的地址空间转换算法实现

采用了新的操作结构和文档结构后，算法定义也需要做出对应的更改。本小节采用了 Hydra[45]中的基于全局空间标识符的地址空间转换算法的部分思想，去除了地址空间回溯操作，保留了文档结构的设计。这样整个算法模型在效率不损失的情况下变得更加简单，另外还对后文提出的部分复制技术提供了适配。

关于算法部分的介绍分为两部分，控制算法，操作执行算法。每次客户端发起一次操作同步时，会将本地产生的操作序列上传至服务器。服务器接收到操作序列后依次调用操作执行算法在本地执行，然后返回远程操作序列。

4.3.1 控制算法实现

算法 5 Control Algorithm in Server (QOP, R, Q, LTS, TS) 参数：客户端上传操作序列 QOP， 服务器副本 R, 历史操作队列 Q， 客户端上次和服务器同步的时间 LTS 服务器当前时间戳 TS 返回：需要同步到客户端的操作
1.For i in 1 to |QOP| 2. O = QOP[i] 3. Execute(O, R) 4.End For 5.Ret = [Q.OQ | Q.OQ[i].T > LTS] 6.Update(Q) 7.Q.OQ = Q.OQ + QOP 8.TS ++; 9.Return [Ret, TS]

算法 5 介绍了服务器端的控制算法，收到了客户端的操作同步请求后服务器首先在本地的副本上执行每一个操作。其后更新队列 Q，通过找到当前所有客户端中上一次同步的最小值来删除队列中已经被所有客户端同步过的操作。接着更新 Q.QN 中每一个被发送的计数减一，然后删除 Q.OQ 中计数为 0 的操作。

算法 6 Control Algorithm in Client (QOP, R, Q, LTS) 参数：客户端本地操作序列 QOP， 客户端本地副本 R 客户端上次和服务器同步的时间 LTS
1.(Ret, TS) <- 向服务器发送操作同步请求 2.LTS = TS 3.For i in 1 to |Ret| 4. O = Ret[i] 5. Execute(O, R) 6.End For

客户端的控制算法流程十分的简单，首先向服务器发送操作同步请求得到远程操作和新的时间戳，其后将远程操作在本地副本执行并且更新时间戳。

4.3.2 操作执行算法实现

操作执行算法是协同编辑系统中维护一致性的关键，和传统 AST 模型一致，对于插入操作采用 Rangescan 查找实际插入位置，对于删除操作直接将操作附加在节点之后。客户端和服务端的操作执行算法一致，具体算法流程如下：

算法 7 Execute Algorithm (O, R) 参数：操作 O， 本地副本 R
1.If O.T == ‘ins’ Then 2. Rangescan(O) 3.Else If O.T == ‘del’ Then 4. Add O in R.HT[O.UI].OpList 5.End If

4.4 协同编辑算法中的排序规则及讨论

节点排序规则是用来在协同编辑算法中确定插入操作的实际插入位置。节点排序规则也往往是协同编辑算法中的精髓，对于 AST[32]、RGA[9]、LOGOOT[7]、WOOT[6]等算法都是基于不同的节点排序规则的。以本文所采用的 AST 算法为例，在操作确定了目标节点后对于插入操作而言两个目标节点中往往存在着其他客户端产生的并发的插入节点，新插入的节点对于这些节点的相对位置需要根据 Rangescan 算法来确定。同样其他的基于节点的算法也十分类似，对于新插入的

节点都需要采用排序算法来确定节点在文档中的具体位置。另一方面，排序算法之间也存在很大的差异，典型的如 LOGOOT，排序规则十分的简单，而相对于 AST 和 WOOT 排序规则要复杂的很多。由于排序规则设计的复杂，证明之困难，本小节首先介绍排序算法在文档集合中构建全序关系需要满足的条件，然后介绍 Rangescan 算法及其在基于全局唯一标识符的 AST 中的等价算法，最后分析 Rangescan 是如何满足上述提出的条件在文档集合上构建全序关系的。

4.4.1 全序关系的满足条件

对于协同编辑的文档模型来说可以看作为一个集合 E，而协同编辑中的排序算法则需要构建这个集合中的二元全序关系。

定义 4.7 全序关系 集合 E 上的存在全序关系采用≤表示，对于集合中任意选出的 a，b，c 需要满足几个条件：

- 自反性 如果 a≤b && b≤a 则 a 和 b 相等。
- 传递性 如果 a≤b && b≤c 则 a≤c 成立。
- 完全性 一定存在 a≤b || b≤a。

全序关系需要满足对于任意从集合中选出的节点都有自反性，传递性，完全性。在协同编辑中无论是 AST 算法中的 Torder 还是 CRDTs 中的全局唯一标识符，根据三个条件中自反性的要求，如果出现了 a≤b && b≤a 那 a 和 b 必须为同一个节点。传递性则是协同编辑下最需要考虑的情形，由于用户的插入操作只确保了可见节点之间的顺序，而在数据结构中用户并发插入的节点的顺序是依赖算法决定的，因此算法中的排序规则必须满足传递性。完全性对于设计上的要求相对简单，一般来说只需要给节点加入一个可比较的标识符及可满足。

协同编辑算法中删除操作是不能改变节点的位置的，因此几乎所有的算法都只为插入操作设计特殊的排序算法。为了满足最基本的 CCI 模型，在用户操作体现在文档上时一些关系已经确定了，本文称这些关系为依赖关系。

定义 4.8 依赖关系 对于文档集合 E 中的任意两个节点 a 和 b，假设 b 节点的插入操作发生时 a 节点已经在文档中，满足 a 依赖 b 写作 a => b 必须满足以下任意一个条件（1）a 为有效节点且 b 节点的插入操作的目标前驱节点为 a，这样的情况称为直接依赖。（2）存在序列[O1, O2, …, On]，存在 a 直接依赖 O1，O1 直接依赖 O2，…且 On 直接依赖 b。

对于依赖关系的满足等价于在 CCI 模型中对于用户的意图保持，也就是说在插入操作产生时对于操作目标前驱节点以及之前和操作目标节点以及之后都已经通过依赖关系确定了一个偏序关系，而剩余的无效节点和并发插入节点之间的关系需要排序算法来完成全序定义的。因此，列举在插入操作的执行区间内可能存在哪几种情况：

对于插入操作 O 插入的节点为 a，对应的目标节点为[s, e]，在[s, e]中任意存在的两个节点 b，c，有以下几种可能存在的情况：

（1）b || a && c || a && b || c （全并发关系）

（2）b || a && c || a && b -> c （偏并发关系）

（3）b -> a && c || a && b -> c

（4）b -> a && c -> a

如果以上四种情况都能对任意的 a、b、c 满足自反性，传递性和完全性，那么结合之前的依赖关系本文称该二元关系是在文档集合 E 上全序关系。

4.4.2 Rangescan 算法与 Torder

在脱离了时间戳的支持后，不能够采用操作之间的关系来判断节点插入位置。而原有的 Rangescan 定义是基于操作之间的关系来判断的，这里本文提出一种等价的方法来替代原有的算法，以适配基于全局唯一标识符的地址空间转换算法。在介绍操作结构是，本文定义了操作的目标节点，而插入操作的目标节点为一个范围对应 Rangescan 的扫描范围。原先 Rangescan 的扫描范围为插入操作的前一个有效节点到后一个有效节点，而现在操作执行的范围为前一个有效节点及操作产生当时前一个有效节点的后一个节点。

算法 8 Base-rangescan (CNa，CNb，CNnew) 参数：[CNa, CNb]为 Rangescan 的查找范围 CNnew 位置的返回值
1.P = Nill and CNscan = CNa.right
2.Do
3. Compare CNnew with CNscan
4. If CNscan || CNnew Then
5. If TOrder(CNnew) < TOrder(CNscan) && P=NULL Then
6. P <- CNscan
7. End If

```
8.      If TOrder(CNnew) > TOrder(CNscan) && CNscan -> P Then
9.          P <- NULL
10.     End If
11. End If
12. If CNscan -> CNnew Then
13.     If P == Nill Then
14.         P = CNscan
15.     EndIf
16.     Exit the Loop
17. EndIf
18.     CNscan = CNscan.right
19.While CNscan=CNb
20.If P != Nill Then
21.   RETURN P
22.ELSEIf
23.   RETURN CNb
24.End If
```

上图展示了原始论文中的 Rangescan 算法，CNa 和 CNb 对应的前后有效节点，观察 12 行-17 行循环中一旦遇到有节点因果先序于插入节点，则退出循环，并且插入位置位于该节点的前面。而在操作产生时，根据并发关系的定义（参照上一章定义 3.2）当前客户端的前后有效节点之间是不会存在并发插入节点的，有效节点的后一个节点即为后一个有效节点，或者为一个因果先序于当前插入节点的节点。因此，更改了 Rangescan 的执行范围后，去除此部分的判断和原来的算法是等价的。以下给出支持全局唯一标识符的 Rangescan 算法。

```
算法 9 Rangescan (O)
参数：需要查找位置的插入操作 O
25.P = NULL
26.CNscan <- O.TN.S
27.DO
28.If CNscan || CNnew Then
29.   If TOrder(CNnew) < TOrder(CNscan) && P=NULL Then
30.       P <- CNscan
31.   End If
32.   If TOrder(CNnew) > TOrder(CNscan) && CNscan -> P Then
33.       P <- NULL
34.   End If
35. CNscan = CNscan.NEXT
36.While CNscan=O.TN.E
```

```
37.If P != NULL Then
38.    RETURN P
39.ELSEIf
40.    RETURN O.TN.E
41.End If
```

Rangescan 算法通过计算 Torder 的值来确定节点之间的关系，Torder 的值在上一章给出了定义（可以知道的是如有两个节点 a, b 他们插入操作 Oa -> Ob 一定有 Torder(a)<Torder(b)）。除此之外 Rangescan 还设计了复杂的排序规则来保证全序关系的满足，那么为何不能通过 Torder 的值直接从小到大排序节点呢？参考上述四种情况进行分析：

（1）对于两两并发的三个节点，节点之间的顺序没有限制因此任意的排列都是对的，直接按照 Torder 构建关系没有问题。

（2）对于第二种情况中，观察到 b||a 和 c||a 都为并发关系，他们需要通过 Torder 来建立关系，而由于 b->c 因此 b，c 节点之间的关系是之前确定的，根据 Torder 的定义 b 在 c 的前面，那当 Torder(a)<Torder(b) && Torder(c)<Torder(a)时出现矛盾。

对于（3）（4）两种情况，在确定插入操作目标节点时已经有了清晰的顺序关系。

经过上述分析，仅仅依靠 Torder 排序无法构建文档集合 E 上的全序关系，而 Rangescan 算法则是对于第二种情况做了特殊的处理，第二种情况也称为偏并发关系对于偏并发关系和 Torder 规定顺序出现矛盾时，这里以 b 和 a 的 Torder 顺序为准，换句话说也就是发生顺序比较早的操作具有 Torder 的优先权见算法的 8 行-10 行。

4.5 复杂度分析

本章提出的改进后的全局唯一标识符的算法，通过采用全局唯一标识符定位编辑操作的目标节点，避免了地址空间回溯的执行。相较于传统的地址空间转换算法，模型更为简单效率也变高了。对于本地操作无论是删除还是插入操作平均执行的复杂度为 O(1)，对于远程的删除操作平均执行的复杂度也为 O(1)，插入操作平均执行复杂度为 O(m)，其中 m 为插入操作目标节点范围内的并发节点。一般在协同编辑平台中用户会在同一时间向同一位置执行插入操作的情况较少，

因此 m 可以视为常数且在大多数情况下 m 的值为 1。

表格 1 基于全局唯一标识符的地址空间转换算法执行效率

	本地	远程
插入操作	O(1)	O(m)
删除操作	O(1)	O(1)

4.6 本章小结

本章首先介绍了基于全局唯一标识符的中心化地址空间转换算法的基本概念，定义了全局唯一标识符。接着，介绍了操作的结构和操作执行前目标节点的定位方式。另一方面，在文档结构上采用哈希表的形式，通过额外使用一个临时队列来支持操作同步的快速进行，整个结构相较于原来的更为高效和简洁。最后，本文提出了新的操作同步协议和操作排序算法，针对全局唯一标示符的特点提出了一个等价的 Rangescan 算法，新算法较为原算法更加的简洁。本章算法的介绍也为下一章提出部分复制技术打下了关键的基础，保证了从算法模型层面上地址空间转换算法能对部分复制技术做一个很好的适配。

第五章 地址空间转换算法中的部分复制技术

5.1 引言

在协同编辑系统中，为了快速的在本地响应用户产生的操作，客户端往往会存储一个副本的复制。在早期的协同编辑应用中，编辑对象往往是文本不会占用太多的用户空间，参与编辑的设备也只是 PC 内存空间尚且足够。而随着新一代应用的产生和整个协同编辑算法的发展，第一，编辑对象变得越来越丰富，对象从纯文本信息到结构化信息甚至有一些 3D 模型。用户所编辑的对象在增大，而且移动端的参与又对协同编辑应用提出了更加苛刻的性能要求。从另一方面看，服务端由于计算机硬件设备的不断发展单个服务器的内存往往可以到上百 GB 甚至更高。在传统的全复制技术下，资源占用在服务器和客户端相对对等，这样的结构在现今的协同编辑应用中已经不再适用。因此需要一种更加高效的机制能够多占用服务端的一些资源从而大大提高客户端程序的性能。

部分复制技术是从分布式系统中产生的一种技术，利用了用户在一段时间内的编辑操作往往集中在大副本中的一小个范围这一特点。部分复制技术允许客户端仅仅持有部分副本，而服务端来维护整个副本的完整结构。从应用的角度来说，服务器的资源可以受到人为的控制，当服务器资源不足时可以添置或置换性能更好的服务器。而对于客户来说并不会因为使用一个应用用户体验不好而换手机，从这个角度来说部分复制技术也是更加贴近当前协同编辑应用的现状。回顾其他分布式系统中的协作应用如 Cimbiosys[39]、Docx2Go[40]和 Hydra[24]部分复制技术也已经在这些应用中得到了体现，相对于全复制技术在这些系统中的客户端能够节省大量的内存，从而具有更加好的性能。

协同编辑算法被应用在很多中心化系统中，从最早的 Jupiter[37]到近几年的 TIPS[38]，而部分复制技术一直都没有得到应用，直到最近的 Hydra[24]才采用了一种应用层面的部分复制方法。算法通过构建骨架树来使得客户端可以增量式的维护部分副本，然后利用 AST 来维护每一层的节点关系。这样的做法虽然在移动评论系统中有着突出的表现，但是不能够移植到其他协同编辑应用中，因此需要一个更加泛化并支持部分复制技术的算法模型。

在上一章节中，提出了基于全局唯一标识符的地址空间转换算法，并且讨论

了 AST 的排序规则，完整的定义了一种更加高效并且实现更加轻巧的协同编辑算法。本章在上一章的基础上将会介绍部分复制技术是如何应用在 AST 上的。本章的剩余部分会先说明支持部分复制技术所面临的挑战，接着介绍部分复制技术需要用到的一些数据结构和概念，其后介绍 AST 在算法层面需要做哪些改进来支持部分复制技术，最后讨论部分复制的算法性能和操作执行复杂度。

5.1.1 部分复制技术方法概述

如图 5.1 所示，客户允许在任意时间段内连接服务器，客户端首次上线就会向服务器索要自己需要的副本，这里称为副本同步。其后客户端会在自己的部分副本上编辑，产生的操作会暂存在本地队列中，队列中的操作会定时上传到服务器，并且服务器会返回远程操作。客户端在返回后在本地执行远程操作，这个过程称为操作同步。另外服务器存储的完整文档结构称为全副本，而客户端存储的部分文档结构称为部分副本。在整个同步过程中，服务器会同时和多个客户端交互，但是对于客户端来说永远只存在服务器一个“远程站点”，其他客户端和服务器的同步过程对于单个客户端来说是隐身的。最后客户端如果编辑完了本地的部分副本，可以通过一次新的副本同步向服务器请求其他范围内的副本。

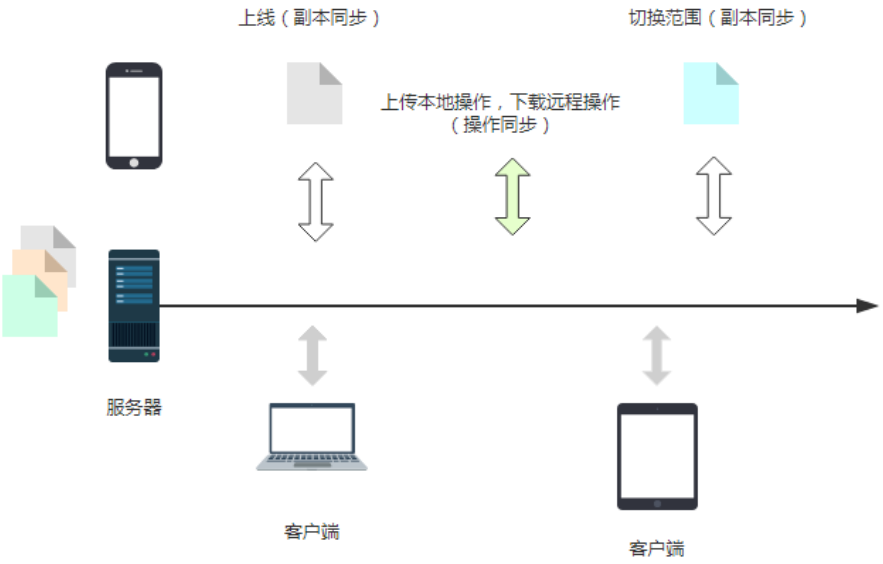

图 5.1 部分复制技术方法流程

客户端在整个过程中，除了副本同步之外全副本这个概念是隐藏的，客户端所执行的操作完全可以把本地的部分副本视作全副本执行。对于服务器来说则需

要记录客户端副本范围，在客户端发起副本同步时及时的更新信息。另外在操作同步过程中为了确保后续操作的合法性，服务器需要适时修正客户端的部分副本范围。

5.1.2 部分复制技术的挑战

和全副本不同的是在采用了部分复制技术后，不同客户端对于副本范围的需求不一样，同一个客户端在不同的时间段内也会请求多次不同范围的部分副本。设计一个高效的支持部分复制技术的算法可能会面临如下挑战：

如何设计部分副本同步协议 为了在系统中给用户提供一个流畅的编辑体验，用户可以方便的切换副本的范围。在切换副本范围后，客户端也不再会保存原有的部分副本转而在新的副本上进行编辑。一方面如何使得用户之前的操作不丢失，另一方面如何保证整个同步过程更加的高效是第一个要面临的挑战。

如何增量的构建副本 在部分副本范围进行改变时，用户可能并不希望舍弃原来的副本而是在现有的基础上增加一段（如：常见的滚动操作）。需要设计合适的机制能够使得部分副本能够增量更新，这是需要面临的第二个挑战。

如何维持客户端和服务器副本的最终一致 最终一致是每个协同编辑系统中都需要面对的挑战，对于存储部分副本的客户端操作的执行机制显然不能依赖于之前或之后缺失的副本。如何为部分复制设计新的操作同步协议是第三个挑战。

5.2 服务器和客户端的副本结构

5.2.1 操作结构与标记

在协同编辑系统中用户操作包括插入和删除，用来表示用户对于文档的编辑行为，本文中称为编辑操作。对于编辑操作，已经在上一章中给出了详细的定义和解释，这里就不再详述。同样，在本文所提出的系统中需要引入发起副本同步的操作，本文中称为查看操作。

定义 5.1 查看操作 查看操作由一个二元组组成[S, L]，S 为用户请求的起始节点的全局唯一标识符，L 为用户请求的副本有效节点长度。

这里需要解释何为副本有效节点的长度，首先有效节点指同步发起的那一刻在服务器上还未删除的节点，换句话说就是节点的操作列表中只有一个插入操作，从用户的角度即为同步了 S 后 L 个可见的节点。

查看操作支持客户端用户随时发起副本同步请求，然而在用户加入协作后对原本的全副本并没有一个相对直观的认识。这时候需要一定的机制来帮助用户找到他真正感兴趣的部分，这里引入标记操作。

定义 5.2 标记操作 标记操作包含一个二元组[Type, Tag, Note]，其中 Type 包含添加和删除两种操作，Tag 表示用户所做标记节点的全局唯一标识符，其中 Note 表示用户对于操作 Tag 的注解。

Tag 为用户在文本中任意划分章节提供了支持，当文章中有章节变更用户可以方便的利用 tag 操作进行标记。同时，需要进行编辑的新用户会首先获得标记列表然后选择对应的副本范围进行编辑。

5.2.2 服务器的副本结构

相对于全复制的协同编辑系统，在支持部分复制技术的协同编辑系统中服务器需要承担更加重要的责任。首先，在整个系统中客户端是动态加入和离开的，客户端加入时需要向服务器请求一定范围内的部分。服务器不仅仅需要向客户端分发给定范围的副本，还需要实时跟踪客户端的副本变化情况，这样才能确保操作同步的合法进行。

活跃范围表（Active Range Table，ART），是一个专为服务器设计的数据结构，目的在于能够记录和查找客户端副本范围变化的情况。以下给出活跃范围表的定义：

定义 5.3 活跃范围表 活跃范围表是一个二元映射<CN, [S, E]>,其中 CN 表示客户端的唯一标识符，[S, E]是一个范围二元组，S 表示当前客户端起始节点的全局唯一标识符，E 表示当前客户端结束节点的全局唯一标识符。

对于客户端的全局唯一标识符由服务器在客户端连接的时候分配，而范围二元组则是由副本同步的时候服务器来构建。

定义 5.4 Tag 集合 各个客户端标记的 Tag 会加入一个 Tag 集合，Tag 集合中的操作是不重复的且以最后到服务器的操作为准。

在本文的协同编辑系统中，由于网络传输的时间，用户产生的 Tag 操作可能会产生冲突，比如删除同一个 Tag 多次或者加入同一个 Tag 多次。为了简化服务器的处理逻辑，对于这种情况服务器会以最后收到的操作为准。

除去为部分复制技术新增的数据结构外，其余的称为文档结构。对于文档结

构采用了上一章基于全局唯一标识符的 AST 算法中的方案。因此服务器的整体结构如图 5.2 所示：

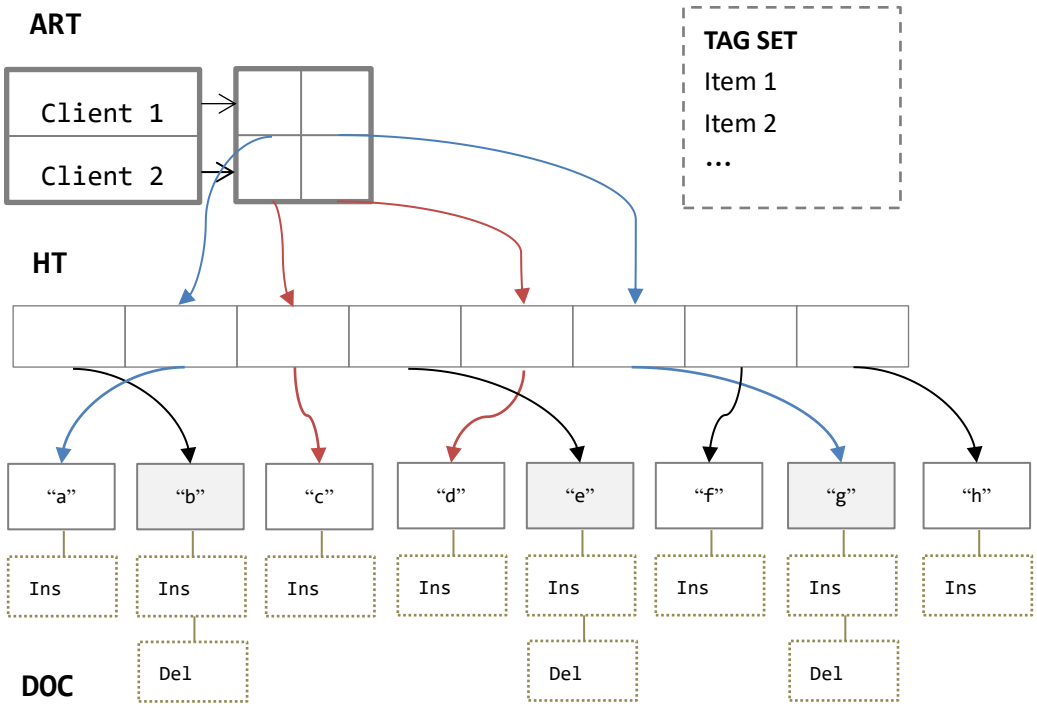

图 5.2 中心服务器上的全副本结构（其中包含文档结构，活跃范围表，标记集合）。

整体的结构可以分为{ART, HT, DOC, TAG SET}这几个部分，上图中是一个典型的服务器副本结构。当前服务器的文本为“abcdfg”，此时连接服务器的客户端有两个，分别指向不同的部分副本。客户端的部分副本范围记录在 ART 中。另外，服务器还存在若干个用户定下的标记存放在标记集合中。相对于原来的全复制结构，整个服务器所持有的结构较为复杂然而考虑到能够为客户端节省大量的内存这点开销还是值得的。

5.2.3 客户端的副本结构

相对于服务器的复杂副本结构，客户端的结构是轻量级的。当用户在一个部分副本上操作的时候完全可以认为他持有了全部的副本，当用户需要不在本地范围的副本只需向服务端发送请求即可。因此，客户端的部分副本仅仅包含单独的文档结构。

如图 5.3 所示，对应于上一节的服务器结构，客户端 1 和客户端 2 本地存储

结构中仅仅包含文档结构，其中客户端 1 的部分副本的可见节点为“acdf”，而客户端 2 的可见节点为“cd”。

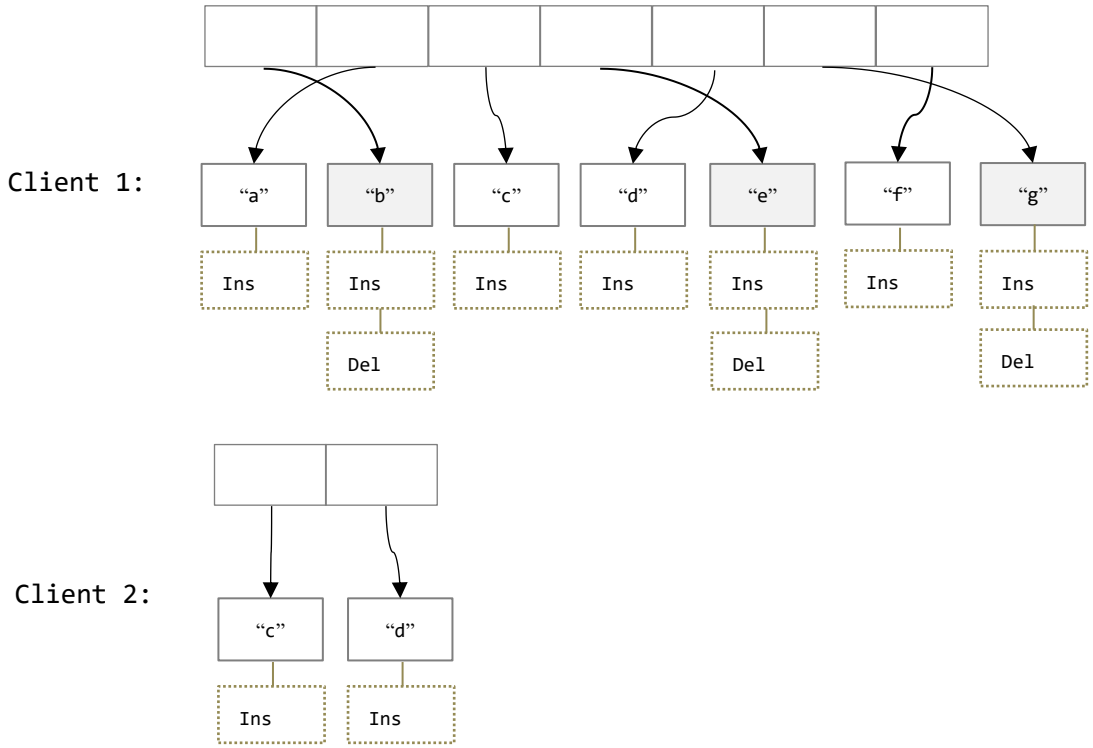

图 5.3 客户端的部分副本结构

5.3 服务器和客户端的同步协议

客户端和服务器端的同步协议分为两种，操作同步协议和部分副本同步协议。在支持部分复制技术的中心化协同编辑系统中操作同步协议是参照上一章定义的操作同步协议的改进。另外，为了支持客户端随时变换副本范围的需求，本文还设计了副本同步协议。下文首先介绍支持部分复制技术的操作同步协议，然后介绍副本同步协议。

5.3.1 支持部分复制技术的操作同步协议流程

操作同步协议在支持部分副本技术的系统编辑系统中会发生一些改变，操作同步的过程中会对客户端的部分副本进行修改这样会消除某些必要存在的有效节点从而导致客户端执行的操作会不合法。这也是部分复制应用中面临的一大挑战，需要解决这个问题首先来分析一下操作在部分副本中执行可能产生的几种情况。（1）操作在部分副本的第一个位置和最后一个位置执行操作，这种情况下

会出现部分副本中前导的无效节点。这样用户执行在文档开头的插入操作就无法找到目标节点了，为了保证系统中文档结构最终的一致性需要避免这样不合法的情况出现。（2）操作在部分副本的中间节点执行。对于这种类型的操作无论是删除还是插入，只要操作执行位置的前后至少有一个有效节点操作就总是合法的。

需要确保在客户端执行的每个操作都是合法的（每一个操作都能够找到他的有效节点），需要保证客户端的部分副本中的第一个和最后一个节点在每一次操作同步后都是有效的，因此也保证了在文档开头和在文档结尾执行的插入操作总能找到合适的目标节点。另外，还需要保证这个有效节点和全副本的 Head 和 Tail 两个虚拟节点一样不能够被用户操作，也就是说他在本次操作同步到下一次操作同步之前一直都是有效的。为了使得操作同步协议能够支持部分复制技术，在每次操作同步之后都会加入一个叫做副本补全的技术。

部分复制中的副本补全技术是指，当客户端发起一次操作同步之后服务器端检查全副本，当发现客户端在下一次同步之前当前部分副本的第一个节点已经不是有效节点时，需要补全副本直到第一个节点为有效节点。同理，当服务器发现最后一个节点不是有效节点时也需要补全副本直到最后一个节点是有效节点。下图展示了支持部分复制技术的算法中一次操作同步的流程。

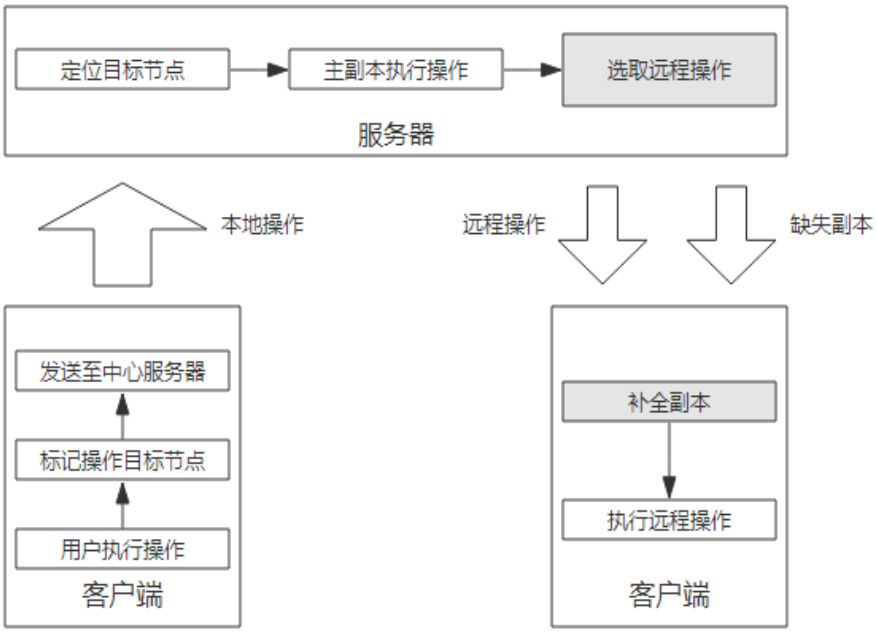

图 5.4 支持部分复制技术的操作同步流程

如上图所示操作在客户端执行操作后首先标记操作的目标节点信息，然后附

带操作本身的信息发送至服务器，其后服务器找到操作执行的目标节点并且在本地执行操作，最后服务器选取出远程操作加入返回值，同时检查是否需要补全副本。最后，远程操作和缺失的副本一起发送到客户端，客户端会进行副本补全并执行远程操作。

由于每次操作同步后，到下一次操作同步之前地址空间是确定的，这里包含两点：第一修改头结点和尾节点的远程操作不会同步到本地，第二本地操作不会修改头节点和尾节点。另外，全副本文档结构中本身就包含虚拟的头结点和尾节点。综上所述，经过每次操作同步的副本补全，在两次操作同步的过程中用户所有执行的操作都是合法的。

5.3.2 副本同步协议流程

部分副本同步协议是客户端发送查看操作时服务器和客户端所进行的一系列交互动作。首先假设用户在发起查看操作的时候已经对本地客户端持有的副本完成了操作，因此此时可以阻塞的进行副本切换从而防止非法操作的产生。另外，假设部分副本同步开始前用户所有的本地修改都已经对服务器完成同步，因此用户原始的部分副本可以被客户端无条件的抛弃。首先，下图展示了整个部分副本同步协议的过程：

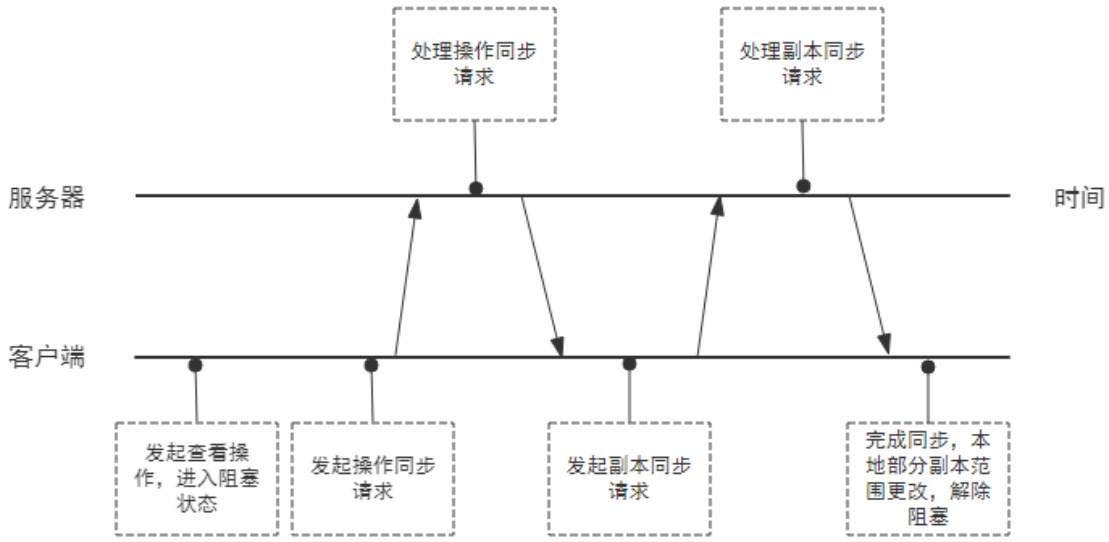

图 5.5 部分副本同步流程

部分副本同步的流程分为以下几个步骤：

1) 用户发起查看操作，客户端进入阻塞状态。

2) 客户端发起一次操作同步，保证所有本地操作已经上传。

3) 客户端发起副本同步请求。

4) 服务器收到请求后，根据请求范围返回对应副本。

5) 服务器修改本地 ART，记录副本范围及对应的服务器时间戳。

6) 客户端接受并更新本地副本，解除阻塞状态。

5.4 支持部分复制技术的 AST 算法实现

本章的前几部分主要介绍了支持部分复制技术所需要用到的客户端和服务端结构，另外还介绍了改进后的操作同步协议和副本同步协议的基本流程。本章的后半部分将补充其中的实现细节，包括如何增量式的补全副本，副本同步时服务器如何更新本地的数据结构等，最后本文会讨论算法每一部分的时间和空间消耗。

部分复制技术是一种通用技术，本节将会介绍部分复制技术在传统线性结构 AST 中的应用。在线性结构的中心化地址空间转换算法中，客户端持有部分副本，服务器持有全副本。操作同步协议采用前文所述的标量时间戳同步协议，数据结构和上一章节类似。节点以线性表的形式存储，节点中包含操作列表，这里不再重复介绍。支持部分复制技术的系统中，客户端会向服务器发送两种同步请求，第一种是操作同步请求，第二种是副本同步请求。

5.4.1 操作同步协议控制算法

在部分复制的情况下，服务器的控制算法分为两类操作同步控制算法和部分副本同步控制算法。在操作同步中，操作执行算法和 Rangescan 已经在上一章中详细的介绍过了在这里就不再详述。和上一章不同的是，为了确保用户的操作在部分副本中执行的正确性，操作同步协议添加了副本补全这个步骤。另外服务器和客户端都需要增加部分副本同步协议对应的控制算法，在本小节将逐个介绍。

算法 10 Control Algorithm in PRServer (QOP,R,Q,LTS,TS,ART,CNUM)
参数：客户端上传操作序列 QOP，
服务器副本 R,
历史操作队列 Q，
客户端上次和服务器同步的时间 LTS,
服务器当前时间戳 TS,
服务器地址范围表 ART,
当前请求的客户端编号 CNUM
返回：需要同步到客户端的操作和需要补全的副本

```
1.For i in 1 to |QOP|
2.    O = QOP[i]
3.    Execute(O, R)
4.End For
5.Ret = [Q.OQ | Q.OQ[i].T > LTS]
6.Update(Q)
7.Q.OQ = Q.OQ + QOP
8.TS ++;
9.While ART[CNUM].S 无效 and ART[CNUM].S != R.DOC.HEAD
10.    ART[CNUM].S = ART[CNUM].S.PRE;
11.    REP.S = REP.S + DOC[ART[CNUM].S];
12.End While
13.While ART[CNUM].E 无效 and ART[CNUM].E != R.DOC.End
14.    ART[CNUM].E = ART[CNUM].E.NXT;
15.    REP.E = REP.E + DOC[ART[CNUM.E]];
16.End While
17.Return [RET, TS, REP]
```

支持部分复制技术的服务器控制代码分为两个部分，第一部分为客户端上传操作在本地副本上的执行，并且找出需要返回的远程操作。第二部分为副本补全，需要找到地址范围表中缺失的无效节点并且将其返回。因此服务器的返回值也同样包括远程操作和补全副本两个部分。

算法 11 Control Algorithm in PRClient (QOP, R, Q, LTS)
参数：客户端本地操作序列 QOP，
客户端本地副本 R,
客户端上次和服务器同步的时间 LTS

```
1. (RET, TS, REP) <- 向服务器发送操作同步请求
2. LTS = TS
3. For i in 1 to |RET|
4.     O = RET[i]
5.     Execute(O, R)
```

```
6. End For
7. For NODE in 1 to |REP.S|
8.    将 NODE 加入当前副本的最前面
9. End For
10.For NODE in 1 to |REP.E|
11.   将 NODE 加在当前副本的最后面
12.End For
```

和服务器的控制算法相对应，客户端的控制算法也分为两部分。客户端在接收到服务器的请求后首先执行服务器返回的远程操作，然后执行副本补全操作。接着客户端需要隐藏副本的第一个和最后一个有效节点来构造虚拟的 Head 和 Tail（和全副本保持一致），这样能够确保下一次操作同步前客户端执行的操作是合法的。另外，正是由于两次操作同步之间操作执行一定在上一次同步的副本范围内。

5.4.2 副本同步协议控制算法

部分副本同步算法是客户端发送查看操作所执行的同步算法，查看操作发出后客户端就不能响应客户的操作请求，并且客户端需要先和服务器完成一次操作同步然后再执行部分副本同步。部分副本同步分为两种情况，第一种情况为客户端已经对当前所持有的部分副本编辑完毕，想要切换到新范围内的部分副本。第二种情况为客户端不想抛弃现有的副本并向服务器请求向后连续的副本。对于第二种情况和操作同步类似客户端需要从服务器得到新的副本然后拼接在当前副本的后面。接着分别给出部分副本同步的服务器与客户端控制算法：

```
算法 12 Replication Control Algorithm in PRServer (OW, R, ART, CNUM)
参数：客户端上查看操作 OW，
      服务器副本 R,
      服务器地址范围表 ART,
      当前请求的客户端编号 CNUM
返回：需要同步到客户端的部分副本
1.O = R.DOC[OW.S]
2.ART[CNUM].S = OW.S
3.For i in 1 to L
4.   将 O 加入 REP
5.   If i == L Then
6.      ART[CNUM].E = O.UI
```

```
7.  End If
8.  O = O.NEXT
9.End For
10.Return [REP]
```

在算法 12 中，对于用户发起的查看操作，首先根据哈希表找到对应的节点。然后依次扫描 L 个节点，并且将节点的数据保存返回给客户端。另外，在扫描的过程中需要更新地址范围表来支持客户端后续的同步操作。

```
算法 13 Replication Control Algorithm in PRClient (TYPE, R)
参数：副本同步类型  TYPE，
      TYPE 为 REPLACE 或者 APPEND
      前者替换当前副本后者拼接在其后
      本地副本  R
1.(REP) <-  向服务器发送操作同步请求
2.If TYPE == REPLACE Then
3.  INIT(R)
4.End If
5.For I in 1 to |REP|
6.  APPEND O IN R
7.End For
```

对于客户端副本同步的控制算法来说，在发起查看操作时客户端需要记录用户发起的操作类型以 TYPE 变量输入，当用户需要替换副本时需要先初始化本地副本的结构，接着用户将同步收到的副本拼接在当前副本的后面。

5.5 复杂度分析

本章展现了部分复制技术在地址空间转换算法中的应用，通过副本同步协议的支持与操作同步协议的改进客户端能够方便的切换当前所编辑的副本。由于客户端所存储的为部分副本，因此客户端的执行效率也发生了改变，本小节对此进行讨论。

首先对于单个操作执行效率，并没有产生变化由于基于唯一标识符的设计单个操作对于副本的长度没有依赖。而对于一次操作同步来说，由于增加了副本拼接的部分因此工作量有所增加，然而副本拼接只是补全了原有缺失的无效节点并

不会跨越有效节点，因此副本拼接的效率是正比于两个有效节点之间的无效节点数 m 的。接着给出单次操作同步的效率，假设操作同步中客户端的本地操作的数量为 n，远程删除操作数量为 rnd，远程插入操作数量为 rni。一次操作同步的开销为 O(n+rni+rnd*m+m),另外由于副本范围的限制，对客户端来说远程操作中不在副本范围内的操作不会执行，因此采用了部分复制技术后不仅节省了客户端的内存，客户端的负载也变小了。

对于副本同步，服务器每次都需要扫描文档结构将节点返回给客户端同时客户端将收到的节点加入自己的副本中，显然一次副本同步的时间开销为 O(|R|),也就是和需要同步的部分副本成正比。

5.6 讨论

部分复制技术思想的魅力不仅仅局限于其本身，还能从以下两方面来体现。其一，与中心化地址空间转换算法系统应用优化的整合。其二，算法思想在其他协同编辑算法中的应用。以下我们来详细讨论：

本章所讨论的部分复制思想是应用来算法层面的优化，其主要的方式是通过在服务端增加一些额外的数据结构来维护客户端存储的副本范围，以此使得客户端仅仅需要存储部分副本。最终达到大幅减少客户端内存、带宽使用量的优化效果。而对于一些使用地址空间转换算法的应用，可以通过将底层数据结构与算法的替换来达到优化效果。Hydra 是采用协同编辑技术的移动评论系统，系统中应用树型数据结构来维护用户评论，并且在骨架树的基础上设计了一套部分复制技术，其思想和本章所介绍的内容有异曲同工之妙。在 Hydra 的底层数据结构中同一层级的用户评论节点的先后顺序采用中心化的地址空间转换算法来维护，仅仅采用骨架树技术会在操作同步过程中将所有同一层级的信息从服务器同步下来，这是算法层面 AST 的不足之处，也使得 Hydra 在处理层级较少，宽度很宽的评论树结构上效率较低。而将本章提出的支持部分复制技术的地址空间转换算法替换原文中中心化地址空间转换算法之后，可以将统一层级的节点宽度限制在一个可控范围内，解决了原有算法存在的问题。

另一方面部分复制思想不仅仅能够采用在中心化的地址空间转换算法中，一切中心化的协同编辑算法都适用。以 CRDTs 中性能最好的 RGA 为例，如果整个 P2P 环境中存在一个固定节点维护全副本，其他节点也可以通过在固定节点上增加附加数据结构的方式来维护副本范围，也可以通过设计副本同步协议来向固定节点更新副本，具体的实现方式将不在本文中具体探讨，留给感兴趣的读者作为

思考。

5.7 本章小结

本章介绍了部分复制技术在基于全局唯一标识符的中心化地址空间转换算法中的应用。主要提出了了 ART 和 Tag 两种支持部分复制技术的数据结构，设计了全新的操作同步协议并且支持了副本补全，并提出了副本同步协议并且支持增量式的构建客户端副本。部分复制技术以服务端增加一小部分内存为代价换来了，客户端内存消耗的大幅缩减和操作执行效率的提升。

另外，本章最后对于复杂度进行了分析，并且结合部分复制的特点与算法的应用范围作了详细的讨论。相信本章的工作只是部分复制思想在协同编辑领域应用的一个先例，之后会有越来越多的工作将其应用进来。

第六章 实验设计、评估与分析

本文介绍了两种关于地址空间转换算法的改进，第一种算法是应用在 P2P 结构下的支持字符串操作的 ASTS，第二种是中心化结构下的部分复制技术在 AST 中的应用。本章节将分成两个部分介绍，改进后的算法模型的实验设计和结果分析。

6.1 ASTS 算法的实验与分析

6.1.1 实验设计与分析

对于 ASTS 算法本文在单机上通过多线程和发送/接受队列模拟了 P2P 环境下多个客户端互相发送操作并执行的场景。实验的目的是通过由操作发生器大量产生操作，测量操作在每个客户端使用不同算法模块的执行时间。从而得出 ASTS 算法的性能和 AST 算法性能的比较。

单个客户端包含 Generator、Sender、Executor，Queue，AST Algorithm 几个模块。本地操作由 Generator 产生后发送给 Executor 和 Out Queue，前者调用 AST 算法模块直接执行，后者会将操作传递给 Sender，并由 Sender 广播到其他远程站点。远程操作则会被 In Queue 接收，然后直接送给 Executor 并调用 AST 算法模块执行。ASTS 算法的实现结构如图 6.1 所示:

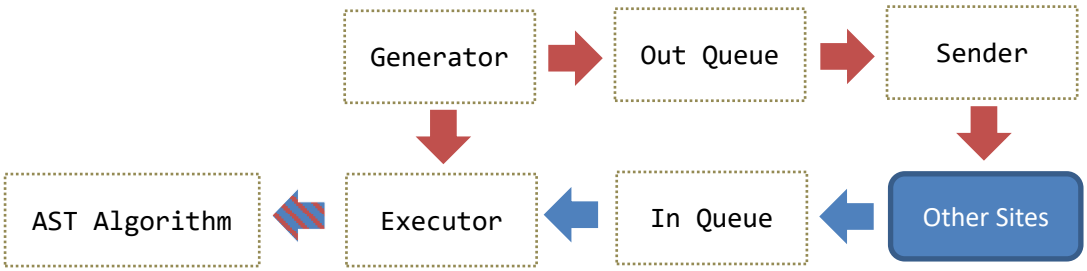

图 6.1 ASTS 算法实现结构图，其中蓝色箭头代表远程操作的传递，红色操作代表本地操作的传递。

AST 算法模块包含了多种算法的实现，为了比较不同场景下 ASTS 和 AST 的性能本次实验一共实现了四种算法，分别为：原始 AST[41]（AST），平衡树

优化 AST[41]（AST_Tree）、原始 ASTS（ASTS）、平衡树优化 ASTS（ASTS_Tree）。对于原始 AST 和平衡树优化的 AST 算法细节请参考[41]，在此将不再详述。表格中展示了四种算法对应的操作执行复杂度：

表格 2 四种算法的算法复杂度比较

	复杂度	
算法	**插入操作**	**删除操作**
AST	O(n + m)	O(n)
AST_Tree	O(d*h + h*log(n) + log(n) + h + d + m)	O(d*h + h*log(n) + log(n) + h + d)
ASTS	O(n*d/b + m/b + d)	O(n*d/b + d)
ASTS_Tree	O(d*h + h*log(n/b) + log(n/b) + m/b + h + d)	O(d*h + h*log(n/b) + log(n/b) + h + d)

在表格中 n 表示文档的长度，m 是 Rangescan 算法的扫描范围一般视作一个常数，d 表示单个节点下被附加的操作一般情况下也为常数，h 为和插入操作并发的操作个数，b 是节点中包含字符串的平均长度。从复杂度分析来看，当 b 大于 1 时，无论是原始 ASTS 还是平衡树优化后的 ASTS 效率都要好于对应的 AST 算法，稍后的实验也会展示 b 等于 1 时，ASTS 相比于 AST 损失的性能也极小。

相比较 ASTS，OT 类算法在字符串环境下的特点有所不同，其中支持字符串操作的最新算法 ABTS，本地操作执行的复杂度为 O(|Old|*|O.str|)，远程操作执行的复杂度为 O(|Hi|^2+(|sqc|+|Old|)*|O.str|)，其中 O 为操作，Old 为历史删除操作所对应的子操作，sqc 为与 Hi 并发产生的操作。仅仅从操作执行的复杂度来看 ASTS 和 ABTS，都依赖于操作节点中包含的字符串长度，并发操作的数量，不同的是前者依赖于文档的长度，而后者依赖于操作队列的长度。因此在不同的应用场景下，ASTS 和 ABTS 具有不同的特点和应用价值。

6.1.2 实验评估与结果分析

本次实验中整个算法实现都采用 JAVA 并运行于 Dell OptiPlex 980 个人电脑上，CPU 为 i3 530 2.93Hz，8GB 内存，Win7 企业版操作系统。在实验中采用两个线程模拟单个站点，一个线程负责执行 Generator 和 Sender 模块，另一个线程负责执行 Executor 模块。在实验中的设计忽略了操作传输的时间，站点个数就和操作执行的时间互不关联，因此本次实验中的站点个数设定为 2。对于每个站点操作是随机产生的，Generator 保证了每个操作的合法性并且操作的执行位置是符合均匀分布的。

在本文的实现中采用了 SBT（节点大小平衡树）来替代红黑树作为平衡树优化算法的基础数据结构，SBT 在随机的操作集上比红黑书具有更加出色的性能[]，并且本文在单字符环境下和字符串环境下分别比较了四种算法的性能。

第一个实验比较了在单字符环境下四个算法的性能，本文分别观察了不同操作数量下操作的执行时间，并且针对不同的插入删除操作比做了多次实验，通过实验结果可以观察到 AST 和 ASTS 操作执行时间非常的接近，ASTS_Tree 比 AST_Tree 有微小的性能损失。并且随着操作执行的数量增大，文档长度也变的越来越长，因此在 AST 和 ASTS 中操作执行时间会随着操作数目的上升而变长，而对于 AST_Tree 和 ASTS_Tree 则没有什么影响。

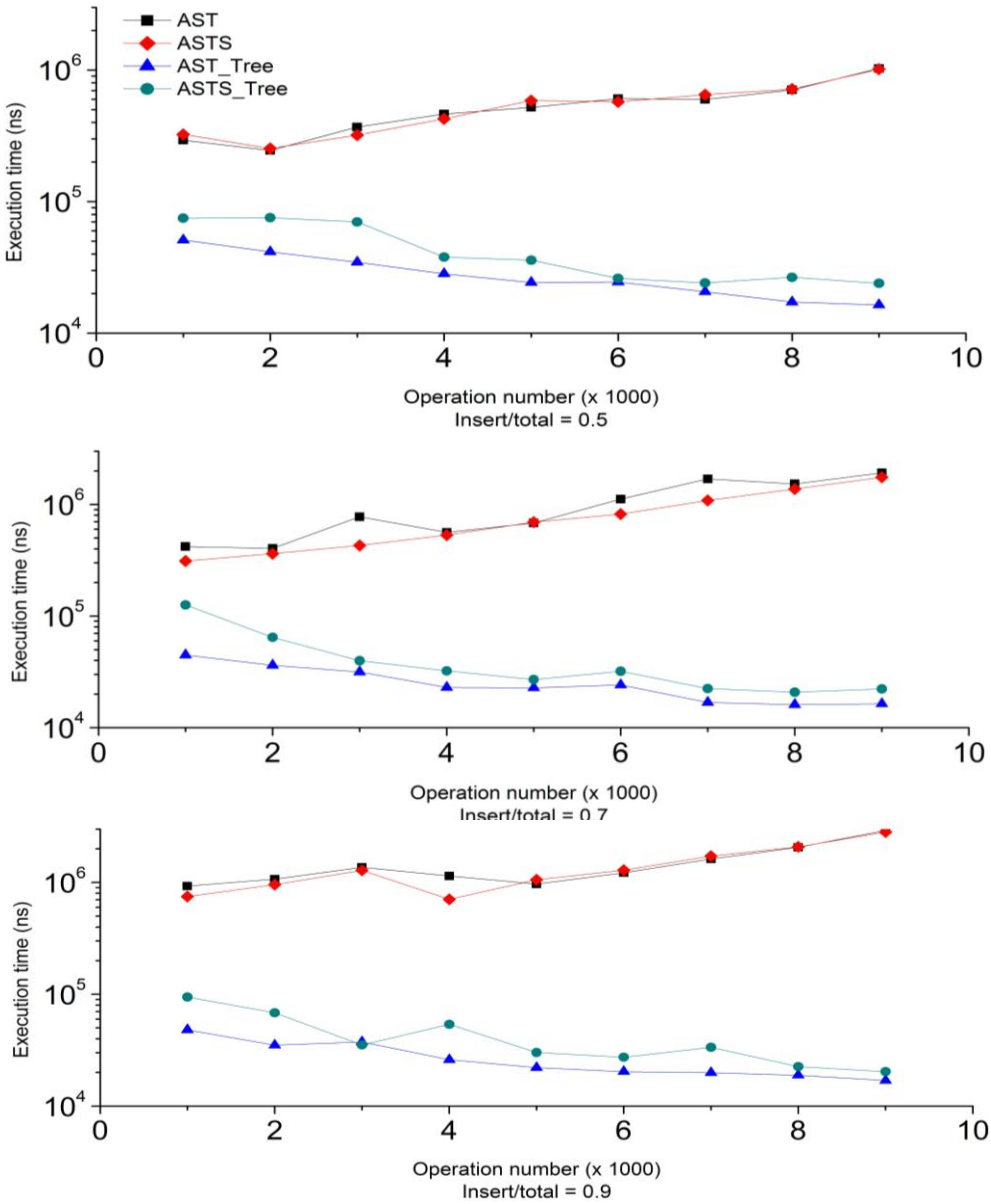

图 6.2 单个字符环境下各个算法的性能比较

第二个实验比较了 AST 和 ASTS 在字符串操作环境下的性能，对于不同的字符串长度 10，20，30 分别进行了三次实验，同样本文控制了不同的操作数量分别对每个实验运行了 9 次。在图中采用算法名_平均操作大小来表示一个算法在当前操作大小下的性能。通过结果可以观察到无论是否加入平衡树优化，在字符串操作环境下 ASTS 的效率都要明显高于 AST，并且随着平均操作长度的提升两者之间的差距也越来越大。

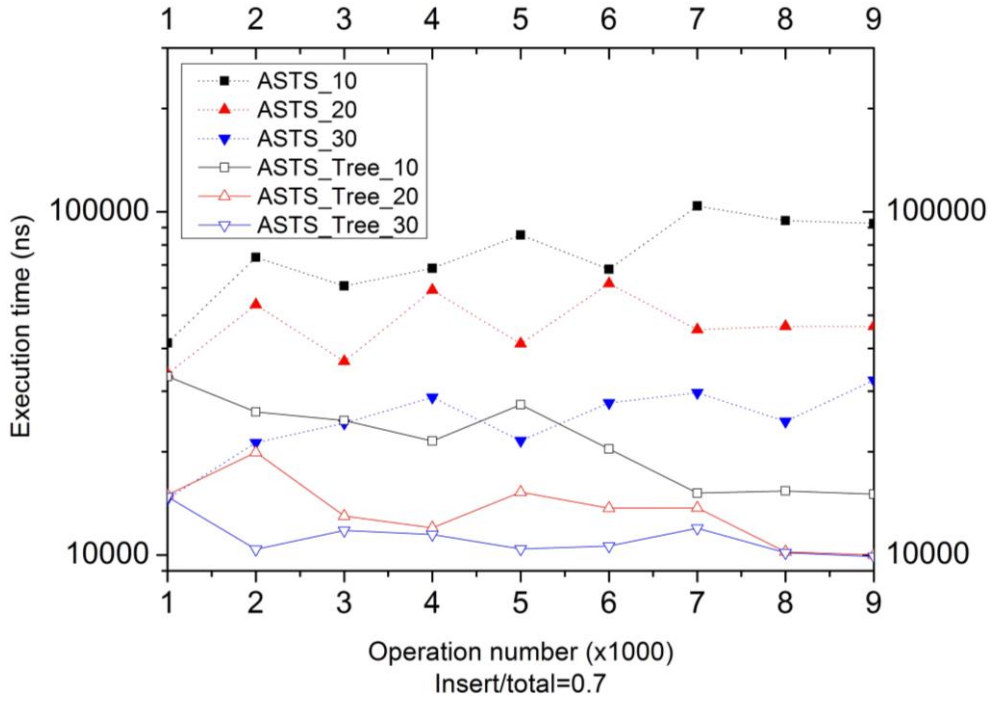

图 6.3 字符串环境下各个算法的性能比较

6.2 部分复制的实验与分析

6.2.1 实验设计与分析

部分复制采用的是服务器和客户端的形式，实验中同样采用多线程和发送/接受队列模拟了客户端和服务器交互的过程。实验通过在客户端的操作发生器执行大量操作，同时检测在不同参数设置下，客户端的操作执行时间、服务端 QPS、客户端和服务端的内存使用情况。

对于单个客户端中，包含 AST 算法模块、tag 列表维护模块，对于服务端则包含 AST 算法模块、tag 集合、地址范围表 ART、操作临时队列，另外还包含一个通信模块。其中客户端和服务器的传递消息都会现将消息发给一个中介者（通

信模块），再由中介者转发给目标对象。

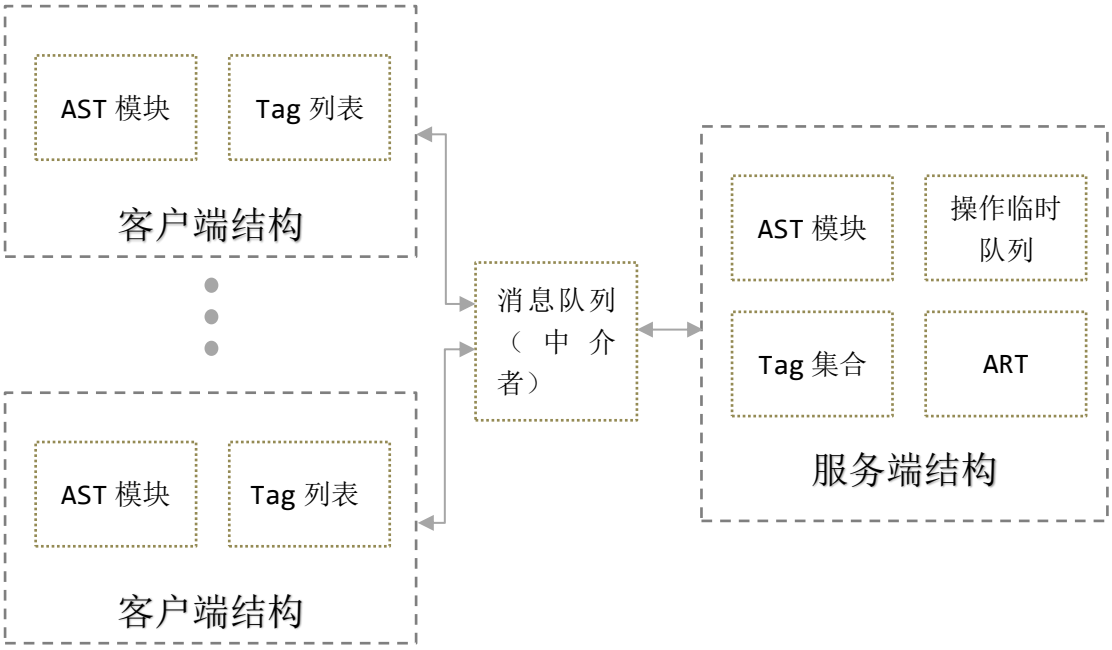

图 6.4 部分复制实验结构图

6.2.2 实验评估与结果分析

本次实验中整个算法实现都采用 C++，运行于 Ubuntu 16.04 操作系统的虚拟机上，虚拟机的配置为：CPU 4 核，4G 内存。实验中采用两个线程模拟一个客户端，其中一个线程负责产生和执行本地操作，另一个线程负责和服务器的交互。服务端则由单独一个线程模拟，该线程会定时访问中介者队列处理并返回同步请求。

整个实验过程分为三部分，第一部分通过操作发生器生成随机操作来比较全复制模式下和部分复制模式下客户端操作的响应速度，第二部分通过操作发生器生成随机操作得出服务器在不同客户端个数下支持的 QPS，第三部分则通过操作发生器不断的向本地副本中插入随机字符来检测客户端和服务端在有部分复制和没有部分复制下分别的内存使用情况。通过前一章的理论分析可以知道，在 AST 算法中插入操作产生新节点而删除操作只会产生新操作，因此对于全是插入操作的测试集，无论是客户端的内存消耗还是服务端的内存消耗都是大于其他操作集合的。

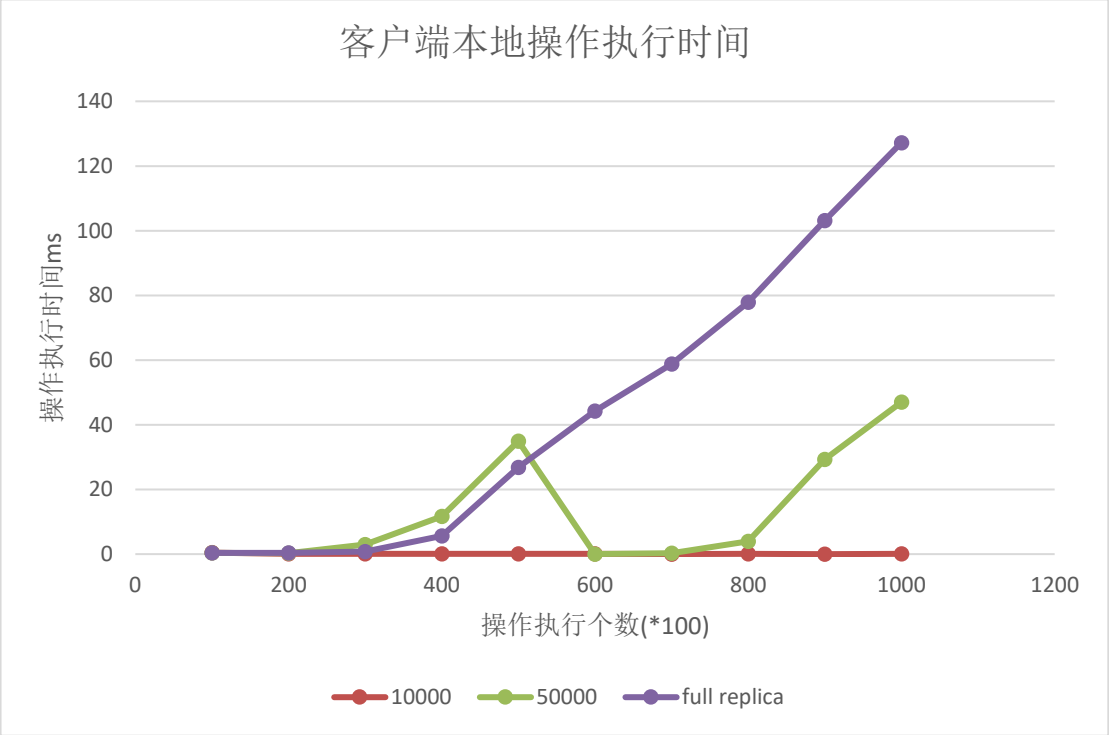

图 6.5 不同副本下的客户端本地操作执行时间

实验的第一部分比较了在不同副本大小下客户端本地操作执行的时间，通过实验结果可以观察到在没有部分复制技术的情况下，操作的执行速度会随着文本的增长而增长。而通过部分复制技术能够将客户端文本的长度控制在一个有限范围内，因此客户端操作执行的时间相比较全副本情况下有明显提升，并且客户端持有的副本越小操作执行的时间越短。

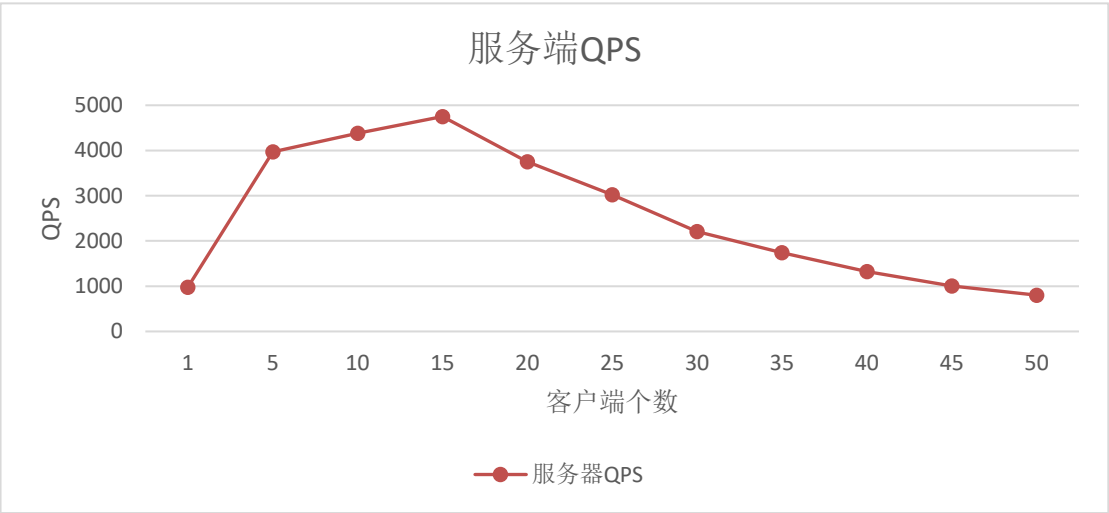

图 6.6 不同客户端数量下的服务端 QPS

实验的第二部分则是比较了在不同客户端数量的情况下，服务器的 QPS，每秒钟能够处理的操作数量。通过实验结果可以看到，在客户端较少的情况下服务器的资源没有使用饱和，因此 QPS 较低，随着客户端的数量上升 QPS 逐渐升高。当客户端的数量大于 20 的时候服务器的资源使用饱和，因此 QPS 也逐渐降低。

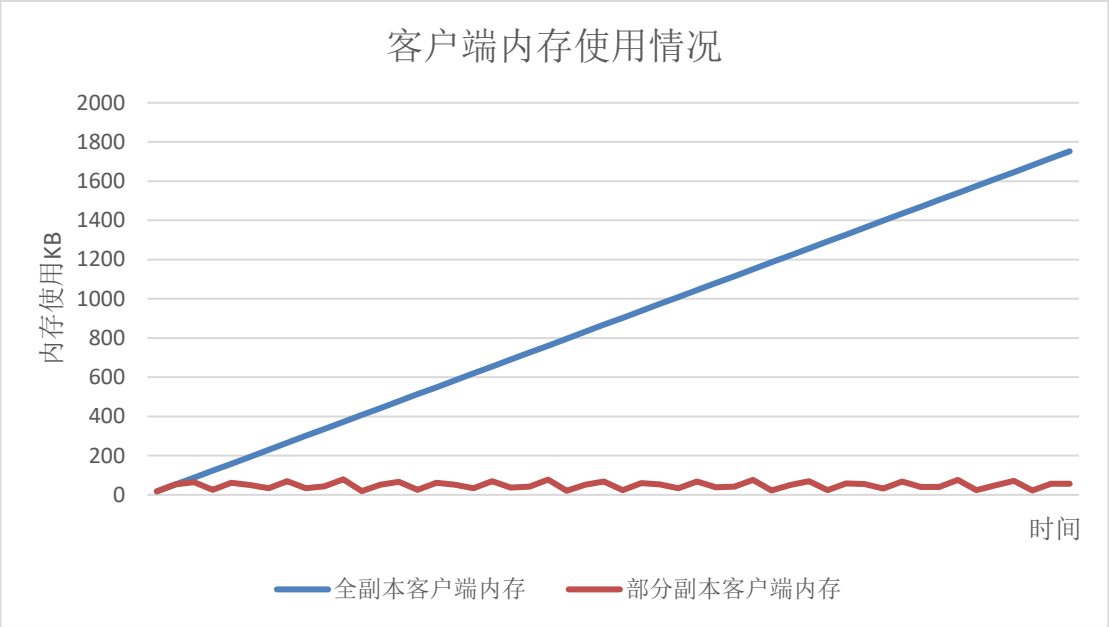

图 6.7 全副本和部分副本下客户端的内存使用情况

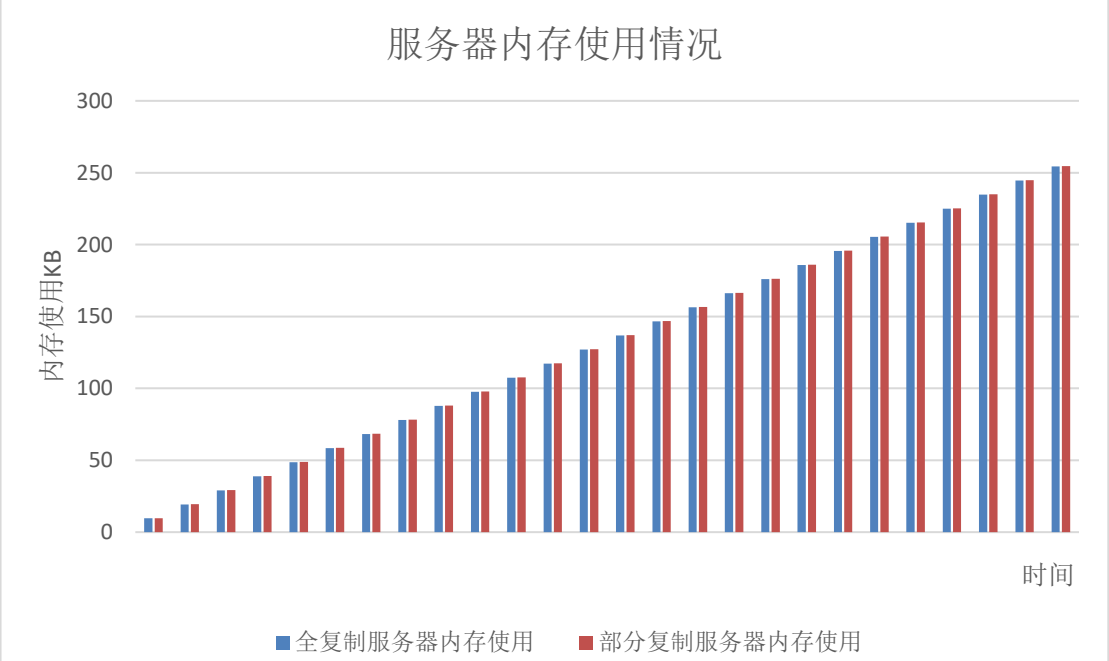

图 6.8 全副本和部分副本下服务器的内存使用情况

实验的第三部分通过生成连续的插入操作，分别测试客户端和服务端在部分副本和全副本下内存的使用情况。通过实验结果可以看到，对于客户端随着操作的执行文本的长度会越来越长，内存的使用随着时间越长大致呈线性增长。在加入了副本同步之后，客户端的副本长度一旦大于 5000 则执行一次副本同步，客户端的内存使用一直控制在 100KB 下。因此，使用部分复制技术能够大大的节省客户端内存的使用。对于服务器全副本和部分副本的内存使用结果，可以发现虽然部分副本会在服务器多存储 ART，Tag Set 等信息。但是相比较副本长度来说这些信息占用的内存都足够的小，因此部分复制技术并不会对服务器产生很大的内存负担。总体来说，在部分复制技术下服务器以很小的内存代价换来了客户端在内存上很大程度的节省。

6.3 小结

本章主要介绍了 AST 算法优化的实验设计与分析，第一部分介绍了 ASTS 的实验设计和复杂度分析，通过实验结果得出了 ASTS 在字符串环境下对于操作的执行速度有明显的提升，并且在单个字符环境下并没有很大的性能损失。第二部分介绍了部分复制的实验设计，通过实验结果得出了部分复制能够大幅提高客户端的性能包含操作执行速度和内存使用，同时部分复制技术相比全复制给服务器增加的内存负担几乎可以忽略不计。

第七章 总结与展望

7.1 总结

在当今的社会环境下，计算机科学的发展正在推动着人类社会的进步，一方面软件种类越来越多，另一方面硬件设备越来越廉价。这给寻常百姓能够使用更多新型电子设备提供了条件，这进一步导致了用户对于应用的用户体验要求越来越高，和应用对于底层算法的要求越来越高。在协同编辑领域中，不断的优化算法的性能和将算法应用在更多的领域成为了当前发展的两大趋势。本文的主要工作集中于地址空间转换算法的优化技术中，提出了两种全新的改进算法使得地址空间转换技术在更多更广的领域中应用得到可能。

总结来说本文在地址空间转换算法优化上提出的贡献主要包括以下三个部分：

1. 提出了支持字符串操作的地址空间转换算法 ASTS。通过在原有地址空间转换算法模型上进行改进并且引入了节点分裂的概念，从模型上支持了字符串操作。提出了 ASTS 支持 Undo 的方法，在支持字符串操作的基础上设计了模型层面的 Undo 操作支持。改善了字符串环境下地址空间转换算法的性能。
2. 提出了部分复制技术在中心化地址空间转换算法中的优化方法。通过对于原本中心化地址空间转换算法操作同步协议的改进并引入全新的副本同步协议，使客户端能够在仅仅存储部分副本的情况下进行协同编辑。不仅仅节省了客户端的内存开销，也增加了客户端操作响应速度。在提升用户体验的同时，节约了宝贵的硬件资源。
3. 对于以上优化算法通过多线程技术完成模拟实验，验证了本文的理论推导。其一，通过对于 ASTS 和 AST 的比较得出了 ASTS 在字符串操作环境下的性能有显著的提升，同时在单字符环境下性能没有太大损失，符合理论预期。其二，通过中心化环境中采用部分副本和全副本的比较，验证了部分复制技术下，地址空间转换算法能够大幅节省客户端的内存，大幅增加客户端的操作响应速度，并且给服务器增加的代价近乎可以忽略不计，符合理论预期。

7.2 展望

本文的研究工作主要包含了地址空间转换算法优化层面的工作，而基于这些工作还有很多值得人们深入挖掘下去的地方，接下来从多个方面来切入讨论：

1. ASTS 算法在其他领域的应用与拓展。本文对于 ASTS 的研究是假定编辑对象都是线性文本的情况下的，而在现在的协同编辑场景下对象可以是多种多样的数据结构。因此，是够可以将合并连续区间操作打包执行的概念应用到其他数据结构中是第一层面值得探索的地方，进一步如何在支持连续区间操作打包的情况下保证算法的高效则是第二层面值得探索的地方。
2. 部分复制下的服务器的扩展问题。本文提出的部分复制技术能够大大改善客户端的性能，而从实验中可以看到服务器在存在大量客户端同时编辑的情况下性能会有所下降。因此，如何通过服务器的横向扩展或者多线程的支持来改善这个问题是另一个值得探索的地方。
3. 部分复制算法在其他领域的应用与扩展。部分复制算法的提出给地址空间转换算法在中心化系统中不同硬件终端上的使用增加了可能性。智能手表，嵌入式设备这些对资源利用率要求极高的设备都有可能使用地址空间转换算法。对于这些设备中的协同应用开发值得深究与讨论。
4. 部分复制技术向其他协同编辑算法的普及。本文提出的部分复制技术其实是一种算法思想，其不仅仅可以应用在地址空间转换算法中。对于协同编辑领域的其他中心化结构算法也可以应用，对于使其它协同编辑算法支持部分复制技术也是一个新的值得探索的课题。

参考文献

[1] Ellis, C. A. "Concurrency control in groupware systems." Acm Sigmod Record 18.2(1989):399-407.

[2] Ressel, Matthias, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. "An integrating, transformation-oriented approach to concurrency control and undo in group editors." Proceedings of the 1996 ACM conference on Computer supported cooperative work. ACM, 1996.

[3] Sun, C., et al. "A Consistency Model and Supporting Schemes for Real-time Cooperative Editing Systems." Theoretical Computer Science 1996.

[4] Sun, Chengzheng, and Clarence Ellis. "Operational transformation in real-time group editors: issues, algorithms, and achievements." Proceedings of the 1998 ACM conference on Computer supported cooperative work. ACM, 1998.

[5] Li, Du, and Rui Li. "An admissibility-based operational transformation framework for collaborative editing systems." Computer Supported Cooperative Work (CSCW) 19.1 (2010): 1-43.

[6] Oster, Gérald, et al. "Data consistency for P2P collaborative editing."Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. ACM, 2006.

[7] Weiss, Stéphane, Pascal Urso, and Pascal Molli. "Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks."Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on. IEEE, 2009.

[8] Preguica, Nuno, et al. "A commutative replicated data type for cooperative editing." Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on. IEEE, 2009.

[9] Roh, Hyun-Gul, et al. "Replicated abstract data types: Building blocks for collaborative applications." Journal of Parallel and Distributed Computing71.3 (2011): 354-368.

[10] Yu, Weihai. “A string-wise CRDT for group editing.” ACM International Conference on Supporting Group Work 2012, pp. 141-144, 2012.

[11] Yu, Weihai. “Supporting String-Wise Operations and Selective Undo for Peer-to-Peer Group Editing.” International Conference on Supporting Group Work 2014, pp. 226-237, 2014. (String-Undo-WOOT)

[12] Andre, L., et al. “Supporting Adaptable Granularity of Changes for Massive-scale Collaborative Editing.” IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing 2013,pp. 50-59, 2013.

[13] Urso, Pascal, and M. Shapiro. "High Responsiveness for Group Editing CRDTs." International Conference on Supporting Group Work ACM, 2016:51-60.

[14] Lv, Xiao, et al. "An efficient collaborative editing algorithm supporting string-based operations." IEEE, International Conference on Computer Supported Cooperative Work in Design IEEE, 2016:45-50.

[15] Lv, Xiao, et al. "A string-wise CRDT algorithm for smart and large-scale collaborative editing systems ☆." Advanced Engineering Informatics33(2016).

[16] Ahmed-Nacer, Mehdi, et al. "Evaluating CRDTs for real-time document editing." DocEng 2011 - Proceedings of the 2011 ACM Symposium on Document Engineering (2011):103-112.

[17] Roh, Hyun Gul, J. Kim, and J. Lee. "How to Design Optimistic Operations for Peer-to-Peer Replication." Joint Conference on Information Sciences, Jcis 2006, Kaohsiung, Taiwan, Roc, October DBLP, 2006.

[18] Nicolaescu, Petru, et al. "Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types." International Conference on Supporting Group Work ACM, 2016:39-49.

[19] Sun, Chengzheng, et al. "Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems." ACM Trans. Computer-Human Interaction 5.1(1998):63--108.

[20] Weiss, Stéphane, P. Urso, and P. Molli. "Wooki: A P2P Wiki-Based Collaborative Writing Tool." International Conference on Web Information Systems Engineering – WISE Springer-Verlag, 2007:503--512.

[21] Nicolaescu, Petru, et al. "Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types." Engineering the Web in the Big Data Era, International Conference, ICWE 2015:675-678.

[22] Davis, Aguido Horatio, C. Sun, and J. Lu. "Generalizing Operational Transformation to the Standard General Markup Language." ACM Conference on Computer Supported Cooperative Work ACM, 2002:58-67.

[23] Yang, Jiangming, et al. "Lock-free consistency control for web 2.0 applications." International Conference on World Wide Web, WWW 2008, Beijing, China, April 2008:725-734.

[24] Xia, Huanhuan, et al. "A partial replication approach for anywhere anytime mobile commenting." ACM Conference on Computer Supported Cooperative Work & Social Computing 2014:530-541.

[25] Ng, Agustina, and Chengzheng Sun. "Operational Transformation for Real-time Synchronization of Shared Workspace in Cloud Storage." Proceedings of the 19th International Conference on Supporting Group Work. ACM, 2016.

[26] Shao, Bin, D. Li, and N. Gu. "ABTS: A transformation-based consistency control algorithm for wide-area collaborative applications." International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2009. COLLABORATECOM IEEE, 2009:1-10.

[27] Sun, Chengzheng, Hongkai Wen, and Hongfei Fan. "Operational transformation for orthogonal conflict resolution in real-time collaborative 2d editing systems." Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work. ACM, 2012.

[28] Sun, Chengzheng. "Dependency-conflict detection in real-time collaborative 3D design systems." Proceedings of the 2013 conference on Computer supported cooperative work. ACM, 2013.

[29] Sun, David, and C. Sun. “Operation context and context-based operational transformation.” ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, pp. 279-288, November 2006.

[30] Liu F, Xia S, Shen H, et al. CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools[C]//Proceedings of the 2008 ACM conference on Computer supported cooperative work. ACM, 2008: 5-8.

[31] N. Gu, J. Yang, Q. Zhang, and B. Shao. The CoAutoCAD project. http://cscw.fudan.edu.cn/coautocad/, 2005.

[32] Gu, Ning, Jiangming Yang, and Qiwei Zhang. "Consistency maintenance based on the mark & retrace technique in groupware systems." Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work. ACM, 2005.

[33] Ahmed-Nacer, Mehdi, et al. "Evaluating CRDTs for real-time document editing." DocEng 2011 - Proceedings of the 2011 ACM Symposium on Document Engineering (2011):103-112.

[34] Yang, Dayi, et al. "Making itinerary planning collaborative: An AST-based approach." Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on. IEEE, 2016.

[35] Gu, Ning, et al. "Dcv: a causality detection approach for large-scale dynamic collaboration environments." Proceedings of the 2007 international ACM conference on Supporting group work. ACM, 2007.

[36] Zhang J, Lu T, Xia H, et al. ASTS: A string-wise address space transformation algorithm for real-time collaborative editing[C]. IEEE, International Conference on Computer Supported Cooperative Work in Design. IEEE, 2017:162-167.

[37] Nichols, David A., et al. "High-latency, low-bandwidth windowing in the Jupiter collaboration system." Proceedings of the 8th annual ACM symposium on User interface and software technology. ACM, 1995.

[38] Shao, Bin, et al. "An operational transformation based synchronization protocol for web 2.0 applications." Proceedings of the ACM 2011 conference on Computer supported cooperative work. ACM, 2011.

[39] Ramasubramanian, Venugopalan, et al. "Cimbiosys: A platform for content-based partial replication." Proceedings of the 6th USENIX symposium on Networked systems design and implementation. 2009.

[40] Puttaswamy, Krishna P. N., et al. "Docx2Go: collaborative editing of fidelity reduced documents on mobile devices." International Conference on Mobile Systems, Applications, and Services ACM, 2010:345-356.

[41] 顾宁，杨江明，张琦炜．协同组编辑中基于地址空间转换的一致性维护方法[J]．计算机学报, 2007, 30(5):763-774.

[42] 杨江明．协同组编辑环境中的数据一致性维护方法[D]．复旦大学, 2007.

[43] 张琦炜．大规模协同环境下的实时组编辑技术研究[D]．复旦大学, 2007.

[44] 邵斌．高效的操作转换一致性维护方法研究[D]．复旦大学, 2010.

[45] 夏欢欢．移动云环境下地址空间转换关键技术研究及应用[D]．复旦大学, 2014.

[46] 杨达一. 基于AST 的协同旅游路线规划算法研究[D]. 复旦大学, 2016.

[47] Google Doc https://docs.google.com

[48] IBM Open Coweb http://opencoweb.org/

[49] Codoxware http://www.codoxware.com

[50] Wiki http://en.wikipedia.org/

[51] SBT http://web.stanford.edu/~cqf/SBT.zip

[52] One Drive https://office.live.com/start/OneDrive.aspx?omkt=en-US

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] Zhang J, Lu T, Xia H, et al. ASTS: A string-wise address space transformation algorithm for real-time collaborative editing[C]// IEEE, International Conference on Computer Supported Cooperative Work in Design. IEEE, 2017:162-167.

攻读硕士期间参与的科研项目

[1] 国家自然科学基金重点项目，面向社会秩序的社会计算理论和方法研究（No. 61332008）

[2] 国家自然科学基金 NSAF 基金， 面向多学科协作的数据世系建模及溯源关键问题研究。（No. U1630115）

致谢

时光飞逝，三年前背着书包到实验室参观和师兄们打球的场景仿佛历历在目，而此时此刻我却已经坐在实验室撰写着自己的毕业论文了。时光带走的是岁月，带来的是更加成熟的心智和进入社会海洋的那一件救生衣。三年间，遇到了许多人也经历的许多事，大部分都成为了人生中的匆匆过客而其中给予我知道帮助的老师和前辈会永远的留在心中，在这里我要借此机会表达我的感激之情！

首先要感谢的是我的导师卢暾老师，卢暾老师有着专业的学术眼光和敏锐的洞察力，带领我攀登一座又一座的山峰。在这三年里，卢暾老师通过不断地悉心教导让我在实践中得出真知。同时，卢暾老师更是以身作则常常加班到深夜，对待我们学生呕心沥血，这样的精神也深深影响到了我，成为了我在汪汪大海中航行的一盏灯塔。在此我要向卢老师表示最深的谢意！

在过去的三年学校生活中我的成长也离不开实验室的顾宁、丁向华、刘铁江老师。顾老师是协同领域的专家，正是他的这种专家精神影响着我们实验室一代又一代的学生。丁老师则有着丰富的学术积累，为实验室不断的添砖加瓦。刘老师则对我们学生的生活无微不至的关心，让我们时刻在健康的环境下快乐的成长。如今我所得到的成就和实验室的几位老师的辛勤付出息息相关，在这里我要对老师们说声真挚的“谢谢”！

另外，毕业论文和学术论文的完成离不开许多师兄的帮助，包括远在 MSRA 的邵斌师兄和夏欢欢师兄。两位师兄经常在百忙中抽出时间参与讨论，纠正指导我对于学术问题的理解。他们的关心和帮助成就了今天我的学术水平，在此表示感谢！

在复旦的三年里我还同样认识了许多同学，结识了不少朋友。丁海洋、王海明、石宛露、杜震坤、曾彬、唐祥轩，还有实验室的其他同学，正是由于你们平时的热心帮助让我迈过了一道又一道坎。在困难的时候我们互帮互助，项目完成了我们共同庆祝。感谢你们给我带来的美好时光，也祝你们今后前程似锦！

同样需要感谢的是我的家人们，从我开始读书时父母都给予我无微不至的照顾。家成为了我的避风港，如今我的扬帆起航离不开你们一直以来的关心和支持。在这里我要对你们说“爸妈辛苦了！”。

最后向百忙之后评审这篇论文的专家们致敬并报以真挚的感谢！

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：_____