课程编号：C0801006001

分布式系统导论课程设计

基于 Web 的多人协同文档编辑系统的技术分析和概要设计

组名	写的	选题方向	多人互动的协同
组长(100%)姓名	XXXX	组长学号	xxx
副组长(95%)姓名	XXXX	副组长学号	xxx
副组长(95%)姓名	XXXX	副组长学号	xxx
副组长(95%)姓名	XXXX	副组长学号	xxx
组员(80%)姓名和学号(不多于3人)			
参与人员(75%)姓名和学号(不多于	无		
评定成绩		评定人	宋 杰
		评定日期	2019-1-15

东北大学软件学院

评分标准 **得分**

需求明确：清晰定义分布式系统的技术需求(25%)；

设计完整：涵盖大部分所学知识点(15%)；

设计正确：可以完成既定技术需求(30%)；

自主知识运用：是自己想的还是查阅资料获得的(20%)；

选题新颖：常规的教科书均可见的还是较为新颖的分布式系统(10%)；

大段文献抄袭且不给出引用的，对应考核点按零分处理

目 录

目 录 3
1. 系统简介 6
2. 体系结构 7
2.1 技术需求 7
(1) 操作请求集中处理 7
(2) 协同站点分布处理操作请求，减缓服务器压力 7
(3) 该系统需要能够跨平台使用 7
2.2 概要设计 7
(1) 采用非对称集中式结构 7
(2) 将计算分布，减缓服务其压力 8
(3) 混合式体系结构 8
3. 进程 10
3.1 技术需求 10
(1) 进程的同步和异步 10
(2) 进程的并发控制 10
(3) 客户端与服务器端架构 10
3.2 概要设计 10
(1) 进程的同步和异步 10
(2) 并发控制 11
(3) 客户端服务器端架构 11
4. 通信 11
4.1 技术需求 12

(1) 一致性（同步性） 12
(2) 互斥性 12
(3) 关于主服务器和从服务器 12
4.2 概要设计 12
(1) 解决一致性 12
(2) 保证互斥性 13
(3) 解决关于主从服务器通信问题 13
5. 命名 14
5.1 技术需求 14
(1) 唯一标识用户 14
(2) 多用户可同时在线编辑多个资源 15
5.2 概要设计 15
(1) 唯一标识用户概要设计 15
(2) 用户如何找到协同文档中的不同资源技术设计 15
6. 同步 17
6.1 技术需求 17
(1) 判断操作顺序 17
(2) 安排操作顺序 18
(3) 选举协作者 18
6.2 概要设计 18
(1) 使用 Vector clocks 处理判断先后的问题 18
(2) 协调者负责安排同步顺序 18
(3) 使用 ring 算法来选举协作者 19
7. 一致性与复制 20
7.1 技术需求 20
(1) 高可用性、高响应，编辑效果实时显示 20
(2) 解决操作冲突 20
(3) 副本的更新 20
7.2 概要设计 20
(1) 最终一致性与客户端一致性结合 21
(2) 使用 Operational Transformation 技术解决操作冲突 21
(3) 推式更新，以多播方式传播操作 21
8. 容错 22
8.1 技术需求 22
(1) 系统底层文件系统 22
(2) 主版本-副版本模式 22
(3) 防止 Byzantine 故障 22
(4) 确保对等系统中的可用性 23

(5) 需要可靠的发布-订阅通信 23
(6) 共享数据空间的容错性 23
8.2 概要设计 23
(1) 系统底层文件系统 23
(2) 主版本-副版本模式 24
(3) 解决拜占庭问题 25
(4) 对等系统可用性 26
(5) 可靠的发布-订阅通信 26
(6) 共享数据空间的容错性 27
9. 总结 27
参考文献 27
附录 (得分栏勿填) 28

1. 系统简介

该协同文档办公系统专注于团队沟通协作，支持云文件，在线文档，和收发消息等办公功能。

上传文件至服务器端存储，收发文件长期有效。如图 1-1

图 1-1 前端页面样式

在线文档，在线 word/Excel 等格式资源，可供多端多人随时编辑。如图 1-2

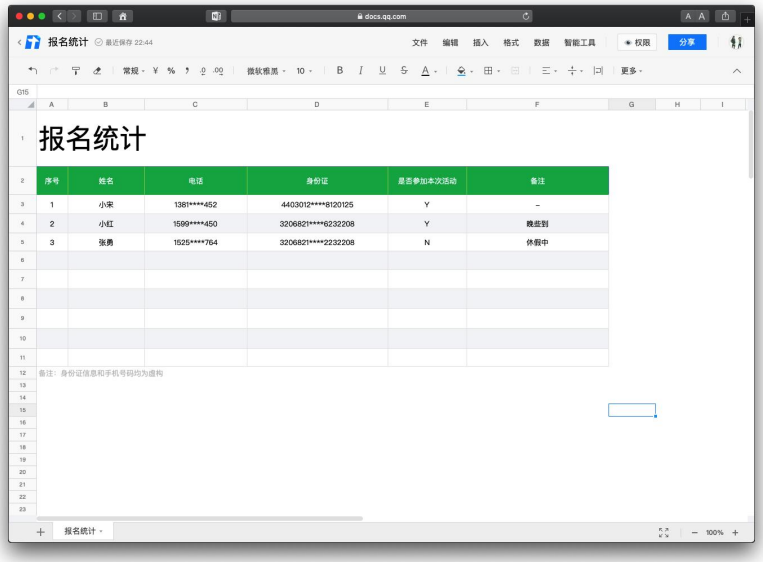

图 1-2 在线编辑表格资源实例

该协同办公系统具体功能项：

一. 在线编辑，实时保存：

1. 打开网页就能写，通过 QQ/微信等社交账号登录，实现在云端的实时保存
2. 可创建在线文档，同时支持导入本地文档
3. 能将在线文档保存至本地

二. 多端互通，实时同步

PC、Mac、移动端，任意设备皆可顺畅访问、创建和编辑文档，让您随时随地都能编辑文件；一处更新，信息多端实时同步

三. 多人协作，效率倍增

1. 分享文档，邀请其他用户一同编辑可设置访问权限针对共享成员，可以设置**「可查看」**或**「可编辑」**的权限，也可以随时进行**「移除」**
2. 编辑记录可追溯，文档旧版本可还原

2. 体系结构

基于协作的分布式系统，其本质在于各个组件之间的是分布式，开发这样的系统它的真正问题在于如何协调不同组件之间的活动。我们的系统要求进程间能够实时通信，并且实时同步，以达到互相协作的目的。

2.1 技术需求

基于协作的分布式系统，其重点在于分布式组件之间的活动协调。

(1) 操作请求集中处理

使用集中式的体系结构，系统协同工作的设计相对简单，服务器端程序只要简单的并发控制机制就可以保证用户并发操作的一致性；操作请求统一由一个程序副本进行处理，效率更高。而且添加退出站点更加容易，客户端只需要发送操作，并返回结果进行显示，计算能力要求较低[1]。

(2) 协同站点分布处理操作请求，减缓服务器压力

由于，若单使用集中式结构，有可能会由于宽带不足造成响应时间过长，或是服务器崩坏造成工作崩溃等问题。

系统应将操作请求分布，减缓该服务器的压力。由于将操作请求分布，响应时间也会大大提升。

(3) 该系统需要能够跨平台使用

我们的系统是基于 web 的协同编辑文档系统。系统应既简化计算要求，又需要实时性。

2.2 概要设计

针对以上需求，我们既需要简化计算要求，又需要能够实时通讯，我们组提出混合式体系结构。

集中式对计算能力要求相对较低，一致性操作也较为简单。而分布计算能够使服务器之间的压力减缓，而且提高效率。综上优点，我们提出将集中式和分布式相结合的去中心化体系结构，系统有一个专门集中处理请求的中心服务端以及多个分布协调的协同站点。各个协同站点在本地协同客户端的一致，中心服务端协同各个服务站点之间的一致。既有利于计算的简化又有利于压力的减缓。

(1) 采用非对称集中式结构

为集中式结构中的一种特殊结构。最典型的协同系统为 Netmeeting 程序。发起站点即为此次的服务器站点。

对于我们的在线协同编辑系统，发起文档的站点即为服务器站点，专门处理各个其他协同站点的操作请求。这种系统结构使得协同系统具有一定的动态特性，协同系统在一次协同工作初始化运行时才确定具体的服务器站点[2]，如图 2-1 所示。

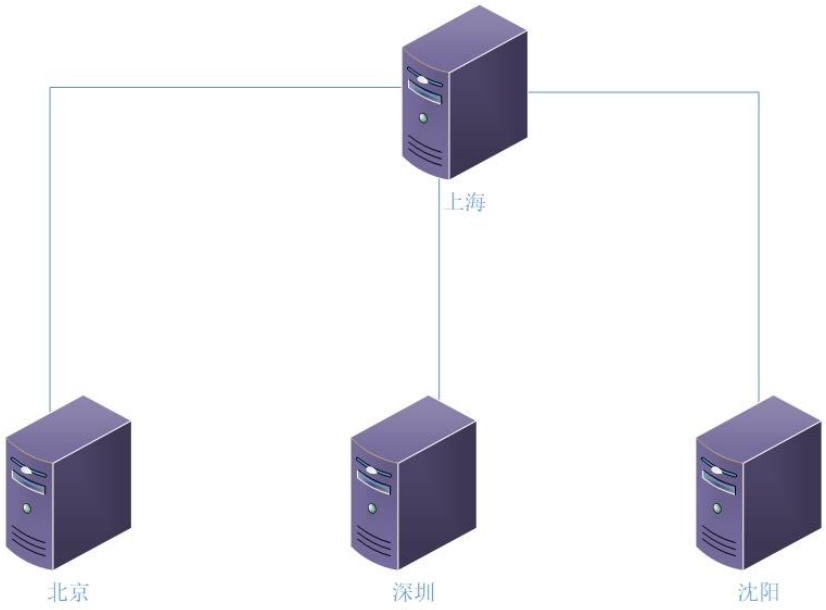

图 2-1. 集中式结构

(2) 将计算分布，减缓服务其压力

将集中式的服务端程序分布执行，能够减缓服务器之间的压力。

每个站点都需要协调其所拥有的客户之间的操作请求。即各个站点专门负责操作本地副本，保证本地副本的一致。将多组协调分布到各个站点之上去，由于本地的操作请求由本地站点进行处理直接返回给客户，所以操作时间会大大提升，其响应速度只与本地站点的计算能力有关。

(3) 混合式体系结构

综上，融合其优点，我们提出混合式的体系机构。

每个参与协同的站点协同其本地程序。而中心服务器协调其他的协同站点之间的同步。协同站点将操作请求转发到特定的主服务器上，服务器只需要完成对连接到本地站点的所有协同站点之间的操作请求处理，并在所有站点之间进行数据同步与并发控制。

使用此体系结构，能够将大规模的协同工作进一步细化为小规模的协同工作系统，而集中式的处理也避免了大规模的瓶颈问题[3]，如图 2-2 所示。

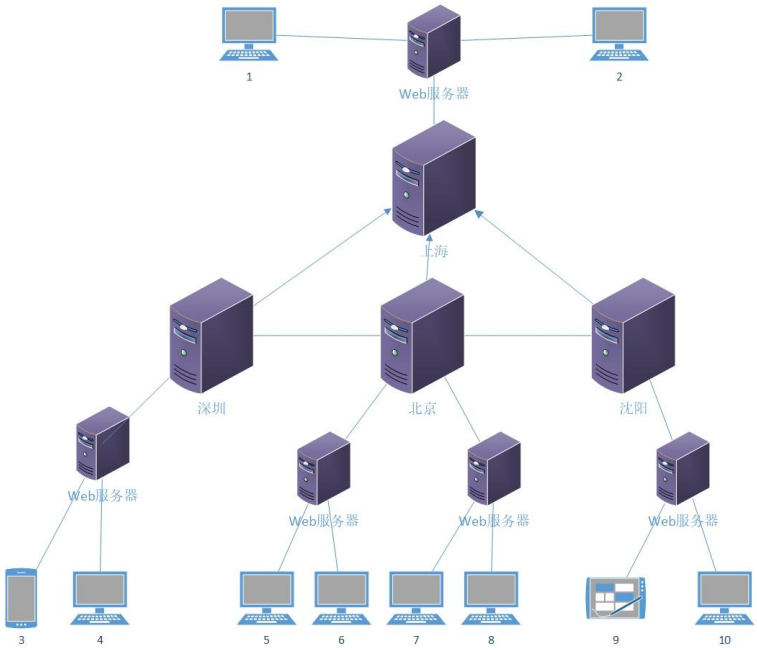

图 2-2. 混合式体系结构

如图 2-2 所示，上海为此次协同的主服务器。北京深圳沈阳站点的为参与的协同服务器，当 4，5，6 号客户发起操作请求，北京站点负责协同 4，5，6 号的操作请求，在本地将请求同步。然后将此请求转发到主服务器上海站。上海站将北京站点的操作请求同步到其他的协同站点上，保证各地的之间信息的一致性。

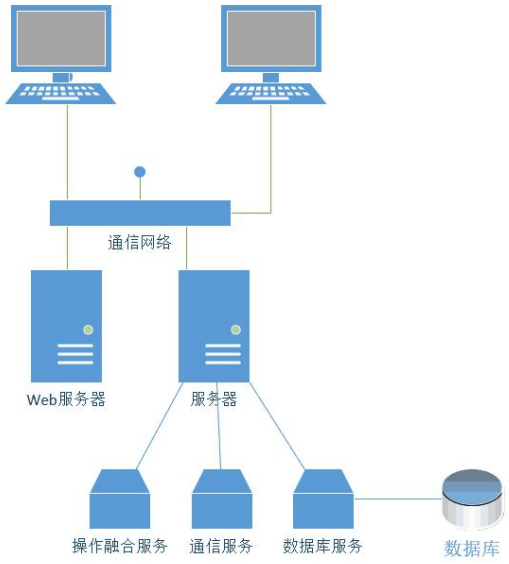

图 2-3. B/S 架构

而由于我们的系统要求跨平台服务，要求跨系统，因此我们采用基于 web 的技术，我们采取 B/S 架构如图 2-3。如此，能够简化本地的计算量。客户端只需要显示数据就可以。

3. 进程

进行协同办公时，有同步和异步两种工作模式。为了保证协作者同时对文档的操作不出错，就必须有合理的并发控制，最后为了保证这个平台正常高效地工作，而且要有足够的负载能力和灵活性，必须设计最佳的客户端服务器架构。

3.1 技术需求

(1) 进程的同步和异步

根据协同成员之间的协作方式，协同编辑系统可分为同步协作模式和异步协作模式。同步协作模式下，协同成员实时地对同一个共享文档进行协同编辑，本地成员对文档的修改操作实时地广播给远程的协同成员，后者能够及时地观看到修改后的效果；异步协作模式下，协同成员异时地对文档进行修改编辑。当某个成员完成编辑任务后，将文档传送给服务器或其他成员，其他成员在此前工作的基础上开始自己的文档编辑任务。

(2) 进程的并发控制

1.以往有的协同编辑大多采用的是以下两种：1.编辑锁，做法是，某一位协作者在编辑文档前，对文档某个区域设置一把“锁”，其他用户的进程不能再编辑这个区域。只有这个协作者的进程拥有写权限，其他协作者只有只读权限，这种方法并没有真正允许多用户实时协作。

2.串行化方法，每个协作者按照预先定义的顺序依次进行自己的操作，协作者使用消息序列进行传输和沟通，然而这种技术无法解决并行操作所带来的数据一致性问题。

在协同文档编辑平台中，协作者可能处在不同位置，多个协作者可能产生相互独立的对同一个共享文档的操作。所以为了确保协作者对共享文件的操作在服务器端和本地客户端的执行效果保持一致，就必须使用并发控制技术对来自不同用户的进程的操作进行协调处理，以便进行一致性模型的维护。

(3) 客户端与服务器端架构

设计时考虑到：单线程客户端:将请求转发到其中一台服务器，负载均衡等轮询;多线程客户端:可以建立不同副本的连接，允许并行传输数据

对于多线程可以提供一种方便的方法来允许阻塞系统调用，而不阻塞线程运行的整个进程。所以设计应该采取多线程服务器和客户端

3.2 概要设计

实现同步和异步两种工作模式，并发控制提供了两种解决方案：时间戳和分布式下的OT。平台使用混合式体系结构。

(1) 进程的同步和异步

文档协同编辑系统底层的数据通信包括同步和异步两种工作模式。当多个协同用户对协同文档进行实时协同编辑时，系统工作在同步模式下，此时系统的通信模块必须将协同用户对共享文档的修改结果实时同步地广播给其他协同成员。如果是异步工作，系统的通信模块的线程就被阻塞，等待文档修改完成，再广播到其他用户。

(2) 并发控制

第一种解决方案时间戳：对于协同文档编辑平台的并发信息流，使用时间戳（timestampt）来解决。将时间戳打包，和并发信息流一起发送，接受的时候按照时间戳的信息获得每个并发信息流的准确时间，然后按照时间顺序，把并发信息流排序到消息序列，这样就实现了执行顺序与操作顺序相同。

第二种解决方案是分布式下的 OT：把对文档的内容修改，等价成 3 种原子操作：

retain(n)　　保持 n 个字符

insert(str)　插入字符(串)str

delete(str)　删除字符（串）str

有多个协作者同时对文档进行编辑，将这些操作转换以后，再进行对共享文档的处理，由 OT 合并这些编辑，修改共享文档，并广播到协作者的客户端。

但是在分布式系统下，有多个线程/进程在同时处理请求时就会遇到覆盖问题，解决方法为：保证操作只在一个线程中执行。如果数据库支持事务，可以通过事务来解决；如果数据库不支持，就只能使用分布式锁了。两段新文本的修改都是基于同一个旧版本的，如果旧版本不一样，就有可能出错。解决这个问题，就必须引入版本，每次修改后都需要存储下新版本，有了版本我们就能使用 diff 来计算不同版本的差异，得到其他人修改的内容，然后通过 OT 合并算法合并两个操作[4]。

(3) 客户端服务器端架构

基于 Web 的多用户在线协同文档编辑平台，其采用 B / S 架构模式，主要由数据库与 Web 服务器端及具有 Web 浏览器的客户端两部分构成，其系统体系结构如图 2-3 所示，其中采用集中式存储协作者的信息和文档数据，采用分布式管理协同编辑的文档。

使用这个系统架构传递数据的时候，浏览器必须建立 TCP/IP 连接，向服务器请求最新版本文档，作为本地副本。客户端的一个线程负责读取输入数据并应用于本地副本，将数据传递给显示层。另一个线程负责与服务器通信，发送和接收用户的操作。

在使用多线程客户的时候，可以与不同服务器副本建立连接，这样就可以并行地进行数据传输了，并且确保整个文档完全显示出来所需的时间与使用无复制的服务器的情况相比要短得多。

服务器一般等待输入的文档操作请求，随后执行该请求，最后送回应答。服务器中，有一个称为分发器的线程，由它来读取请求。客户发送请求到服务器的某个已知端点。在对请求进行检查以后，服务器选择一个阻塞的工作线程，由它来处理该请求。

4. 通信

进程间通信是所有分布式系统的核心。作为一个可以多人互动的协同办公平台，我们需要保证不同的协作工作彼此互相独立，在同一个协同工作下的用户得到的是同一份文档，并且每当有用户对其进行修改，其他小组成员便可以看到已经修改过的最新文档；还应保证当有两个或两个以上不同的用户在同时编辑相同的部分时，该平台会自动检测并防止因同时修改但修改内容不一致所导致的不同用户看到的文档不同的问题；还应该尝试将文档存储在云端，方便用户随时进行编辑和保存而不损坏。

4.1 技术需求

(1) 一致性（同步性）

系统需保证在创建了一个多人协作工作后，每个参与其中的成员都将得到的是同一份文档。每进行一次编辑，该编辑记录将会被保存并且会更新所有人的文档，使其一直为当前最新修改之后的状态。

(2) 互斥性

系统避免了当两个或两个以上成员同时修改同一个位置所造成的文档显示不一致的问题。

(3) 关于主服务器和从服务器

当许多成员被安排在服务器下时，就需要在主服务器和用户连接的服务器之间进行通信连接，以确保即使位于不同服务器下的各个用户也能及时的显示出最新更新过的文档内容。

4.2 概要设计

本系统是一个基于 Socket 的网络通信系统，服务器与客户端的通信方式是基于 socket 的面向消息的通信。软件通信有七层结构（osi 模型）中由协议套协议最终组成最高级应用层协议，下三层结构偏向于数据处理，中间的传输层则是连接上三层与下三层之间的桥梁，每一层都做 不同的工作，上层协议依赖下层协议。而 Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就是一个门面模式，他把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部 ，让 Socket 去组织数据，以符合指定的协议。具体通信流程如图 4-1 所示。

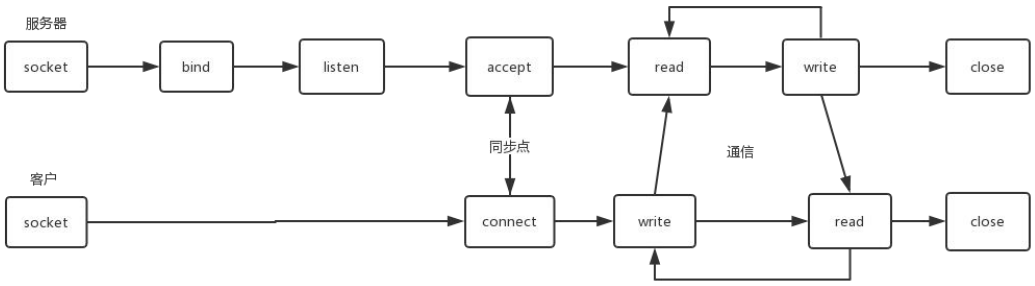

图 4-1.socket 通信流程

(1) 解决一致性

系统采取就近通信原则，将处于一个服务器周围的全部用户设置在同一个服务器下，建立服务器与客户端的 Socket 连接，将用户对文档进行的每一次修改都当成是客户端对服务器发送的请求信息，服务器接受该请求信息后便记录，每当有用户对文档进行修改，服务器就保存该修改并发送响应信息给其他用户，保证每个用户看到的文档处于最新状态。

为了对时间敏感的信息进行交换，本系统还提供对数据流的支持，每个用户对文档的修改就是用户与用户之间建立的连续数据流。对于连续数据流来说，同步非常关键，这样才能保证每次用户打开文档都能看到最新修改过的文档进行编辑。在同步传输模式下，数据流中的每一个单元都定义了一个端到端的最大延迟时间。数据单元的传输时间是否远远小于最大允许延迟并不重要。图 4-2 是一个数据流描述一个一般的客户-服务器体系结构[6]。

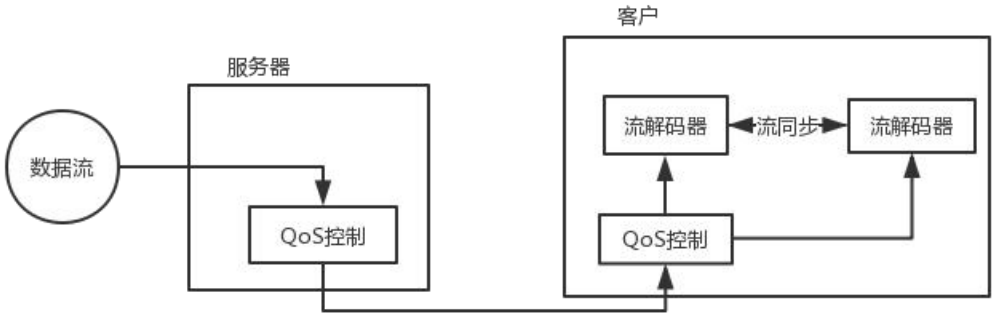

图 4-2.用于储存数据的流的客户-服务器体系结构

(2) 保证互斥性

互斥是指对于共享的数据，同一时刻只有一个进程可以操作它，在我们的系统中，必须保证任意时刻只有一个进程在操作服务器上的数据库，即记录文档发生什么编辑的数据库。由于数据库 I/O 速度远慢于通信和算法执行速度，可能出现以下情况：一次数据库操作还在进行中，又出现了一个或多个数据库操作请求。此时，为了避免出现写写冲突，可以设置一个专门用于存放数据库操作请求的消息队列，保证串行执行所有操作。

而临界区为对共享内存进行访问的程序片段。在避免竞争条件的一个好的解决方法还应该保证并发进程能够正确和高效地进行协作，因此要满足：任何两个进程不能同时处于其临界区；临界区外运行的进程不能阻碍其他进程；不得使进程无限期等待进入临界区；不应对 CPU 的速度和数量做任何假设。为解决进程互斥问题，提供了如下解决方案：1、屏蔽代码块：CPU 在发生时钟中断时才会切换线程，进程进入临界区后屏蔽掉所有中断，并在离开临界区时打开中断，这样就可以实现同一时刻只有一个进程读写共享数据；2、锁变量：共享（锁）变量为 0 表示没有进程进入临界区，否则有进程进入临界区；3、严格轮换法（自旋锁）：严格轮状法采用忙等待，即连续测试一个变量直达某个值出现为止；4、睡眠与唤醒：睡眠是将一个无法进入临界区的进程阻塞，而不是忙等待，该进程被挂起，直到另外一个进程将其唤醒；5、信号量：设置一个整型变量来累计唤醒次数；6、互斥量[7]。

(3) 解决关于主从服务器通信问题

采用远程过程调用（RPC）的通信方法来处理，具体表现为：当机器 A 上的进程调用机器 B 上的进程时，A 上的调用进程被挂起，而 B 上的被调用进程开始执行。调用方法可以通过使用参数将信息传送给被调用方，然后可以通过传回的结果得到信息。编程人员看不到任何消息传递过程。

当客户端调用远程过程时，客户将会阻塞，直到有应答返回为止。在没有结果要返回的情况下这种严格的请求-应答方式时不必要的，它只会导致客户过程向远程过程发出调用请求之后处于阻塞状态，从而无法进行本来能够进行的其他有用工作，为了支持上述情况，我们采用异步 RPC 的功能，客户可以在发出 RPC 请求之后立即向客户送回应答，之后再调用客户请求的过程。下图给出了常规 RPC 和异步 RPC 的对比[8]。

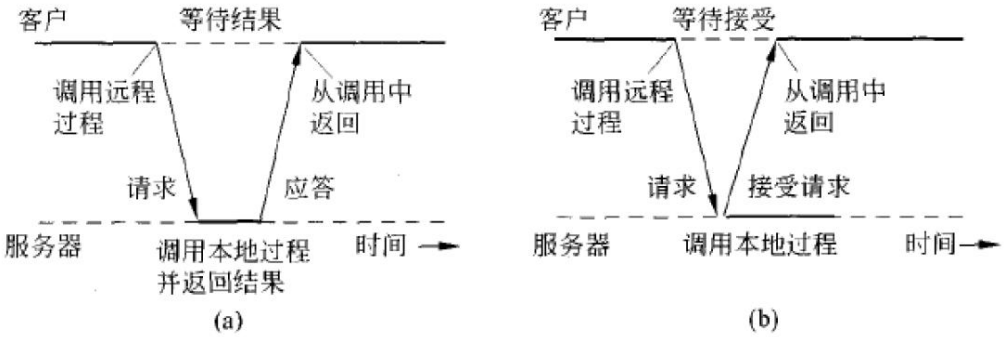

图 4-3(a). 传统 RPC 交互 (b).异步 RPC 交互

5. 命名

通过对编辑文档的各个用户命名而来标识一段文字是由哪位用户编辑

5.1 技术需求

(1) 唯一标识用户

通过唯一标识用户来给光标一个用户标签，如图 5-1 的 tiiko 用户

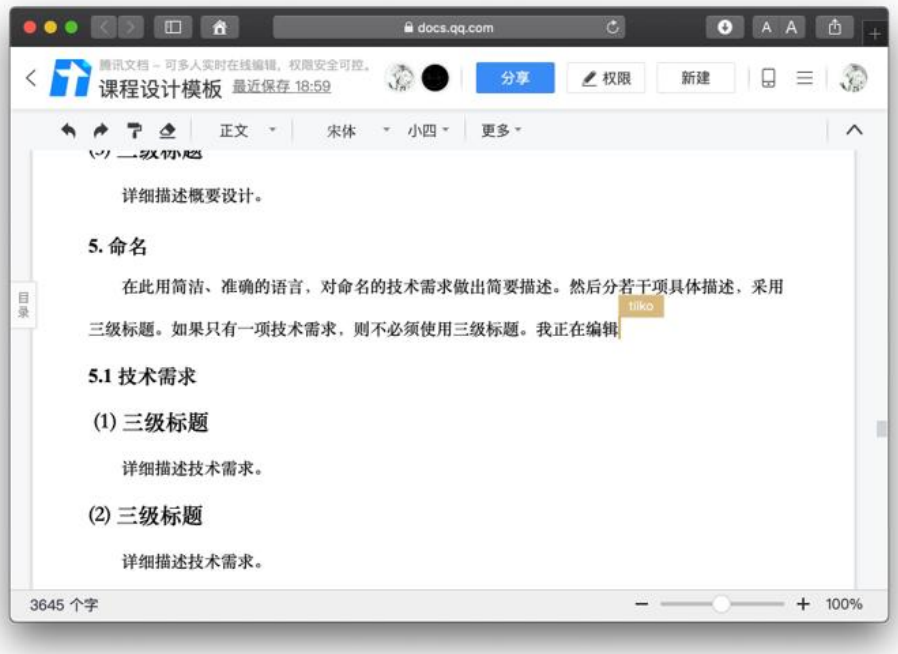

图 5-1.显示 tiiko 用户的例子

(2) 多用户可同时在线编辑多个资源

用户可以登录文档, 在加入的协同组中可以选择可编辑权限的多个资源进行在线编辑

5.2 概要设计

(1) 唯一标识用户概要设计

这里的 Entity 是各个用户开始编辑的 process, 因为在服务器上要放的资源仅仅是用户的一个标识, 直接使用 Home-Based 方式. 在多个用户开启协同文档编辑时, 用一个叫做 home 的 server 记录所有用户开始编辑的 process 信息. 同时 Client 将自己的用户信息发给 Home Server. 如图 5-2 所示

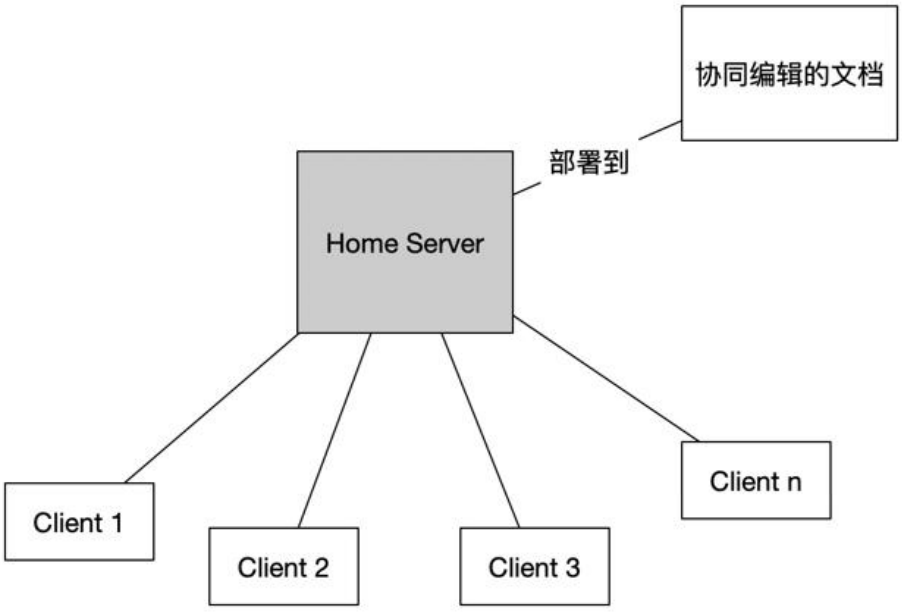

图 5-2 Home-Based 结构简略图

当 client 1 对文档进行编辑时, Home Server 将记录 client 1 的光标位置和 client 1 的用户名, 用户名从服务器获取. 然后将信息发给其他的 client 是他们能够知晓当前正在编辑的人,以及他在哪个位置编辑, 如图 5-3 所示.

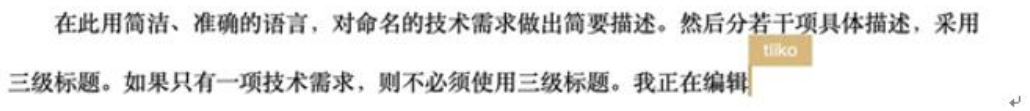

图 5-3. Tiik 编辑消息

(2) 用户如何找到协同文档中的不同资源技术设计

我们使用 distributed hash tables, 将可编辑资源用 KID 命名, 放置资源的服务器用 NID 命名, NID 等于其服务器的 IP 地址 hash 后得到, KID 由 key 值 hash 而来. 具体 hash 算法为 SHA-1, 如下图 5-4

Example hashes [edit]

These are examples of SHA-1 message digests in hexadecimal and in Base64 binary to ASCII text encoding.

```
SHA1("The quick brown fox jumps over the lazy dog")
gives hexadecimal: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
gives Base64 binary to ASCII text encoding: L9ThxnotKPzthJ7hu3bnORuT6xI=
```

图 5-4. SHA-1 的一个例子

服务器节点被放置在像环一样的结构中.

图 5-5 展示了一个要在 7 个服务器节点上放置 3 个资源的情况:

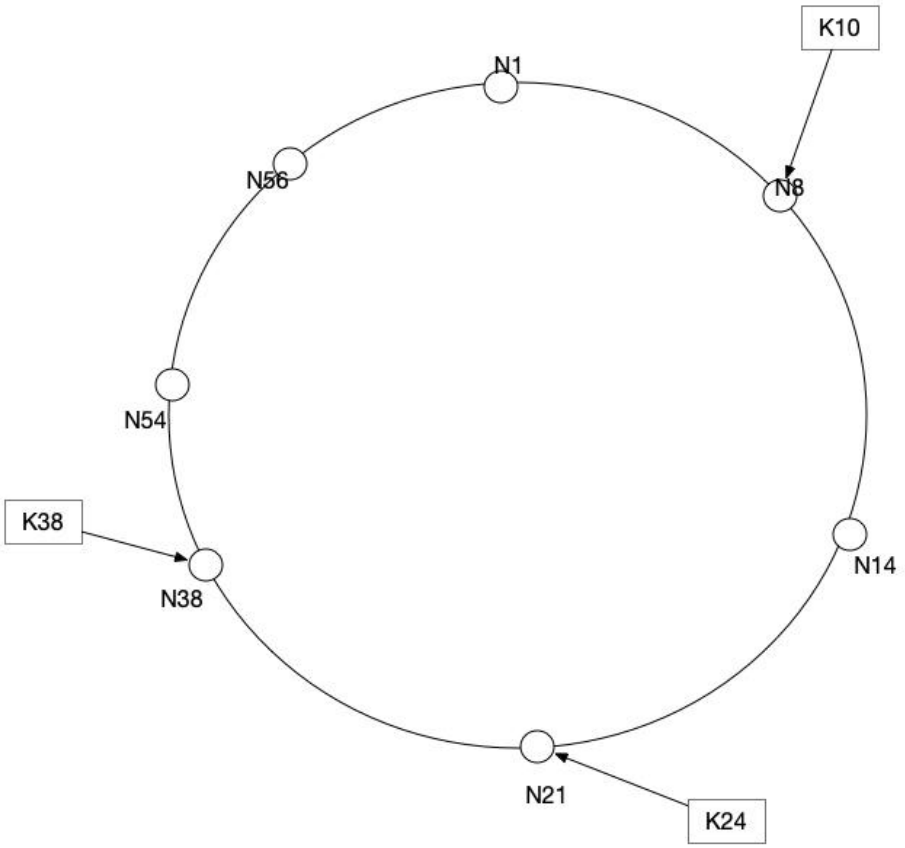

图 5-5 chord 环

对于服务器节点少, 资源多的情况可以直接用线性查找, 但在这里我们使用 finger table, 首先建立 finger table:

公式:successor($(n+^{2i-1})\bmod 2^m$) ($i \in [1,m]$)

对于上图:

$$i = 1, \text{successor}(9) = N14$$
$$i = 2, \text{successor}(10) = N14$$
$$i = 3, \text{successor}(12) = N14$$
$$i = 4, \text{successor}(16) = N21$$
$$i = 5, \text{successor}(24) = N32$$

$$i = 6, successor(40) = N42$$

每次有 NID 的节点想要使用 KID 查找密钥时，如果 NID < KID <= 后继者（NID），则后继者 (NID) 是目标；否则，它转发 NID 最接近且不大于 finger table 中的 KID 的节点，该节点是最接近的密钥的前身.

当用户想要编辑 KID = 54 的资源时，该算法查找如图 5-6 所示

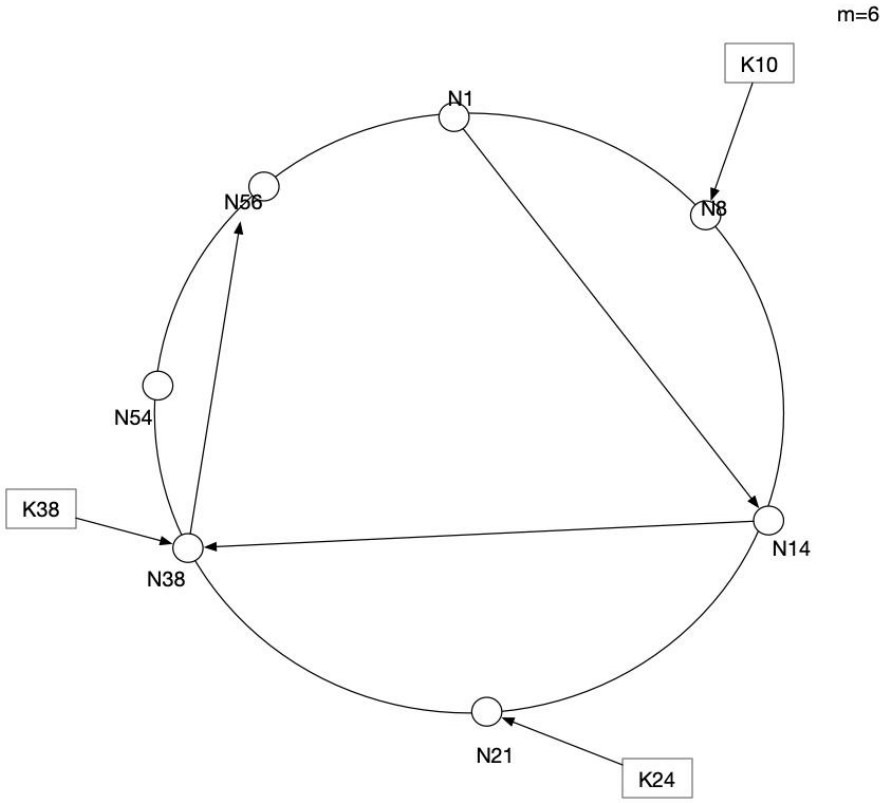

图 5-6 chord 环查找 kid = 54

相比于线性查找要快许多，能让用户更早的访问储存在系统内的在线可编辑资料

6. 同步

基于协作的办公系统中，如何使各端的信息实时同步是我们需要解决的问题。我们需要解决对同一文档，用户之间的操作顺序的安排，因此需要向量时钟的支持。而如何调度这些顺序，我们需要有特定的协调者来安排顺序。若协调者失联，我们需要进行选举，选出新的调度者来安排操作顺序。

6.1 技术需求

(1) 判断操作顺序

当服务端需要处理由各端发送而来的信息时，由于客户之间系统的时间不同，当信息一起到达时，服务端需要判断信息来源的逻辑先后。如何判断信息来源的逻辑先后是需要解决的问题。系统应当能够判断是哪位成员先对某区域进行的书写，并按顺序进行写操作。

(2) 安排操作顺序

当多用户对同一块区域进行书写的时候，系统需要能够决定操作顺序。服务端应当能够协调并发的信息。

(3) 选举协作者

信息的并发决定需要有协调者负责协调。这就需要我们在系统中选举一个协调者来进行安排。因此系统需要决定协作者，操作顺序由此协作者来决定；若协作者因为一些原因宕机，系统需要能够选举新的协作者进行协调。

6.2 概要设计

针对 6.1 提出的技术需求，以下提出对应的解决方案。对于实时同步的问题，

(1) 使用 Vector clocks 处理判断先后的问题

当多位成员同时在线编辑文档的时候，可能出现并行同时操作文档的问题，于是此处引入向量时钟来解决冲突问题。

每位用户都向自己的缓存写内容。当一定时间，向各成员发送自己的信息，其信息包括传递的消息以及自己当前的向量时钟。接收端根据成员发送而来的向量时钟，分析事件发生的因果关系，从而决定执行的顺序。

对于每一个结点，都维护自己的一个向量时钟。当某结点 i 执行写操作之前，将自己的向量时钟累加，即$VC_i[i] = VC_i[i] + 1$;而这个操作的时间戳便设为$VC_i[i]$。于是，结点 j 接收消息时，可以根据自己对应的时间戳来判断是否为最新消息。若对于向量 j，$VC_j[i] + 1 < VC_i[i]$,意味着仍然有消息未到达 j 结点，便延迟此操作，等待未到达的消息到达此结点，如图 6-1 所示。

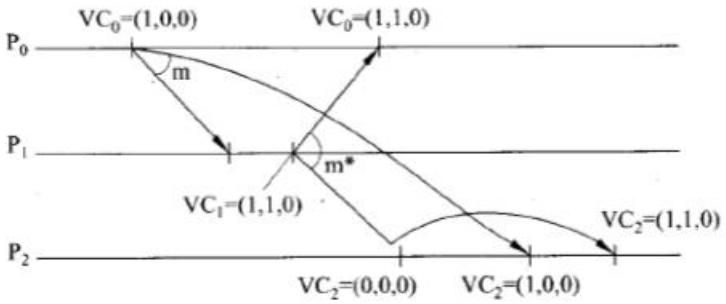

图 6-1. Vector clocks 示例

如此，当有多位成员同时在线同行操作文档时，通过判断多位成员的消息发送的向量时钟，能够判断成员编写的先后顺序。

(2) 协调者负责安排同步顺序

实现协同编辑最简单的方法就是为文件加锁，控制访问文件，当有人在访问时，避免其他人的同时编辑。然而，这种方式虽然避免了一定程度上的覆盖问题，但可能会影响实时的体验性。

于是，我们的实时文件编辑系统采用 OT 技术来协同各种操作。OT 技术通过将编辑转换成操作来解决冲突。点与点之间传递的是操作。

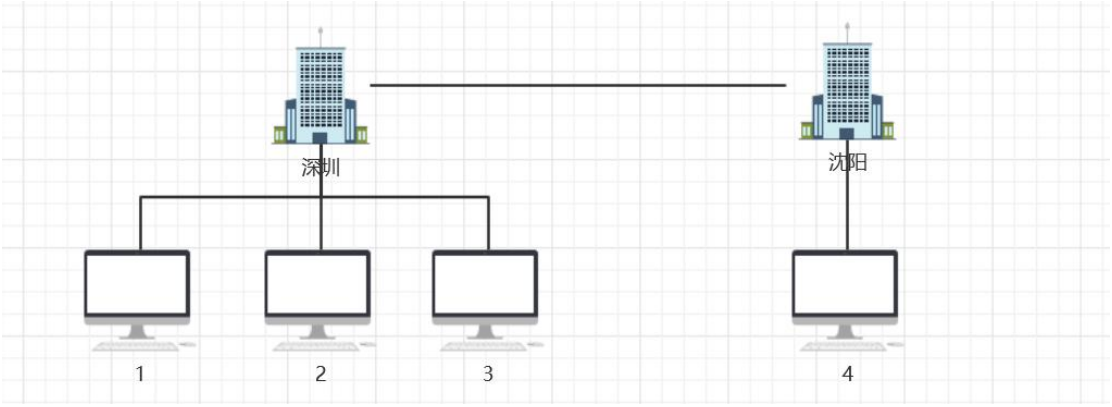

图 6-2. 协调同步顺序举例

如图，深圳，北京皆为协同站点，其中深圳为集中服务站，即为协调者，所有最终的操作请求都在深圳站处理。对于每一个协同站点，其所拥有的客户程序之间的同步协同由此站点来安排。如图的 1，2，3 号之间的顺序安排由深圳站来处理，采用 OT 技术将操作之间合并；4 号客户的顺序安排由北京站来处理，将负责的客户之间的请求合并。当协同站点之间的协同完毕时，将所有合并后的请求转发到深圳站，在深圳站完成对连接到此站点的所有协同站点的合并请求的处理，并在所有站点之间进行数据的同步与并发控制。

对于每个站点内，采用 OT 技术来保证顺序的一致。(详见 7.2.2 算法详解)

(3) 使用 ring 算法来选举协作者

结点之间的同步化需要协作者的参与，从而能够判断应当同步哪一位成员的信息。而如何选举结点，当结点失效时如何选举新结点，应通过选举算法来确定。

将所有参与的服务端结点按照物理顺序进行编号。当一个成员发现原先的协作者下线/宕机，系统使用 ring 算法来决定新的协作者，立刻发起选举。

当发现结点发起选举，将 ELECTION 消息发送给自己的后继者，若后继者崩溃，继续沿环发送，直到找到正在运行继承。每一步，发送者都将自己的进程号加到消息列表，将自己加入候选者队列。若下一个结点 ID 比自己的大，则选举下一个结点；否则，继续传递消息。最终，直到某结点收到选举自己结点的信息为止，其便为协调者。协调者此时需要广播自己成为协调者的信息，让其他参与成员直到自己的选举。

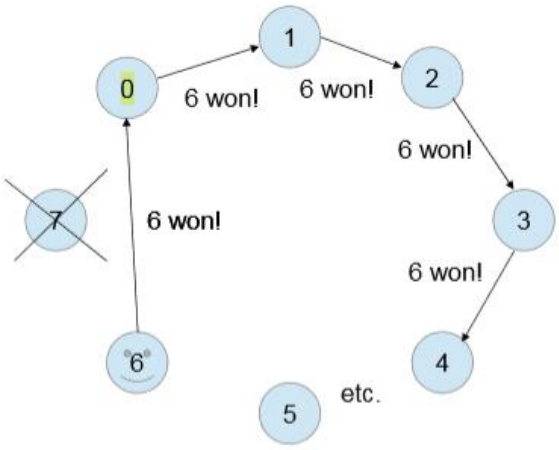

图 6-3. Ring 算法示例图

7. 一致性与复制

7.1 技术需求

在多人协同文档系统中，任意用户对文档的编辑必须实时同步到各个副本上，保证所有用户看到相同的文档。因此，从用户体验、核心算法和副本维护三个方面提出以下要求：

(1) 高可用性、高响应，编辑效果实时显示

用户对文档进行的任意操作应该及时反映在用户界面上，并能够看到其他用户对文档的实时修改，不应该出现文档被锁死、等待其他用户更新等情况。

(2) 解决操作冲突

多个用户并发操作时，应该解决操作之间的冲突，并保留用户的操作意图。以云表格为例，A 用户执行“在第 AB 列之间插入一列”，B 用户执行“在 B3 格写入数字 1”，两个操作合并后，表格的 C3 位置应该出现数字 1。

(3) 副本的更新

更新副本时必须满足以下需求：

1. 实时性高：副本必须及时更新，保证参与编辑的用户总在较新的文档上操作。

2. 网络代价小：为了保证多人协同编辑时网络顺畅，必须以一种对网络要求较低的方式传播更新

7.2 概要设计

本系统中，每个用户在本地保存一个副本，一致性问题主要由多个用户对同一文档的并发操作引起。用户打开文档时，首先向服务器请求最新副本，作为新的本地副本。当一个用户对文档进行修改时，首先转化为一个或多个操作（operation），该操作首先无条件应用于（apply）本地副本，随后被提交到服务器。服务器按照一定规则，将收到的多个操作合并（merge），在最大延迟时间内完成其他副本的更新，符合最终一致性模型。

(1) 最终一致性与客户端一致性结合

如果使用强一致性模型，任意时刻只有一个用户可以编辑，其他用户必须等待更新结束。显然这是极差的用户体验。因此本系统应该使用最终一致性模型与客户端一致性相结合的方式。

本系统中，每个用户应保存一个本地副本（只缓存关键数据），每次打开文档时，客户端从服务器处得到最新副本(可以通过计算版本差异,只传输操作不传输整个文档)作为本地副本。用户操作应该无条件应用于本地副本，因此文档总是可用的、本地副本总是高响应的。随后，操作被提交到服务器，服务器负责确认该操作并更新其他副本，最终，所有副本是一致的。这种做法带来的风险是，用户看到的结果可能是服务器未确认操作与已确认操作的混合结果。因此需要在容错机制中考虑这一点。

同时，系统应该满足读写一致性。读写一致性要求，一个进程 p 对数据 d 执行一次写操作生成版本 v 后，该进程对数据 d 执行的后续读操作，一定可以得到版本 v，也就是说，进程 p 对数据进行修改后，无论访问数据 d 的哪个副本，都能得到这次修改的结果。由于用户总是操作本地副本，读写一致性很容易满足。

(2) 使用 Operational Transformation 技术解决操作冲突

OT 类算法解决的问题是如何合并（merge）基于相同的状态产生的不同的操作序列。该类算法首先将用户行为转化为带有一定语义的操作（operation），再将多个操作合并，最后串行化执行，符合数据库要求。算法图解如图 7-1[9]：

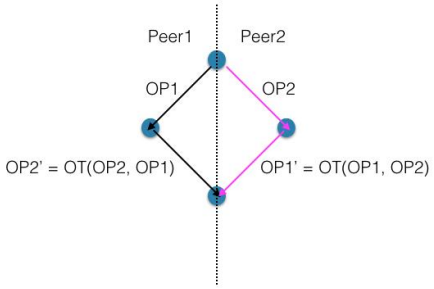

图 7-1 算法图解

如上图所示，从上往下看，基于相同的起点，左右有两个操作 OP1 和 OP2，分别是两个站点提交到服务器的两个不同操作。以左侧为例，执行完 OP1 后，副本状态发生了变化，而 OP2 是基于初始状态的操作，因此不能直接执行，必须进行操作变换，利用 OT 算法，左侧得到 OP2’，同右侧可得 OP1’。经过变换，左侧执行 OP1 和 OP2’，右侧执行 OP2 和 OP1’，两侧结果保持一致。

客户端收到服务器发来的 merge 后的操作序列，通过 ajax 修改 web 页面。

(3) 推式更新，以多播方式传播操作

基于推式的方法中，其他副本无需请求，服务器就会向它们推送更新。而本系统中，每个客户端的副本应该保持高度一致，采用推式更新，客户响应时间为 0，符合实时性要求。

本系统中，读写比例接近，服务器对各个站点提交的操作进行转化（OT），因此传播的更新应该是带有语义的操作，并以多播的形式通知副本。与传播更新通知、传播数

据相比，传播操作对网络要求通常最小，一定程度上使得用户体验更为流畅。与单播相比，多播不需要对不同客户端发送不同数据，因此维护较为简单，服务器负载较低。

8. 容错

我们的系统是一个在线协同办公的系统，面向全国乃至世界的上亿人在线协同办公，所以并发量，存储量都非常大，那么这样一个系统就有以下几个系统的特点：

1. 分布式文件系统

   由于系统支持在线编辑，所有的文档内容均存储在云端，需要大容量，一台机器虽然可以有许多磁盘阵列，但是就容错和运算效率来说，一台机器肯定是不如多台机器可靠和快速的，于是采用分布式文件系统。

2. 基于 Web 的分布式系统

   出于兼容性、成本以及方便程度，在 TIM 中内置功能进行在线编辑需要考虑到 Mac、windows 以及手机用户，如果都内置在线编辑功能，显然需要迭代多个系统上的软件，如果使用 web 就十分方便省事。

3. 基于协作的分布式系统

   由系统的名字可以得知，这是一个在线协同编辑的系统，自然就是基于协作的。

4. 基于对象的分布式系统

   后台在传播数据的时候都会将数据包装成为一个对象进行传递，使得前端 javascript 更方便解析，在分布式的数据存储系统中也是每一个对象存一行，很明显，这里封装数据，通过接口来操作这些数据， 我们将对象和接口严格隔离，这样的组织就是分布式对象，我们的系统也就是基于对象的分布式系统。

所以，系统关于容错的需求也就可以拆分成四部分，容错设计也就会针对这四部分进行设计。

8.1 技术需求

(1) 系统底层文件系统

由于并发量，存储量巨大，故采用分布式的文件系统，现在比较流行的是 apache 开源项目中的 Hadoop，部署较为简单。

首先购置或租用多台高性能稳定的服务器，每台机器安装上 CentOS，分发给它们 Hadoop，配置好配置文件，设置几台 NameNode，其余都设置为 DataNode，然后使得外网可以访问到这个 Hadoop 集群，HDFS 即准备完成。

(2) 主版本-副版本模式

我们的后台服务器打算使用 Java 的框架，于是可以使用 Java 虚拟机，由于 Java 虚拟机的不确定性，需要解决副本服务器可能崩溃的问题，那么就需要使用主版本-副版本模式来解决这个问题。

(3) 防止 Byzantine 故障

需要一种处理 Byzantine 故障的解决方案，通过构造有限状态机的集合来部署主动复制，并且这个集合中具有无故障的进程以相同的顺序执行操作。

(4) 确保对等系统中的可用性

使用冗余来作为高度可用性的关键解决方案。于是，需要设计冗余方案来解决实现高可用性。

(5) 需要可靠的发布-订阅通信

在基于协作的系统中，已发布的数据项只与活动的订阅者相匹配，可靠的通信至关重要。此时，需要一种可靠的多播系统来实现容错性。

(6) 共享数据空间的容错性

在共享数据空间中加入容错性，解决方案通常效率会十分低下。所以只有集中式实现才是切实可行的。此时就需要一种传统的解决方案，结合使用中央服务器与检查点技术，用来处理确保对等系统中的可用性。

8.2 概要设计

设计对应了之前容错概述的内容，分为了四大块，每一块有一到两个小块。四大块分别对应了分布式系统的主版本-副版本的设计，分布式文件系统的容错设计，基于 web 的分布式系统容错设计，基于协作的分布式系统容错设计。

(1) 系统底层文件系统

一个可靠的分布式文件系统是十分重要的，可以选择 Apache 顶级开源项目 Hadoop 来构建，这样是完全免费的，也可以选用商用软件一键构建，这里仅仅阐述设计。

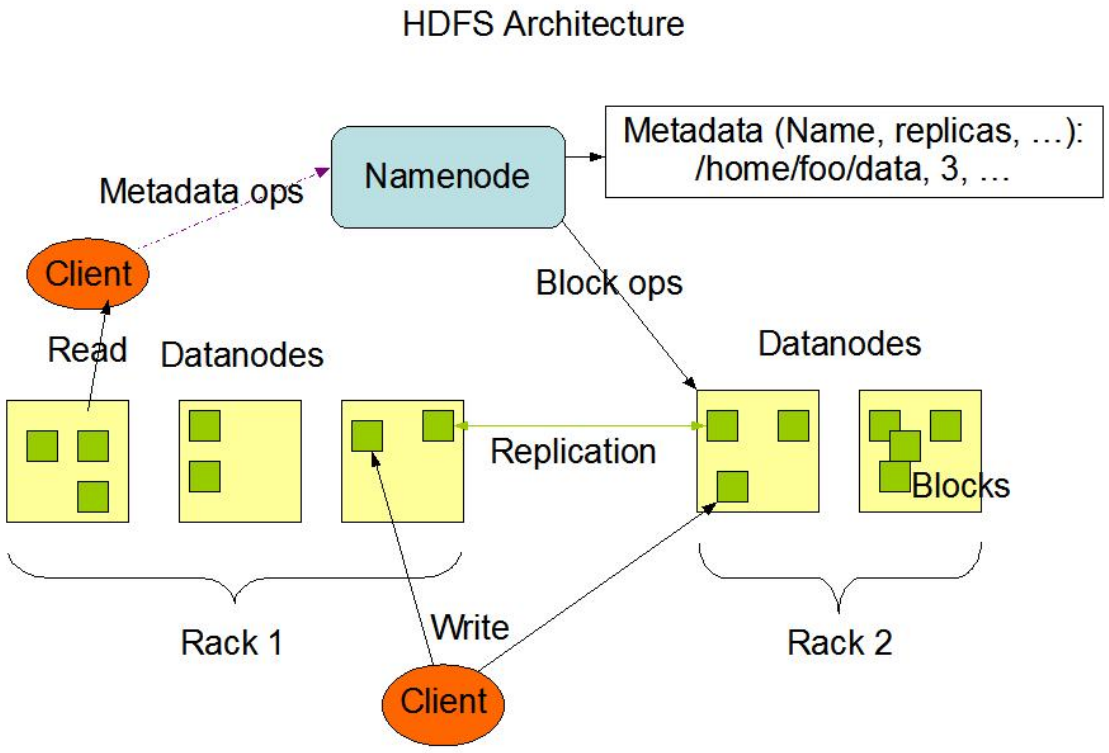

图 8-1 HDFS 文件系统

如图 8-1 所示，该图为 HDFS 文件系统，分布式文件系统都有至少一个 NameNode 以及多个 DataNode，顾名思义，DataNode 用来存储数据，NameNode 对它们进行管理，不管是读操作还是写操作，请求都需要先经过 NameNode，同时，在每个 DataNode 中都存在多个副本，不仅增快了访问速度也由此增加了容错性。集群中即便某些节点出现了问题，也可以通过其他的节点来恢复[10]。

(2) 主版本-副版本模式

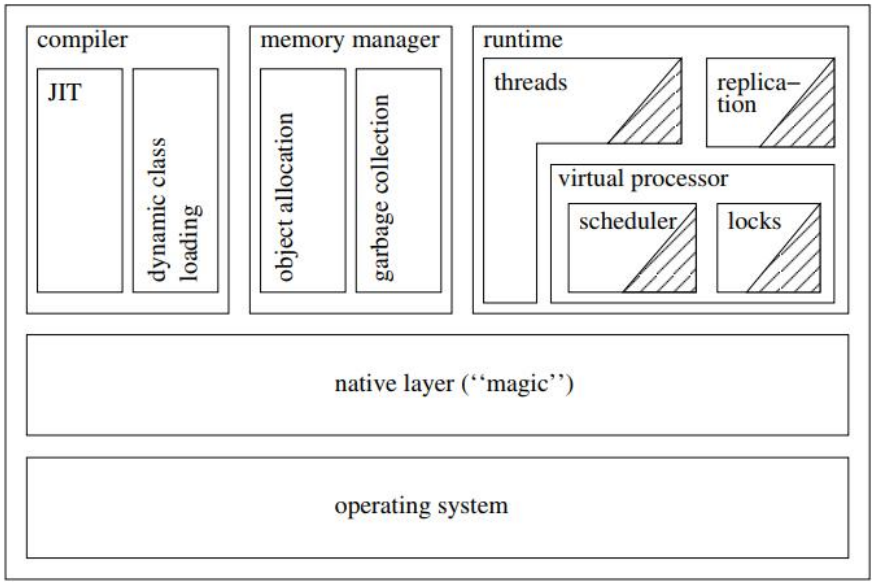

图 8-2. 主-副版本模式结构图

如图 8-2 所示，这里使用的是 Roy Friedman 和 Alon Kama 中论文提到的办法，首先，帧当中包含了几个上下文切换，可能因为线程结束 I/O 后阻塞然后执行帧指令，或者在预定义次数的上下文切换之后执行。只要线程发出一个 I/O 操作，线程就会被 Java 虚拟机阻塞然后停止运行。当一个帧开始执行时，主版本会让所有的 I/O 请求一个接一个地执行，并且把结果发送给其他副本，但同时，我们需要考虑两种情况，如果副本服务器崩溃，没什么太大影响，但是如果主版本崩溃时，就会丢失数据[11]。

于是为了降低损失，我们设计主版本仅在当前帧完成之后，它才会把更新信息发送到其他副本，如图 8-3 所示。

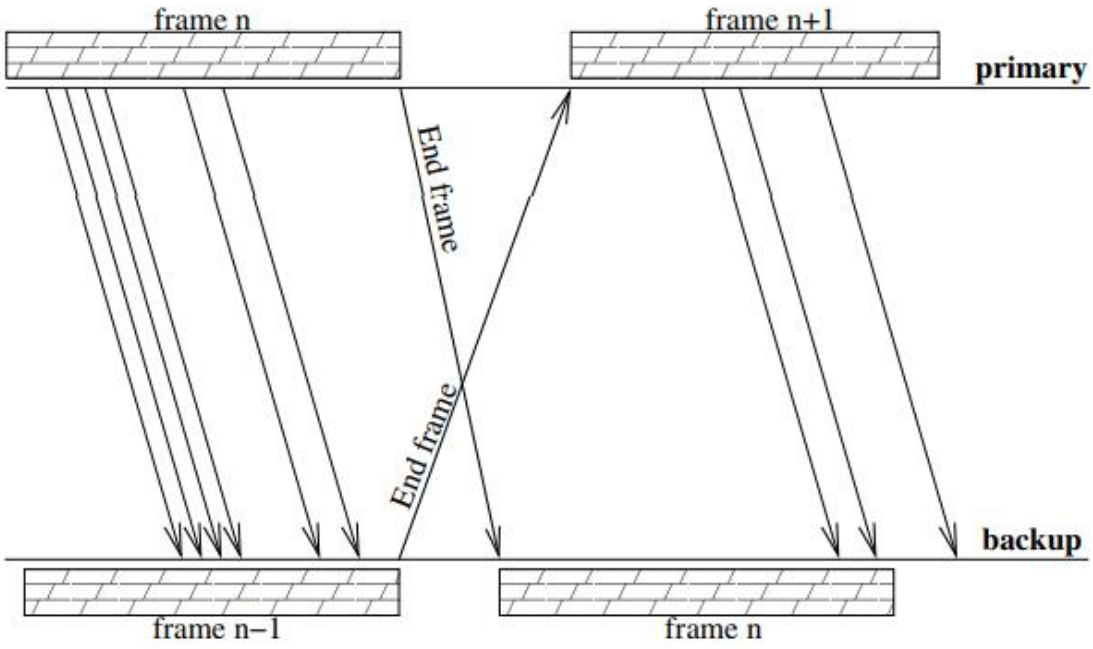

图 8-3. 改良的模式

(3) 解决拜占庭问题

服务器展示任意的故障就是 Byzantine 错误，书上第八章讨论了 Byzantine 错误，但是我们是假设通信延迟必须是有限制的，通常这样的假设并不现实，于是，可以借助 Castro 和 Liskov 的论文 Practical Byzantine Fault Tolerance 来设计，这篇论文的主题就是实践性的 Byzantine 错误，就是说这样的解决方案是可以直接应用于分布式文件系统，应用于生产的。其基本思想，就是通过构造有限状态机的集合来部署主动复制，并且这个集合中无故障的进程以相同的顺序执行，假设至多有 k 个进程同时失败，客户端发送一个操作给整个组，并接收至少 k+1 个进程返回的应答他，如图 8-4 所示。

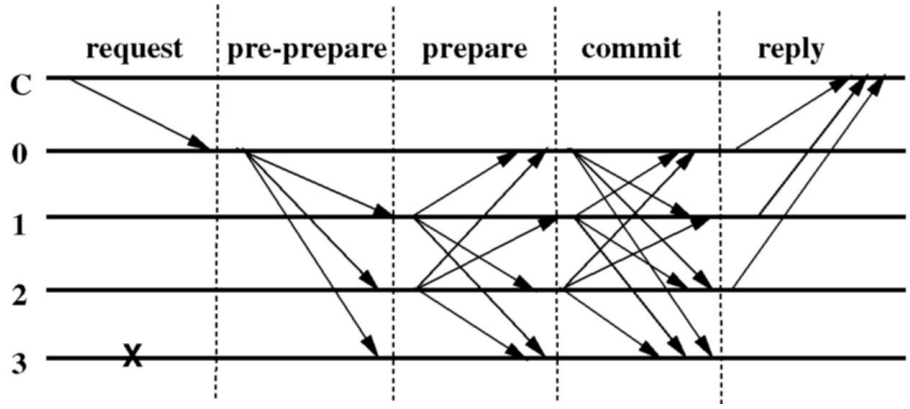

图 8-4 拜占庭问题

在第一个阶段，客户端给整个服务器组发送一个请求。一旦主机接收到请求，它就会在 pre-prepare 阶段多播一个序号，用来对关联操作进行正确的排序，如果副本接受提议的序号，它就会把接收的序号多播给其他的副本，在提交时会达成协议，所有的进程会相互通知并执行操作，最后客户端可以看到最终的结果。

我们基于 NFS 的文件系统实现了这样的协议，还实现了多种重要的优化和精心设计的数据结构，当然，这是 Castro 和 Liskov 在论文中提到的，这种包装器允许把 Byzantine 容错性与遗留的应用结合起来，如图 8-5 所示[12]。

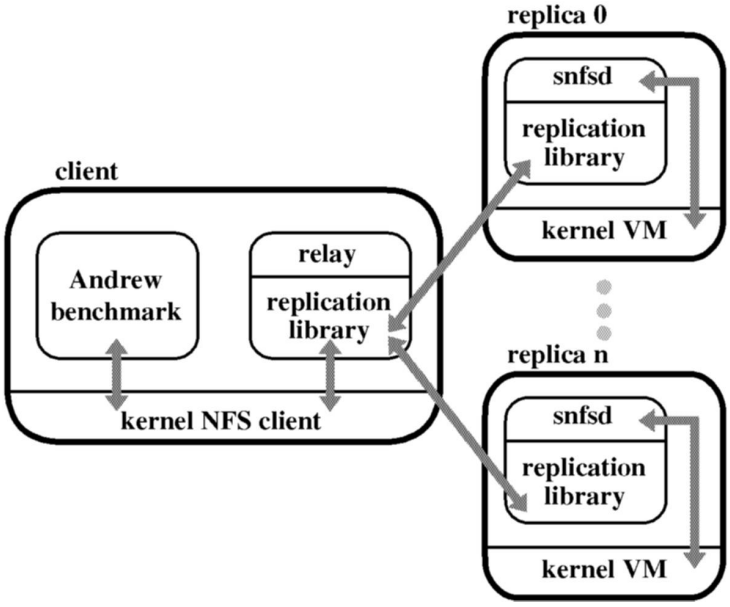

图 8-5. NFS 文件系统

(4) 对等系统可用性

前面提到需要冗余的办法来实现对等系统的高可用性，当涉及文件时，实质上有两种不同的办法来实现冗余，复制和擦除编码，但是我们采用擦除编码的办法。

擦除编码把一个文件分成 m 块，随后把它们记录到 n>m 块中。关键在于：任何 m 个编码块的集合都足以用于重新构造原始文件。然后选择一个恰当的 m，从理论上就可以实现高度可用性，如图 8-6 所示。

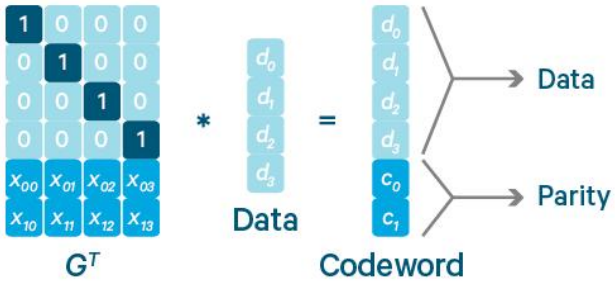

图 8-6. 擦除编码的方法

(5) 可靠的发布-订阅通信

实现实际的发布/订阅的可靠多播系统，我们需要以与基于内容的路由无关的方式建立可靠的多播信道，并且需要处理进程容错性。我们使用 TIB/Rendezvous 来解决这些问题。

任何时候一个 Rendezvous 守护进程向其他守护进程发布一条消息时，它都会把这条消息保存至少 60s。当发布一条消息时，守护进程把一个与主题无关的序号附加到这

条消息上。接收消息的守护进程通过查看序号可以检测是否有消息丢失，当丢失了一条消息，TIB/Rendezvous 会请求发布消息的守护进程重传该消息。

值得一提的是，TIB/Rendezvous 也提供了可靠的多播功能，它使用不可靠的 IP 多播作为底层的通信手段，遵循 PGM 的传输层多播协议。

(6) 共享数据空间的容错性

因为在共享数据空间中加入容错性会使得效率变得低下，只有集中式实现才是可行的，此时将使用传统的解决方案，结合使用中央服务器与检查点技术，中央服务器使用简单的主-备份协议备份的。

在我们采用的 GSpace 中采用了通过跨多个机器放置数据项的副本，更积极地部署复制。然后，每个几点都会计算自己的可用性，然后将其用于计算单个数据项的可用性。

为了计算节点的可用性，它会定期把一个时间戳写到持久化存储器中，允许它计算正常运行的时间以及停止工作的时间。

只需要考虑数据项的可用性以及所有节点的可用性，主节点就可以计算数据项的最佳分布，这样的分布将满足数据项的可用性需求。此外，数据项的分布随着时间也可能发生变化。

9. 总结

综上所述，我们完成了一个基于分布式的多人协同文档编辑系统。用户可以跨地区的和其他用户称为一个小组，对同一份文档进行同一时间的编辑；能在编辑的过程中看到哪部分是由哪位成员编辑的，当一位成员编辑完成后，其他小组成员会同步刷新文档，保证看到的文档是最新修改的，实现多人协同编辑功能。

分工方面，陈格平同学负责同步和体系结构两部分内容的撰写和最后整合；杨沛怡同学负责一致性与复制部分内容的撰写和最后整合；张钧然同学负责系统简介和命名两部分内容的书写和最后整合；蔡奇峰同学负责通信和总结两部分的书写，文档格式修改和最后整合；于岱洋同学负责容错部分的书写；王智同学负责进程部分的书写。

得分点方面，我们较为清晰的定义了每个部分的技术需求和其相应的概要设计，尽可能涵盖了课上内容，也有一些课外拓展的内容；除了参考教材，我们也积极去网上找相应的网址、中文论文和英文论文，结合自己的理解，完成了对相应部分的设计。

参考文献

1. 张志强.《面向异构网络的计算机支持协同编辑系统的体系结构研究》[D].硕士，浙江大学计算机科学与技术学院，2008 年
2. 张志强.《面向异构网络的计算机支持协同编辑系统的体系结构研究》[D].硕士，浙江大学计算机科学与技术学院，2008
3. 张志强.《面向异构网络的计算机支持协同编辑系统的体系结构研究》[D].硕士，浙江大学计算机科学

与技术学院，2008

4. 石映辉.《文档协同编辑协作机制及应用研究》[D]，华中师范大学，2012 年
5. 实时协同编辑的实现 https://blog.csdn.net/luyaran/article/details/54909887
6. Andrew S. Tanenbaum, Maarten Van Steen.《Distributed System Principles and Paradigms》(第二版）[M]: 清华大学出版社，2008 年
7. 进程间通信同步方法（互斥） https://blog.csdn.net/u012232736/article/details/79866452
8. Andrew S. Tanenbaum, Maarten Van Steen.《Distributed System Principles and Paradigms》(第二版）[M]: 清华大学出版社，2008 年
9. Operational Transformation 算法图解 https://blog.csdn.net/pheecian10/article/details/78496854
10. NetWork Working Group Request for comments https://www.rfc-editor.org/rfc/pdfrfc/rfc3208.txt.pdf
11. Roy Frideman, Alon Kama.《Transparent Fault-Tolerant Java Virtual Machine》[M]，2003 年
12. Performance Evaluation https://www.usenix.org/legacy/events/osdi99/full_papers/castro/castro_html/node6.html

附录 (得分栏勿填)

组名	系统名	组长			副组长 1			副组长 2			副组长 3			组员 1		
		姓名	学号	得分	姓名	学号	得分	姓名	学号	得分	姓名	学号	得分	姓名	学号	得分
写的没错	基于 web 的多人协同文档编辑系统	陈格平	2016 5187		杨沛怡	2016 5164		张钧然	2016 5243		蔡奇峰	2016 5123		于岱洋	2016 5269	

组员 2			组员 3			参与人员 1			参与人员 2			参与人员 3		
姓名	学号	得分	姓名	学号	得分	姓名	学号	得分	姓名	学号	得分	姓名	学号	得分
王智	201649 54													