OSH 详细设计报告

实时文本协作系统

徐直前 PB16001828
吴永基 PB16001676
黄子昂 PB16001840
金朔苇 PB16001696

2018 年 7 月 5 日

目录

1 项目概述 **2**

2 背景知识 **3**

2.1 实时文本协作的技术难点 3
2.2 CRDT 5
2.3 Socket.IO 7

3 详细设计 **8**

3.1 主体架构 8
3.1.1 权限管理的具体机制 9
3.2 基于状态的 CRDT 的设计 10
3.2.1 数据结构 10
3.2.2 API 12

1 项目概述

在本项目中，我们参考了有关 CRDT 的论文，用 JavaScript 实现了一个基于状态的 CRDT，并在此 CRDT 基础上用 JavaScript 和 Node.js 编写前后端，最终实现了一个基于网页端，轻量，能以类似 C 等语言中花括号的方式实现块级的权限控制，并且能够对 Markdown 语法支持以及实现实时预览，也具有一定的鲁棒性。其主要特征如下：

- 基于 CRDT 实现，具有很好的可伸缩性；
- 采用纯 JavaScript 编写，为网页端应用；
- 使用 Node.js 作为后端，利用 npm 包管理器可以很方便地完成部署；
- 支持 Markdown，并支持实时预览；
- 能实现精确到块级的权限控制，管理员可以指定某块内容只能由某个用户编辑，同时也可以指定一个公共的编辑区域；
- 具备高度可再开发性，基于本项目可以实现更高级的实时文本协作应用。

目前权限管理采用免登陆的方式，第一个登陆网页的人即为管理员（admin），此后登陆的人都为普通用户（guest）。仅有管理员拥有全部权限，可以控制普通用户的编辑权限。而普通用户不能进行权限的编辑。管理员权限也会动态转移。当当前管理员离线时，管理员权限会即可转移到第一个 guest 上。这种免登陆的权限控制实施方案极适用于一个团队在同一个局域网内进行实时协作。当然，也可以维护一个相应的用户数据库，实现用户名密码登陆的用户管理方式，此部分内容在本项目基础上很容易扩展，而且与实时文本协作的核心技术难题无关，故没有实现该种方式。

我们将本项目部署在了一台腾讯云服务器上，并且也在结题汇报时进行了现场互动演示，实现了很好的效果。

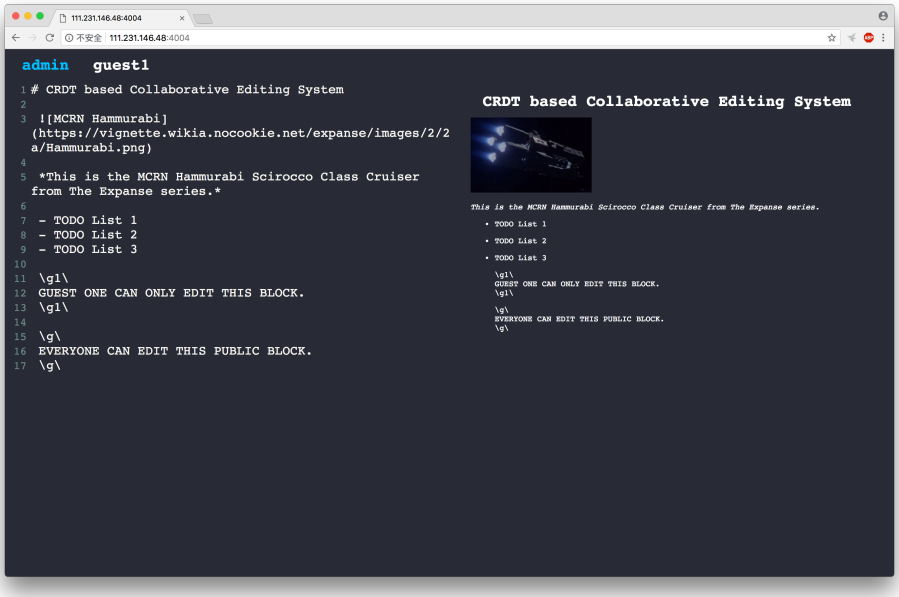

图 1: 最终实现效果

2 背景知识

本部分内容将介绍与此项目相关的背景知识，其部分内容在调研报告和可行性报告中也有所介绍，在此也对其中的关键内容进行一下回顾。

2.1 实时文本协作的技术难点

实时文本协作从用户层面来说看似十分简单，然而其面对的技术难题却是相当巨大的。业界在此问题上也进行了几十年的研究，最终也才在近些年来诞生了像 Google Docs 这样较为完善的实时文本协作系统。其主要的难点在于解决不同用户的数据副本之间的同步问题。

通常在这种高交互性的网络应用中，为了隐藏网络延时对用户体验带来的影响，我们要在每个客户端上保存一个本地的用户副本。每个用户的操作都直接在本地数据副本上执行，这样本地的操作就不会受到网络延时的影

响，用户能获得很好的本地响应性。然而，这样的方式也会带来问题。需要设计一种机制维持各个用户的数据副本之间的一致性。如果用户的副本出现了不一致的情况，那么显然会导致灾难性的后果。

我们用一个具体例子来说明其中的技术难题。假设现有 Alice 和 Bob 两人同时进行文档的编辑。初始状态下 Alice 和 Bob 各自的副本都是相同的，内容如下：

We wanted cars, instead we got characters.

此时 Alice 想在 *got* 后面插入 *140* ，同时 Bob 又想在 *wanted* 后面插入 *flying*。如果网络没有任何延时，所有的操作都能被瞬间应用，而 Alice 和 Bob 之间的操作在物理上也必然会有个先后关系，两者的操作会按照时间的先后关系而被执行，这里我们就不会遇到任何问题了。但是，由于网络有延时，那么问题就来了。Alice 和 Bob 两个人各自都先执行自己的本地编辑操作，Alice 先会进行一个 *insert(140, col=30)* 的操作，在第 30 个位置插入字符串 *140* ，同样 Bob 先会执行的操作是 *insert(flying, col=9)* 。此后，Alice 和 Bob 分别将自己进行的操作广播给对方 Alice 受到 Bob 的操作后然后执行 *insert(flying, col=9)* ，Bob 收到 Alice 的操作后执行 *insert(140, col=30)* 。

此时，对 Alice 来说一切正常，她得到的状态是：

We wanted flying cars, instead we got 140 characters.[1]

但 Bob 就糟糕了，此时 Alice 实际想要插入的位置就不是第 30 个位置了。Bob 的状态变成了：

We wanted flying cars, instead 140 we got characters.

显然，Alice 和 Bob 得到了不一致的结果，这显然是文本协作中不能容忍的。为此，我们的核心问题就是通过一种什么样的机制来保证每个用户都能得到相同的数据副本?

[1]Peter Thiel 的名句，讽刺了人类似乎点错了科技树，近些年来除了互联网领域有飞跃性的进展，我们的科技缺少从 0 到 1 的突破。

2.2 CRDT

为了解决上述问题，业界也展开了数十年的研究，很多种方案例如 AST（Address Space Transformation）、OT（Operational Transformation）、WOOT（WithOut Operational Transformation）、CRDT（Conflict-free Replicated Data Type）被相继提出。目前大多数主流文本协作应用例如 Google Docs、石墨文档用的均是 OT 技术。OT 的核心思想非常简单，当远程的操作到达本地站点后，其操作的上下文可能和相应的客户端发出该操作时不同了，为了正确的执行该操作，我们要在执行远程操作之前对其转换，使得在当前的上下文下执行操作仍然不改变原始操作效果。由于需要谨慎考虑所有可能出现的情况，OT 不具备较好的可伸缩性，同时实现起来也非常复杂。CRDT 则是近些年来新提出的一种技术，也正在逐渐被更加深入的研究和在产品中应用。

CRDT 有很多种不同的具体类型和实现，但大体上可以分为两类：基于状态的 CRDT 和基于操作的 CRDT。基于状态的 CRDT 在更新时会将副本的整个状态广播给其他副本。当一个副本收到了其他副本的状态时，会根据 merge 函数的机制和本地的状态进行 merge。而基于操作的 CRDT 则在每次更新时不广播整个副本的状态，而仅仅广播更新的操作，这样避免了整个状态过大不便于传输的问题。

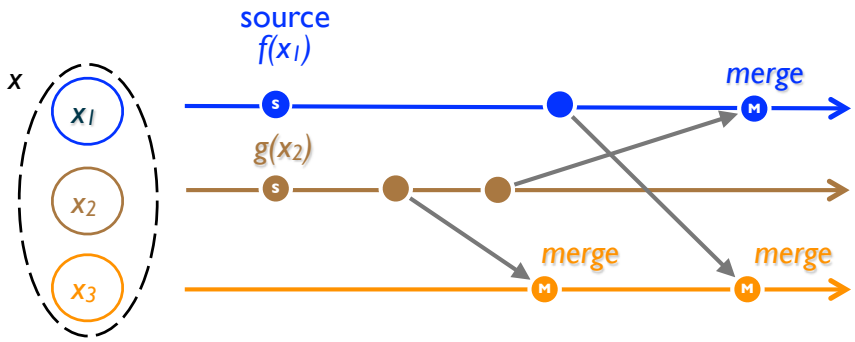

图 2: 基于状态的 CRDT

在该项目中我们采用了一种基于状态的 CRDT 实现文本协作，通过

给文本中的每一个字符一个独一无二的标识符，我们将一个文本文档转换成了一个有序的标识符序列。通过这个标识符序列，我们可以很轻松地维护各个副本之间一致性。CRDT 这种数据类型能够使得并发操作之间相互 commute。如果操作是以先行发生（happen-before）的顺序产生，那么 CRDT 的副本之间不需要复杂的并发控制就能自动收敛达成一致。

在文本实时协作中，我们给每一个原子（也就是每一个字符）赋予一个独一无二的位置标识符（PosID），满足一下三条原则

1. 每一个在缓冲区中的原子都有一个 ID
2. 任意两个不同的原子都有两个不同的 ID
3. 一个给定原子的 ID 在整个文档剩余的生命周期中保持不变
4. ID 之间有一个全序关系，和原子在缓冲区中的顺序一致
5. ID 取值空间是密集的，即：$\forall P, F : P < F \Rightarrow \exists N : P < N < F$

我们定义一个抽象的原子数据缓冲区的状态 T 是一个由 (*atom,PosID*) 二元组构成的集合，状态 T 的内容就是由 T 中所有原子按照 *PosID* 排列构成的序列。

每一个用户都会维持这个 CRDT 的一个副本，并且进行本地编辑

- $insert(PosID_n, \text{newatom})$
- $delete(PosID_n)$

这样，两个指向不同的 PosID 的操作是相互独立的。因为他们对于 CRDT 的作用效果是与他们的执行顺序无关的。那么现在只需要考虑并发的指向相同的 PosID 的两个操作。一个插入操作必定发生在一个删除操作之前签，所以他们不可能是并发的。最后，删除操作是幂等的（idempotent），因为删除掉了一个给定 PosID 的字符再次删除这个 PosID 的字符是无效的。因此，我们这样构建的一个缓冲区就构成了一个简单的 CRDT，实现了文本实时协作功能。

但是，在这里还有两个问题需要考虑。

- 两个客户端在同一时间生成了同样的 PosID（当他们并行地在同一个位置插入字符）
- 一个客户端生成了一个已经被使用过的 PosID（删除一个字符之后重新插入他们）

第二个问题很好解决，每个客户端自己只要维持一个记录，保证不会生成自己已经使用过的 PosID 即可。第一个问题同样可以用一个很简单的方法解决，我们只要将每一个 PosID 变成一个二元组即可，把 PosID 这个标识数字和产生该 ID 的客户端编号放在一起即可。这样就保证了两个客户端不会产生同样的 PosID。

2.3 Socket.IO

Socket.IO 是一个面向实时 Web 应用的 JavaScript 库，封装了 WebSocket 等协议实现了 TCP 客户端和服务器端全双工通信。Socket.IO 提供了一个相当高级的接口，其使用起来也极为简单。在客户端，只需要在 HTML 文件中引入 Socket.IO 脚本即可，只需要一行代码

```
<script src="/socket.io/socket.io.js"></script>
```

服务器端所需要的操作也很简单

```
var io = require("socket.io").listen(server);
```

就可以将 socket.io 绑定到服务器上，任何连接到该服务器上的客户端都具备了实时通信功能。

然后使用

```
io.on("connection", function (socket) {...}
```

就可以监听所有客户端，返回新的连接对象（socket 即为连接对象）。

Socket.IO 是事件驱动的，其核心操作也就是广播一个事件和监听事件并相应，主要通过 socket.emit(eventName[, ...args][, ack]) 和 socket.on(eventName, callback) 两个函数来实现 emit 函数广播一个事件，第一个参数为事件的名字，之后可选为传递的参数，ack 为可选参数，服务器应答时会调用该函数。

on 函数用来监听一个事件，相当于系统中的一个 signal handler，这个函数为指定的事件绑定一个 handler，第一个参数为事件的名字，第二个参数为 handler。

3 详细设计

3.1 主体架构

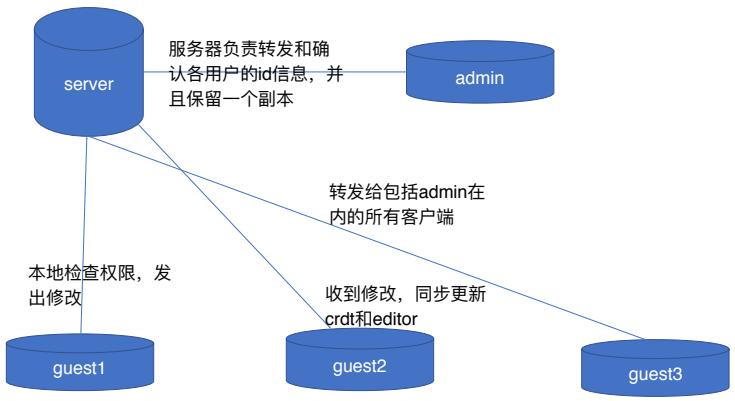

图 3: Client-server model

图 3 和图 4 分别说明了本项目的客户端-服务器模型的基本架构以及权限控制的具体机制。

我们采用了 Node.js 作为后端部署在服务器上运行，借助 npm 包管理器，服务器程序可以轻松部署在任何一台服务器上。package.json 中给出了相应的包依赖关系，使用如下两条命令即可完成服务器端的部署：

```
npm install
node app.js
```

默认监听 4004 端口，可以在 app.js 中更改。

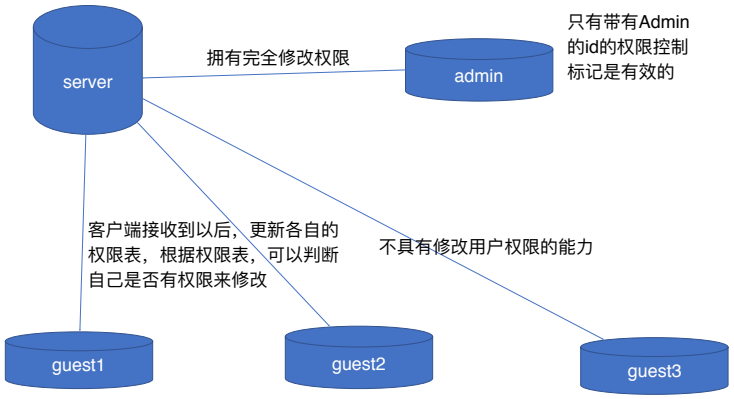

图 4: 权限控制设计

服务器在我们的设计中仅负责转发和确认用户的 ID 信息，并且保留一个副本给新加入的编辑者提供初始状态。本身基于 CRDT 的实时协作架构是可以用完全去中心化的点对点网络实现，但使用客户端-服务器的模型可以简化我们的工作，同时也便于实施权限控制。权限的检查我们采用在客户端本地进行。当客户端检查确认自己有权限编辑某一段时，才会发出修改请求。由于本项目的定位为提供一个用于技术团队在同一局域网下的文本实时协作，我们不考虑恶意攻击者的情况。恶意攻击者可以通过修改客户端跳过权限检查来实现作弊。但将权限的判断移植到服务器端也是可行的，而且也并不困难。

3.1.1 权限管理的具体机制

- 用户分为 admin 与 guest，admin 是唯一的，其余为 guest；
- 第一个登陆的人自动设为 admin，一旦 admin 退出（离线），admin 按顺序继承给下一个 guest
- admin 拥有完全编辑权限，并可划分区域，区域格式为\g{i}\ \g{i}\,

i 代表 guest{i}，即 guest{i} 只能在该区域中间（不包括区域标识符）编辑

- 每个 guest 在自己的编辑区域中再插入\g{i}\ \g{i}\企图划分出一个层次性的编辑区域是无效的，也无法在命令区域中插入一个\g{i}\表示编辑区域的开始，而 admin 或其他人 s 在该 guest 的编辑区之外插入另一个\g{i}\表示编辑区域的结尾而划分编辑区域。只有 admin 能设置权限；
- \g\ \g\表示公共编辑区域，任何人都可以在这一个区域中进行编辑。

3.2 基于状态的 CRDT 的设计

在本项目中，正如背景知识一节所介绍的一样，我们编写了一个基于状态的 CRDT 以实现各客户端副本之间的同步。在这一节里我们将介绍所使用的 CRDT 的具体设计。

3.2.1 数据结构

identifier 文档中每一个字符所对应的位置标识符（position identifier）就是由一系列 identifier 类型构成的，每一个 identifier 是由一个数字标识 digit（我们取 256 进制）和一个客户端 ID 的标识符 site 构成的二元组

```
C.identifier = (digit, site) => {
    return {
        digit : digit || 0,
        site : site || 0
    };
};
```

由这样一些列二元组组成的有序多元组就是一个字符的 position identifier。对于不同的 position identifier 我们可以定义一个大小关系

对于每个 identifier 的比较，先看第一个元素，也就是 digit，如果相等再比较第二个元素（客户端标识符）

```
C.compareIdentifier = (i1, i2) => {
    if (i1.digit < i2.digit) {
        return -1;
    } else if (i1.digit > i2.digit) {
        return 1;
    } else {
        if (i1.site < i2.site) {
            return -1;
        } else if (i1.site > i2.site) {
            return 1;
        } else {
            return 0;
        }
    }
};
```

对于整个 position identifier 的比较，采用类似字符串比较的方式，逐个 identifier 比较直到第一个不同的位置。如果公共部分相同，那么就看两个 position identifier 长度的不同。

```
C.comparePosition = (p1, p2) => {
    for (let i = 0; i < Math.min(p1.length, p2.length); i++) {
        let comp = C.compareIdentifier(p1[i], p2[i]);
        if (comp !== 0)
            return comp;
    }
    if (p1.length < p2.length) {
        return -1;
    } else if (p1.length > p2.length) {
        return 1;
    } else {
        return 0;
```

```
    }
};
```

char 在 CRDT 中，我们的字符类型就是对文档中的普通字符和 position identifier 放在一起做了个封装，每个 char 由三个元素组成，一个 position identifier（也就是 identifier 构成的数组），一个 Lamport 时钟（Lamport 时钟是一种逻辑时钟而非物理时钟，其主要用于在分布式系统中确定事件发生的顺序，其并不用于 CRDT 中的比较），以及字符的值。

```
C.char = (identifiers, lamport, value) => {
    return {
        position : identifiers || [],
        lamport : lamport || 0,
        value : value || ''
    };
};
```

3.2.2 API

我们设计的 CRDT 提供了以下 API 函数

```
C.add = (n1, n2)
C.subtract = (n1,n2)
C.fromIdentifiers = (identifiers)
C.toIdentifiers = (num, before, after, site)
C.increment = (num, delta)
C.compareIdentifier = (i1, i2)
C.equalChar = (c1, c2)
C.comparePosition = (p1, p2)
C.compareChar = (c1, c2)
C.generatePositionBetween = (p1, p2, site)
C.binarySearch = (U, V, comparator, notFoundBehavior)
C.getChar = (lineIndex, charIndex)
```

```
C.getCharValue = (lineIndex, charIndex)
C.getLine = (lineIndex)
C.compareCharWithLine = (char, line)
C.findPosition = (char)
C.getPreChar = (lineIndex, charIndex)
C.updateCrdtRemove = (change)
C.updateCrdtInsert = (lamport, site, change)
C.remoteInsert = (char)
C.remoteDelete = (char)
C.convertLocalToRemote = (lamport, site, change)
C.updateAndConvertRemoteToLocal = (change)
C.isNumber = (ch)
C.findAllAvailSpace = (site)
C.isAvail = (availSpaces, char)
```

参考文献

[1] showdownjs/showdown: A markdown to html converter written in javascript. https://github.com/showdownjs/showdown.

[2] Mehdi Ahmed-Nacer, Pascal Urso, Valter Balegas, and Nuno Preguiça. Merging OT and CRDT Algorithms. In *1st Workshop on Principles and Practice of Eventual Consistency (PaPEC)*, Amsterdam, Netherlands, April 2014.

[3] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Rec.*, 18(2):399–407, June 1989.

[4] Santosh Kumawat and Ajay Khunteta. Analysis of operational transformation algorithms. In Nitin Afzalpulkar, Vishnu Srivastava, Ghanshyam Singh, and Deepak Bhatnagar, editors, *Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing*, pages 9–20, New Delhi, 2016. Springer India.

[5] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, UIST '95, pages 111–120, New York, NY, USA, 1995. ACM.

[6] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM Conference on*

Computer Supported Cooperative Work, CSCW '96, pages 288–297, New York, NY, USA, 1996. ACM.

[8] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, Inria – Centre Paris-Rocquencourt ; INRIA, January 2011.

[9] Chengzheng Sun and Rok Sosic. Optional locking integrated with operational transformation in distributed real-time group editors. In *In Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, pages 43–52, 1999.