OSH 调研报告

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 项目背景

在这个时代中，信息，数据占据着最重要的地位。人们拼劲一切想要节约时间，获得最大的效率。正如 Kevin Kelly 在《必然》一书中提出的整个时代是处于一个流动的时代。以计算机产业为例，计算机产业发展经历了三个阶段：

1. 第一阶段：单机时代。这时候计算机处理的主要对象是从物理世界到信息世界的映射。这时候计算机中存在的概念与物理世界相对应，如“桌面”,“文件夹”,“文件”等。

2. 第二阶段：网络时代。这时候在网络系统中的主要对象不是单机中的“文件”或“文件夹”了，而是网络上的“页面”，及超文本，这些超文本是以无以计数的超链接组织起来的。

3. 第三阶段：流时代。页面和浏览器还存在，但概念远不如以前重要；PC 机也还存在，但在所有计算机中所占的比例越来越小。现在我们是从各种智能设备上接收（或发送）各种形式的信息流，可以是视频、音频或图片、文本等。

随着计算机和互联网的发展，世界上的大部分事物正在以新的形态，在新的媒介上存在，变得可以在互联网上流动起来。譬如书籍，在古代是竹简，流动性很差；后来是纸质书，流动性变强了，但一旦印刷出来，书就确定了以一本一本的实体形式存在了；再后来，书籍有了电子版，但最开始时，电子书的编辑者也只有一个人或者一个团队，是单方面的。而当前及以后会有越来越多的网络书籍是由网络大众一起编辑、一起阅读的，大家都是作者，也都是读者。最典型的例子，就是维基百科，是由网民一起构建和编辑出来的，他没有任何商业性，从来不做广告。另一个例子是音乐。最初的音乐，只能

在胶片或氧化物磁带中保存，一旦生成，就以实体形式存在为一个特定版本，不可以再改变。后来，音乐以电子形式存在于电脑中，可以被复制，流动。而现在，音乐存在于云端，每个人都可以编辑其中的一部分，或者其中的一个音轨，每个人都是创作者，每个人也都是收听者。文档也是这样的一种存在，在这个时代中，我们需要共同处理文本。这样文本实时写作系统就这样诞生了。

2 立项依据

- 目前市面上主流的文本实时协作应用都是对富文本的编辑，缺乏对 Markdown、LaTex 这类所见非所得的编辑方式的支持；
- 主流的协作软件同时也缺乏精确的权限控制，基本上都只能做到对整篇文档的粗糙的权限控制；
- 实时文本协作也经过了几十年的不断研究，直到最近一些较为完善的解决方案才被提出，Google Docs 这类的产品也只是于近些年诞生；
- 目前的轻量级的开源实现方案也较少，包括商业产品在内大多解决方案也都是采用了 OT 等较古老的技术，缺乏基于 CRDT 等较新的技术的实现。

3 重要性、前瞻性分析

目前大部分公司都在做文本实时协作，但使用过程中或多或少有一些不足，比如大部分的实时协作软件都不开源，而且就算是开源的实时协作插件，对客户端的定制化也极高，如果要移植到其他的编辑平台是很困难的。并且在使用苹果 Pages 软件进行实时协作编辑文档时，我们发现插入图片后会有图片位置不同步，并且经常位置出现随机变化，导致最后完成的文件还得重新排版，并且在多个人修改同一行时有时也会出现 bug 影响用户使用体验。同时目前并没有多少实时文本协作软件能支持像 Markdown 这样基于语法控制排版格式的可见非可得的编辑方式，大多采用了类 Word 这样的可

见即可得的富文本编辑模式。但目前对于以开发人员、科技记者、产品经理等人群中，Markdown 这种轻量级的文本编辑方式广受欢迎，我们也因此希望能实现一个支持 Markdown 编辑和实时预览的文本协作系统。此外，目前的文本协作系统几乎都只能提供对整片文档级别的权限控制，文档所有者只能控制某个用户对整篇文档是只读还是可读写。但是，这种权限控制方式在实际应用中就颇为受限。比如公司在开发一个产品编写产品需求文档时，可能几个负责不同领域的人员都只应该编辑自己所负责的领域，不希望能够不经意间对其他区域的文档造成影响。因此，我们也希望实现一个更加精细的权限控制系统，能精确到对于文本块级别的权限控制，例如像 C 语言一样每个大括号括起来的语句块只能由给定人员编辑，而其他人不得编辑。这种精细的权限控制尤其是在技术团队的合作中是很具有现实意义的。除了对 Markdown 的支持和精细的权限控制，我们同时希望实现一个轻量级、基于网页端的解决方案，可以轻松在任意一台服务器上部署，特别适用于一个小团队在同一个局域网下的实时协作解决方案。

4 相关工作和前言进展

4.1 编辑锁

编辑锁是最简单的实现方式，每个人在编辑文件的同时，对文件进行上锁, 其他所有用户都不能编辑此文件，这样的实现最大的好处就是逻辑简单, 而且完全避免了冲突。然而，这种方式的效率很低，类似于计算机网络中对同一链路复用的时候使用了纯 ALOHA 协议，在负载较高的情况下，冲突几率很高，排队时间时间长，应用在多人协作编辑时，可能会出现这样的极端情况：某人几乎长时间占用编辑权限，导致其他人一直处于冲突的状态而无法编辑。我们可以参考计算机网络中的 CSMA 协议, 可以大大提高复用效率，并且减少冲突。对于大流量端淹没小流量端的问题，我们可以借鉴滑动窗口协议，使每个用户都有可以忍受的服务质量。然而，由于延迟和时序等原因，如果要恢复由冲突导致失效的指令，单纯的编辑锁是没办法实现的。如果这些失效的指令丢失了，就会导致修改状态不稳定，或是呈现的编辑结果与修改结果不符，这也是用户很难接受的。

4.2 GNU diff-patch

相比于编辑锁，该算法解决了冲突带来的糟糕体验，并且有许多早期的协作编辑软件都是基于这种方法实现的。假设两台终端合作编辑一份文件，每台终端连接至服务器后，获取该文件的一个副本，然后在本地对副本进行编辑，每隔一段时间或者等用户停止编辑一段时间后，将编辑后的文件副本上传给服务器，服务器对新旧版本进行 diff 运算，并且智能合并。之后，将更改的 patch 广播给每个用户来进行同步。该算法有一个明显的缺陷：GNU diff 通过行匹配，在两个用户同时编辑同一行后，智能合并就失效了，出现无法避免的冲突。在负载较高的情况下，上行与下传会占用大量 I/O 资源，并且智能合并的结果也更加不可预测。

该方法的实时性不够强，但这也成为了一种优势，即对于延迟的要求低，而且通过适当的预测算法，只上传一部分进行 diff，缓解带宽压力。另外一个优势是，可以方便人工核对，然后进行选择合并。

4.3 Myer's diff-patch

在 GNU diff-patch 算法中，我们提到 diff 是行为单位进行检测并合并的，但仍然无法避免高几率发生的冲突，而且在智能合并的复杂度与随着负载的增加而极大增加，结果的不可预测性提高。Myer 对此方案进行了改进，将检测的粒度降低为字，使得冲突的几率和合并结果的随机性和波动性减小。冲突虽然不可能完全避免，但只会在对同一个字编辑的时候发生，这时可能会丢失字符。对于普通文本来说，这是可以忍受的情况。对于包含代码等对正确性要求极高的富文本文件，修复丢失的字符会占用大量的资源。

4.4 Operational Transformation

4.4.1 简介

Operational Transformation 则是一种相对较为先进的解决方案，能较好的解决同步性问题。这项技术相对来说也是较为复杂，已经在学术界和工业界被研究了二十多年了。著名的 Apache Wave 和 Google Docs 的核心技术便是 Operational Transformation。

Operational Transformation 最早是在 1989 年由 C. Ellis and S. Gibbs 提出的。但是过了几年，就有人发现了原始的方案拥有一些正确性的问题，也提出了不同的解决方案。随后十几年，无数研究者又对 OT 进行了不断的扩充和升级。

OT 这项技术，顾名思义就是当文档的状态发生变化时，就对用户发送的操作请求进行变化，从而保证每个用户文档状态的同步。在 OT 中，用户的所有编辑都会被转变成一个操作（operation）。当两个操作被应用到一篇文档时，我们会根据第一个操作对文档的改变，计算第二个操作该如何变化，以保证第二个操作的一正常进行，保留第一个操作原本对文档的操作意图。

4.4.2 核心思想

我们把这个协作系统中的每个参与者叫作一个站点（site），通常一个站点就对应一个用户的工作站。当然，也有一些机器会对应多个站点。我们在这里假设每个站点都可以和其他站点之间相互通信。

这里，我们来定义一下这个协作系统的结构 $G =< S, O >$ 这里 S 是站点的集合，O 是操作（operators）的集合。

每一个站点都由一个站点进程(site process),一个站点对象(site object),以及一个唯一的站点标识符（site identifier）构成。其中站点对象就是我们需要尽力维护站点对象的一致性，使得每个用户看到的结果尽可能一样。一个站点对象的结构是由应用程序决定的。操作的集合 O 也取决于站点对象，比如一个文本编辑系统的站点对象可能就包含一个字符串，并且定义操作集合 O=O1, O2。

O1 = insert[X; P] 在位置 P 插入字符 X

O2 = delete[P] 删除位置 P 的字符

例如之前的例子「ABCD」。

比如说现在一个用户想在 D 之前插入字符 X，那么他就需要发送操作 O1[X; 3]。

站点进程需要处理三种问题：

1. 操作的生成：将用户的编辑转换成操作，并且向整个网络中的其他站点广播

2. 操作的接受：从其他站点接受操作请求
3. 操作的执行：执行一个操作请求

回到 ABCD 这个例子中

1. Alice 发送了操作 O1[X; 3] 插入 X，Bob 发送了操作 O2[1] 删除 B
2. 服务器首先接到了 Bob 的请求，删除了 B，现在文本变为了 ACD
3. 服务器接受了 Alice 的操作 O1[X; 3]，但是现在正确插入的位置应该变为了 2，服务器会自动完成这一操作的变化
4. 现在 Alice、Bob 和服务器都认为文档的状态是 ACXD 了，这样就保证了一致性。

在这样一个协作系统中，我们需要一个服务器来保存文档的「真理」。这样即使一个客户端下线了，我们同样可以从服务器上获取文档。

一般来说，我们有如下两种操作：

- 包含变换（Inclusion Transformation）：用 IT(a, b) 表示，对操作 a 进行变换，以将操作 b 的影响包含在内
- 排除变换（Exclusion Transformation）：用 ET(a, b) 表示，对操作 a 进行变换，以将操作 b 的影响排除在外

现在让我们来看 IT 变换的一个例子：假设我们要对两个插入操作 Insert[p1, c1], Insert[p2, c2] 进行 IT 变换（插入的均是单个字母，p1,p2 为插入位置,c1, c2 为插入字符。那么我们的变换函数将是这样实现的：

```
IT_INS_INS (Insert[p2, c2], Insert[p1, c1]) {
    if ((p2 < p1) || (p1 == p2 && u2 > u1))^^Ireturn Insert[p2, c2];
    else return Insert[p2 + 1, c2];
}
```

以上代码用 user identifier 来标示用户操作的优先级。

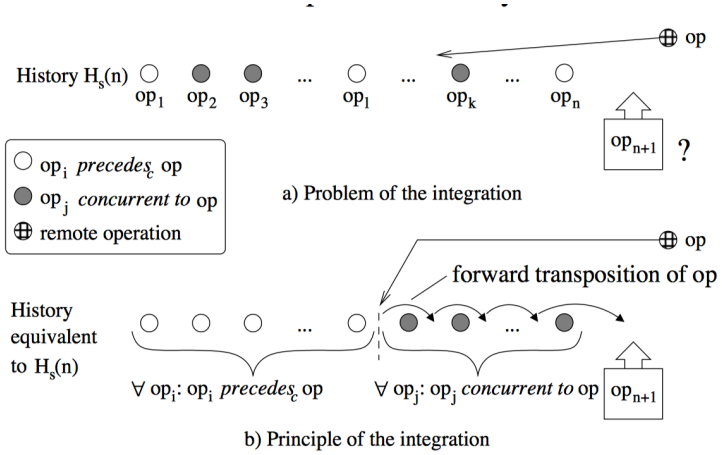

图 1: 操作变换

相比于以上 character-wise 的，实现 string-wise 的操作算法就会复杂多了，因为

- 字符串的删除会覆盖一个范围
- 并发的字符串删除可能会产生重叠（overlap）
- 之前插入操作可能会被之后的插入和删除操作改变

OT 中最大的一个难题就是实现撤销的操作了。在 OT 中我们要撤销操作 O，那么我们就要消除操作 O 的一切影响，同时保留其他所有操作的影响。在一个单用户文本编辑器中，这一点很容易实现，因为操作都是线性的。但在多用户协作系统中，这就成为了一个难题，操作变成非线性的了。

4.4.3 不同算法比较

dOPT 算法 没有中心服务器，且会遇到 dOPT-puzzle 的问题，这个算法并不正确。

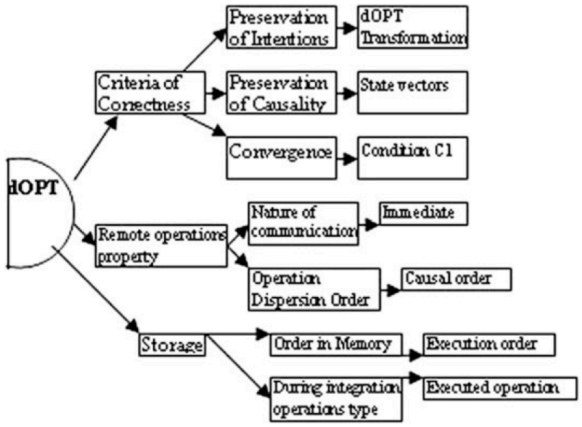

图 2: dOPT

adOPTed 算法 dOPT 算法的改进

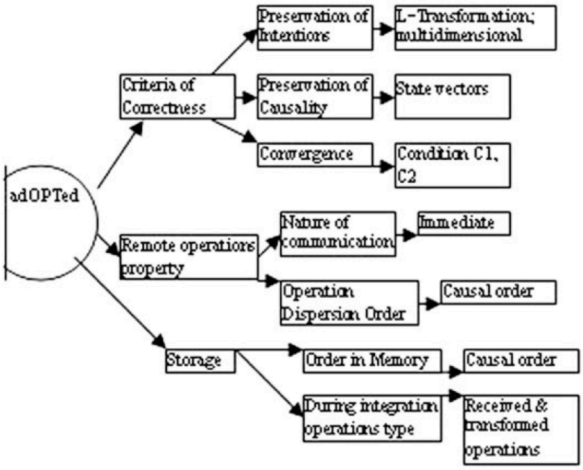

图 3: adOPTed

GOT 算法 GOT 算法很好的实现了一直行，并且支持 undo/do/redo。但是它只能支持两种基本字符串操作：插入/删除。

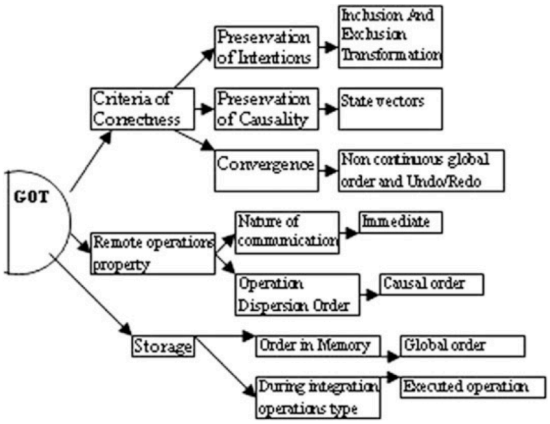

图 4: GOT

4.5 CRDT

4.5.1 简介

CRDT 是 Convergent and Commutative Replicated Data Type 的缩写，换言之“无冲突可复制型数据”。

人类目前实际可行的构建超大规模系统的比较好的解决方案就是利用分布式文件系统。但是分布式文件系统正确性很难保证 CAP 定理证明了在构建分布式系统的时候，Consistency（一致性），Availability（可用性），Partition tolerance（分区容错性），这三者只可以同时选择两样。分区容错性是实际运营的分布式系统所必需的。所以必须在一致性和可用性中选择其一。选择一致性，构建的就是强一致性系统，比如符合 ACID 特性的数据库系统。选择可用性，构建的就是最终一致性系统。前者的特点是数据落地即是一致的，但是可用性不能时时保证，这意思就是，有时系统在忙着保证一致性，无法对外界服务。后者的特点是时时刻刻都保证可用性，用户随时都可以访问，但是各个节点之间会存在不一致的时刻。最终一致性的系统不是

不保证一致性，而是不在保证可用性和分区容错性的同时保证一致性。最终还是要在最终一致性的各节点之间处理数据，使他们达到一致。

CRDT 是一个解决这种问题的很好的方案。其思想从总体上来说就是利用多存储的信息来在某一时刻解决一致性问题。在实际系统中应用，我们必须要考虑很多的数据类型和应用场景。CRDT 就是这样一些适应于不同场景的可以保持最终一致性的数据结构的统称。举例来说，假如我们想利用分布式系统来解决一个账户的支出收入问题。假设这个账户是 T，初始化有 100 块钱，用户可以通过系统里面好几个节点，例如 A,B,C, 访问它。那么我们最终的分布式系统，都会给 A,B，C 三者权限以对账户进行操作。这时我们就需要多加入一些信息来保证信息的正确性。此时我们这样设计这样的一个系统，每个系统存储的不是一个最终的数值，而是一系列包含了时刻与余额的记录，假设我们的系统从 t0 时刻开始的，那么在我们的例子里面，t1 时刻

1. A 系统存储的是 (t0,100), (t1,110)
2. B 系统存储的是 (t0,100), (t1,90)
3. C 系统存储的是 (t0,100), (t1,100)

这样的结构使得我们在传输了足够的信息之后，都能达成一致性。例如对于 C 系统，当收到足够多的信息，即是除自己之外所有的节点信息（A 和 B) 后，就得到了这样的结论

1. A 系统在 t0 至 t1 之间产生的变化是 +10
2. B 系统在 t0 至 t1 之间产生的变化是-10
3. C 系统在 t0 至 t1 之间产生的变化是 +0
4. A 和 B 系统与 C 在 t0 时数据一致，在 t1 之后未至 t2 之前一致的数据应为 100+10-10+0=100

类似的，在 A 和 B 上也可以这样判断

CRDT 不是一个基于共识的系统，其使用一些简单的数学性质来确保事件的连续性。CRDT 快速将备份整合为和使用序列处理效果相同的通用

的状态。由于 CRDT 不需要同步，一个升级会被快速地处理，并不被网络的延迟性所影响。

4.5.2 概念

原子和对象 一个进程可以存储原子和对象。一个原子是一个基本的不可被改变的数据类型，原子是由它的字面意思被识别；如果他们有相同的内容，那么原子可以被认为是相同的。原子的类型在这篇文章中被划分为整数，字符串，集合，元组，和他们的不可改变的操作。原子的类型可以被写成更低级的形式比如集合。

一个对象是可改变的，重复的数据类型。对象的种类是被大写化的。例如"Set"。一个对象拥有一个身份，可能是任意对象的一个内容，一个初始状态，并且一个由操作构成的界面。两个操作拥有相同的身份但是却定位在不同的进程中，这样的情况被称为另一个的复制品。

操作 该环境由未指定的客户端组成，这些客户端通过调用其接口中的操作来查询和修改对象状态，并针对他们选择的称为源副本的副本进行查询和修改。查询在本地执行，即完全在一个副本上执行。更新有两个阶段：第一，客户端在源处调用操作，该操作可能会执行一些初始处理。然后，更新将异步传输到所有副本；这是下游部分。

4.6 Google Docs

Google 办公室套件，是 Google 的一个在线文字处理（Google Docs）、电子表格（Google Sheets）和演示程序（Google Slides）。2006 年 10 月 10 日 Google 将 Writely 与旗下 Google Spreadsheets 集成为 Google 文档。它的第三个组件，演示功能（幻灯片），在引入 Tonic Systems 所开发的而技术之后于 2007 年 9 月 17 日发布。在 2012 年 10 月 31 日更加入了调查软件。

用户可以在 Google 文档中创建文档、电子制表和演示文件，也可以通过 web 界面或电子邮件导入到 Google 文档中。默认情况下这些文件保存在 Google 的服务器上，用户也可以将这些文件以多种格式（包括 OpenOffice、HTML、PDF、RTF、文本文件、Word）下载到本地计算机中。正在编辑的

文件会被自动保存以防止数据丢失，编辑更新的历史也会被记录在案。为方便组织管理，文件可以存档或加上自定义的标签。

Google 文档支持多用户协同工作。文档可以同时被多个用户共享，打开和编辑。在电子制表中，用户可以设置通过电子邮件提醒任何指定区域的更改。程序支持 ISO 标准的开放文档格式格式。支持流行的 Office 的文件格式，包括.doc，.docx，.xls(移动应用程序已不支持编辑存储) 和.xlsx。对 PDF 格式仅支持上传和共享。

4.7 石墨文档

石墨文档是中国第一款支持云端实时协作的企业办公服务软件（功能类比于 Google Docs、Quip），可以实现多人同时在同一文档及表格上进行编辑和实时讨论，同步响应速度达到毫秒级，是团队协作的最佳选择。在赋予产品高协作性功能的基础上，石墨文档还是一款具有中国式美感的科技产品，2015 年获得极客公园评选的最佳互联网创新产品 50 强。在 SaaS 类协作云文档国内市场占有规模排名第一，并且是中国各大 SAAS 产品中云文档服务的首选合作伙伴。2016 年 8 月，石墨文档启动 To B 战略上线企业版，提供诸如权限分级、数据保护等加强功能，目前注册企业超过两万家；同时与钉钉达成深度战略合作，成为钉钉首批推荐应用中的唯一的新品，也是唯一的云文档类应用。目前总用户数超过 200 万，合作伙伴十多家。

参考文献

[1] 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.

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

[3] Tim Jungnickel and Tobias Herb. Simultaneous editing of json objects via operational transformation. In *Proceedings of the 31st Annual ACM*

Symposium on Applied Computing, SAC '16, pages 812–815, New York, NY, USA, 2016. ACM.

[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] 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.

[7] 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.

[8] 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.

[9] Yi Xu, Chengzheng Sun, and Mo Li. Achieving convergence in operational transformation: Conditions, mechanisms and systems. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative*

Work & Social Computing, CSCW '14, pages 505–518, New York, NY, USA, 2014. ACM.