一种云存储服务客户端增量同步算法①

吕 瀛[1,2], 刘 杰[1], 马志柔[1], 叶 丹[1]

[1](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)
[2](中国科学院研究生院, 北京 100049)

摘 要: 随着云计算技术的飞速发展, 越来越多的用户选择使用云存储服务来保存个人文件. 云存储共享与协作技术允许用户之间共享云端文件, 支持其他用户通过各种智能终端上的客户端对文件进行读写操作. 云存储共享与协作技术带来了文件历史版本大规模共享场景的需求, 这对云存储系统的并发 I/O 性能是极大的考验. 针对云存储服务共享场景的特点, 挖掘文件历史版本之间的关系, 采用基于增量传输的优化技术来提升云存储系统的传输性能. 在此基础上, 优化算法中强弱校验过程的内存占用和磁盘读写, 利用文件历史版本数据优化同步流程, 有效的减少数据传输量, 并且提高系统的存储性能, 适用于带宽有限和网络不稳定和大规模共享同步等极端场景.

关键词: 同步算法; 云存储服务; 共享同步; 多客户端; 并发

Increment Based Data Transmission Technique for Cloud Storage Service

LV Ying[1,2], LIU Jie[1], MA Zhi-Rou[1], YE Dan[1]

[1](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](Graduate University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: With the fast development of cloud computing technology, more and more users choose cloud storage to store personal files. Storage and share technique allows users share files and visit others' files with different kinds of client on the cloud. Storage and share technique brings the demand of large-scale share scene for the versioned files. This is a big challenge for the performance of simultaneously I/O. In this paper, according to the characters of the share scene in cloud storage, we try to dig the relationships between the versions of the file and take the increment based data transmission technique. By doing this, we optimized the performance of rolling checksum skill in increment algorithm and reduce the transmission quantity and improve system storage performance. In addition, this technique can help data transmission work in limited bandwidth and network instability scenario and large-scale share-synchronization scenario

Key words: synchronization algorithm; cloud storage service; share-synchronization; multi-client; simultaneously

随着云计算技术的发展, 越来越多的用户选择云存储服务来管理个人工作和生活的文件资源. 云存储是云计算范畴内的概念, 允许用户将文件保存在云端并随时随地的访问云端中央服务的文件资源. 云存储服务通常支持各种智能终端的访问请求, 例如 PC 客户端, 手机客户端, Web 客户端等. 云存储服务提供商在数据中心提供可靠安全的海量文件存储服务[1].

云存储服务中的文件共享功能, 作为云存储的核心功能之一, 已经被业界主流的各大云存储产品支持. 共享功能允许用户将个人存储在云端的文件资源共享给其他用户进行读写, 为存储服务加入了用户交互的属性, 不仅提高文件访问的灵活性, 而且提升了存储资源的利用率. 其特点有三: 1)文件多版本; 2)多终端; 3)高并发. ①多版本, 共享文件可能被其拥有者按时间顺序进行多次修改, 并具有多个历史版本. 共享文件的修改内容需要通过云存储服务同步到其接收共

① 基金项目:国家自然科学基金(61202065);国家高技术研究发展计划(863)(2012AA011204);国家科技支撑计划(2012BAH14B02)
收稿时间:2014-02-23;收到修改稿时间:2014-03-20

享的用户客户端. ②多终端, 一个共享文件可能被共享给多个用户, 而不同用户使用类型不同的客户端. ③高并发, 一次共享文件内容的修改, 将引起所有参与该文件共享的用户接收新版本的共享文件, 而云端海量共享文件的修改, 将会引起云端和各个在线客户端之间某一时刻同步数据的并发传输.

共享同步场景下的文件同步传输, 与个人文件上传下载同步传输有很大不同. 云存储个人文件同步的场景下, 一个文件与服务器之间的数据传输可以抽象为端对端的传输, 传输路径可以看成是线型. 因而一次文件修改只会引起一次服务器与客户端的同步. 而共享场景下不同, 一个文件可能被共享给多个用户的不同终端, 文件数据的传输路径是由中央服务器与各个客户端连接而成的星型的拓扑结构. 此时一次客户端文件的修改, 会带来所有参与共享的用户终端与云端服务器之间的文件数据同步操作. 而作为星形结构的中心, 云端服务器非常容易成为传输性能瓶颈, 文件并发传输会占大量网络带宽和服务器的计算资源. 由此可见, 数据在客户端与服务器之间的如何高效的传输是云存储服务的核心问题之一.

在云存储环境下, 传统的文件传输算法结合具体的传输环境有一定局限, 有进一步优化的空间. 通常情况下, 远程文件存储系统会通过增量传输算法来提高传输效率. 传统的 Rsync[2]算法是 Linux 下端对端的单向增量同步算法. 它被广泛的应用于各种远程文件同步场景, 并且能够一定程度上提高文件同步的效率. 但是其单向同步的算法在双工的传输环境下, 同步两端的运行环境并没有做具体区分. 如上一段提到的, 星形的拓扑结构的中心节点——云端服务器成为传输瓶颈, 在高并发的情况下, 服务器端的 I/O 压力会急剧增加. 端对端的 Rsync 传输算法在这种双工场景下, 并不能够完全适用[3,4]. 因此, 我们提出一种改进的增量同步算法, 通过优化服务器端内存使用以及分摊计算压力给各个客户端的方式, 尽可能减少服务器端的 CPU 计算和 I/O 的开销, 使双工的传输效率和云端服务器资源的利用率都更加得高效.

1 相关工作

文件数据的同步算法的目标是使同步两端的文件数据达到一致. 文件同步中的核心问题是如何确定两端文件之间的相同内容进而定位出不同内容.

增量传输, 从传输的文件类型来划分, 可以分为文本增量传输和字节级别增量传输. 文本增量传输基于历史版本的按行比对算法可以取得不错效果. 主流的版本控制软件 SVN, GIT, CVS 等采用的就是基于行比对的同步算法. 字节级别增量传输的场景, 按行比对的方式将无效. 增量传输算法需要借助相同数据监测技术, 对文件进行字节级别挖掘增量内容进行传输.

字节级别增量传输过程分为三个阶段: a.检测相同数据; b.传输增量数据; c.远端合并数据. 对于 c.合并数据, 一般可以采用线性的数据结构算法, 例如归并排序算法等. b.传输增量数据, 可以采用 TCP/UDP 或者是更高一级的 HTTP 协议来传输. 而相同数据检测则是整个过程中的核心阶段, 该阶段内将通过尽可能高效的相同数据检测算法定位增量数据.

目前业内主流的相同数据检测技术有四种[5,6]: 1)完全文件检测技术(whole file detection); 2)基于 FSP(fixed-size partition)算法的块检测技术; 3)基于 CDC 算法的检测技术; 4)滑动块检测技术. 传统的 Rsync 传输算法和本文改进的增量传输算法都属于 4)滑动块检测技术的范畴:

滑动块检测技术指的是, 通过用固定大小的窗口在文件流上滑动, 分别计算强弱校验码的方式进行数据检测. 如果文件块的弱校验码与预存的弱校验相同, 则进一步计算强校验码, 若强校验码再相同, 则认为该文件块是相同数据块, 滑动窗口越过当前块继续滑动扫描. 滑动块检测技术的特点是块大小固定, 弱校验计算快速, 有效节约计算时间. 对不同数据敏感, 能够检测出所有的不同数据. 缺点是如果文件微少修改, 会产生细粒度的文件碎片.

Rsync 算法是(4)滑动块检测技术的一种实现. 具体应用场景是两台计算机终端之间文件的差异同步, 例如, Linux 系统下同名 Rsync 数据镜像备份软件和远程数据容灾系统的同步备份[7,8]. 随着云存储技术的兴起, Rsync 算法的思想在工业界云存储领域中逐步得到应用. 例如 DropBox 公司[9]通过增量传输的思想对客户端与服务器之间文件进行差异化同步, 从而减少了网络上数据的传输量, 来提高同步效率.

2 云享系统的架构和数据模型

云享文件存储与共享系统是中国科学院软件研究所自主研发的云端文件存储与共享系统, 支持用户通

过多种客户端进行云端文件操作. 多个客户端之间通过与云端服务器数据同步保持本地文件数据与最新版本文件数据的一致性. 云享系统提供了覆盖个人和群组的共享场景的共享策略. 本文对基于增量的文件传输算法的研究正是基于该系统及其数据模型展开.

2.1 云享平台系统架构

云享系统的架构如下图所示, 一共分为 3 层:

图 1　云享系统架构图

图中最上层是应用层, 由各个客户端组成; 中间层是服务层, 包含了云享文件服务、用户管理、设备管理等核心业务功能; 最底层是存储层, 提供了可适配多个不同存储设备的存储接口, 支持本地存储和第三方公有云存储接入.

云享系统中的文件数据传输主要在相邻两层进行数据交互的时候发生. 即在上层和中间层的网络传输以及.中间层与底层之间的存储数据传输. 本文主要关注上层和中间层之间的网络传输中的增量传输算法.

2.2 云享平台文件存储模型

云享系统的存储模型如下图所示, 云享文件元数据根据文件在系统中的被操作的粒度分为三种.

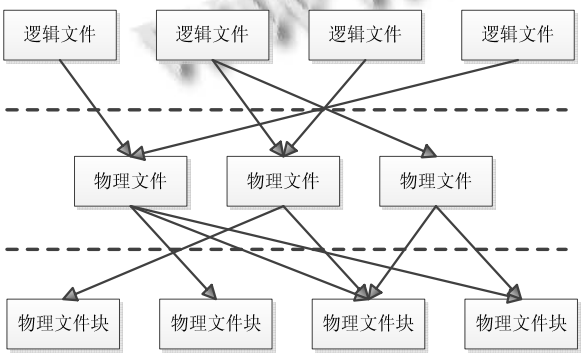

图 2　云享系统存储模型

(1)物理块元数据; (2)物理文件元数据; (3)逻辑文件元数据. 云享系统文件存储模型中, 每个逻辑文件关联一个物理文件. 一个物理文件切分为多个物理文件块存储到底层. 物理文件块是云享系统中最小的可操作的文件数据单元. 其中, 允许出现多个逻辑文件关联同一个物理文件的情况, 允许出现多个物理文件共享一个物理块的情况.

三层存储模型的好处在于, 1)让用户对用个人空间内逻辑文件的操作与实际上传下载物理文件的操作分开, 架构更清晰. 2)分块对于增量传输更加便利, 物理文件块的大小规格即可以作为滚动校验时分块的大小, 能够加速服务器端强弱校验码的计算速度.

3　增量传输优化的关键技术和设计实现

本章将主要结合云享系统介绍云存储服务增量传输模块的设计与实现. 首先介绍 Rsync 增量传输算法的工作原理, 然后介绍本文对 Rsync 传输算法在资源使用方面的优化, 最后介绍增量传输算法在云存储系统中算法流程的优化.

3.1 Rsync 算法工作原理

Rsync 增量传输算法是端对端的单向增量同步算法. 本节以云享系统的上传新版本文件过程为例. 起初, 云端服务器和客户端上都有一份文件 file_old 的相同副本. 随后用户修改了客户端 file_old 副本, 形成 file_new. 于是, 需要把客户端(Client)上的 file_new 上传到云端服务器(Service)上形成新版本. 则我们定义 Client 为发送方, Service 为接收方. 增量传输可以分为四步:

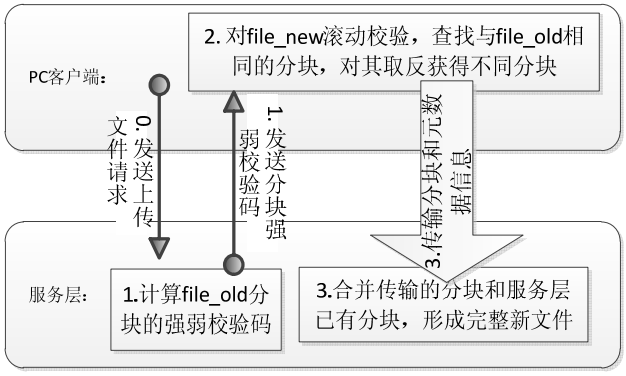

图 3　Rsync 算法执行流程

①Client 向 Service 发起 file_new 上传请求.

②Service 将 file_old 按固定大小 size 分成 block, 计算每个 block 的弱校验码(rollingHash)和强校验码(MD5), 并将各个块的强弱校验码集合 checksumSet 传输到发送端.

③Service 收到 checksumSet 后, 对 file_new 文件开启同样 size 大小的滚动窗口进行滚动校验. 滑动窗口的滚动校验以 Byte 为单次滚动距离向前滚动, 每滚动 1Byte 进行一次强弱校验. 校验方法校验遍历整个文件之后, 便可以定位出 file_new 和 file_old 所有相同数据块的位置, 剩余的就是需要增量传输到 Service 的 file_new 的数据块信息. 所以, 最后 Client 需要传输的数据是 a. file_new 中与 file_old 不相同的数据块、b. 新版本 file_new 的分块元数据信息.

④Service 收到 Client 增量传输的内容后, 将文件数据块与 file_new 的分块元数据进行关联, 形成新版本文件. 需要关联的文件块由两部分组成, 1.file_new 独有的文件块以及 2.file_new 与 file_old 公共的文件块. 因此需要做两步操作: ①将客户端传来的 file_new 增量文件块和 file_new 的分块元数据进行关联; ②将服务器端已有的 file_old 文件中定位出公共文件块, 并将其与 file_new 文件元数据进行关联. 至此, 在 Service 形成完整的 file_new 文件. 一次增量上传文件过程结束.

其中, ③中提到的滚动校验算法是传输过程中主要的时间开销, 算法具体如下:

首先介绍强弱校验码: 强校验码采用的是 MD5 校验码, 特点是碰撞率低, 但是计算比较耗时, 基本可以认为每个不同数据段有唯一的 MD5 值. 弱校验码(rollingHash)采用的是多项式滚动校验码, 特点是计算速度快, 针对滑动窗口, 可以通过将第一个 Byte 移出校验码, 加入最后一个 Byte 的校验码的方式, 更新计算检验码值, 但是有一定的碰撞率. 我们可以这么认为, 判断两个数据段是否相同, 通过计算弱校验码来猜测二者可能相同, 用强校验码确定二者肯定相同.

再看具体的滑动窗口滚动校验方法, 分两步:

①先计算滑动窗口内数据的 rollingHash 值, 并在 checksumSet 里查找其值是否存在.

②如果没有找到, 则滑动窗口向前滚动 1Byte.

如果找到了, 则进一步计算滑动窗口内数据的 MD5 值. 如果 MD5 值在 checksumSet 里存在, 则认为目前滑动窗口内数据为重复数据, 不存在则滑动窗口向前滚动 1Byte.

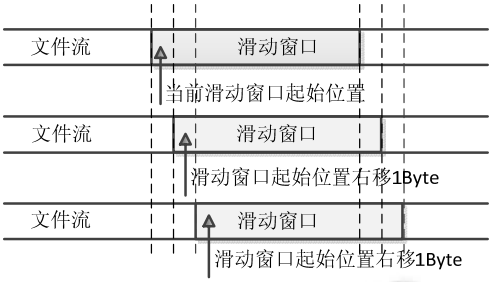

图 4 滑动窗口滑动方式

传统的 Rsync 算法应用了滚动校验强弱校验码的方式, 进行增量传输. 相比其他同步算法, 它在云存储系统中的优势在于能够有效识别出历史版本数据插入、删除和修改操作, 对文件进行增量传输, 能够有效的提升传输效率.

3.2 滑动窗口滚动校验计算优化

Rsync 传输算法在个人文件传输(端对端)过程中, 已经可以在一定程度上提升传输效率. 然而, 与端对端的应用场景不同, 云存储环境下具有高并发的特性. 传统 Rsync 算法在计算校验码的过程中, 因为参与一次传输的终端数量只有两台且文件量小, 在实现算法的过程中, 无需过多考虑内存占用的问题, 只要保证滑动窗口内的数据全部存储在内存中即可. 而面对云存储环境下的增量传输, 服务器端作为星形传输拓扑结构中的中心点, 需要应对高并发的场景, 系统的并发度受内存大小、网络带宽、磁盘 I/O 读写速度的限制. 因此, 如何优化算法计算过程中内存占用大小和磁盘 I/O 次数, 是提升系统并发度的有效方式. 本论文对 Rsync 中滚动校验码的性能进行了优化.

上文中提到 Rsync 的滚动校验码, 运用增量的思想, 每次只需要计算两次窗口之间不同的 Byte 值. 因此磁盘 I/O 只需要关注滑动窗口一头一尾这两个 Byte 的数据即可.

图 5 滑动窗口头尾 Byte 图示

因此, 计算弱校验码时每次从磁盘读取滑动窗口之后的下一个 Byte 进内存, 与滑动窗口第一个 Byte 数据替换, 实现增量更新滑动窗口的强弱校验码信息. 这种方式可以解决每次计算弱校验时内存占用依赖于

滑动窗口大小的问题. 此时, 空间复杂度是 O(1). 但是 Byte 级别的滑动, 读磁盘的操作非常频繁和耗时, 本节针对磁盘 I/O 操作做了性能提升.

本论文采用缓冲区的思想来预读入若干个Byte的方式来优化算法. 具体的缓冲区数据更新算法如下:

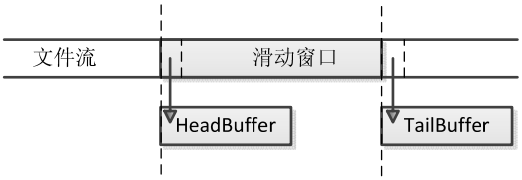

图 6　滑动窗口缓冲区图示

开辟两个缓冲区, headBuffer 和 tailBuffer, 大小都为 bufferSize, 其中 size 远小于滚动窗口大小.

滑动窗口移出 HeadBuffer[i]和包含 tailBuffer[i]是一次窗口滑动.

滑动窗口读取(bufferSize-1)次之后, 将第 bufferSize 个缓冲区数据移至缓冲区第 1 个位置, 随后读入(bufferSize -1)个 byte, 完成缓冲区的向前滑动操作.

```
Algorithm: Rolling checksum memory optimal Algorithm
Input:
hStart  // 滚动窗口的头坐标
tStart  // 滚动窗口的尾坐标
headOffset  // 头坐标的Buffer位移
tailOffset  // 尾坐标的Buffer位移
Output:  next head byte for rolling window
Method:
IF  headOffset == bufferLen - 1
   // 到了buffer最后一个byte，则将其放到数组0的位置
   headBuffer[0] = headBuffer[headOffset];
   curTailOffset = tStart + tailOffset;
   IF curTailOffset + bufferLen > fileLastByteIndex
      // 如果新bufferLen小于当前bufferSize，适配bufferLen
      byte tmpByte = headBuffer[0];
      bufferLen = fileLastByteIndex - curTailOffset + 1;
      // 因为buffer长度要算上前一次buffer中的HeadBuffer[BufferLen-1]，
      // 即前一次的最后一个byte，所以最后+1
      headBuffer = new byte[bufferLen];
      headBuffer[0] = tmpByte;
   ENDIF
   read (bufferLen-1) Byte to Buffer;
   hStart = hStart + headOffset; // 更新hStart
   headOffset = 0;
ENDIF
Return headBuffer[headOffset++]; // headOffset++，为下一次计算做准备
```

优化后滚动校验算法空间复杂度为 O(2*bufferSize), 2 *bufferSize 远小于 滚动窗口大小. 经过实验测试发现, bufferSize 大小为 512 byte 的情况下, 算法性能最优. 以下是滑动窗口性能优化算法的伪代码: (以HeadBuffer举例)这种为滑动窗口头尾数据设置缓冲区的思想, 能够有效的优化磁盘访问. 是滑动窗口内容全部存放内存和全部内容存磁盘每次读取的方式的折中, 兼顾了内存大小和磁盘读写速度的限制, 使算法更加适合并发的实际运行环境.

3.3 增量传输算法上传下载的流程优化

增量传输算法可以分别应用在上传和下载两个过程:

3.3.1.增量上传同步过程:

上传同步过程在本章开始已经介绍, 本节不再复述. 但在服务器端存储文件数据过程中, 增量传输额外能为存储效率的提升做铺垫.

因为上传文件的新旧版间的相同数据已经以文件块形式存储在底层存储池中. 结合 3 中提到的存储模型, 云存储系统可以对相同数据块进行冗余去重, 复用已经存储的旧版本的文件块. 因此, 需要更新的内容只是新版本文件的元数据信息和新版本改变量的文件块.

3.3.2.增量下载同步过程:

增量下载同步的过程, 即是大规模共享同步的典型场景. 此时, 服务器端文件比客户端文件版本更新, 需要将新版本文件同步到各个客户端. 并发下载的发送方是云端服务器, 大规模并发下载的情况下, 滚动校验也会在服务器端并发进行. 这对云端服务器资源无疑是很大的开销.

进一步研究发现, 文件下载的过程, 计算新旧版本文件改变量的过程可以进一步简化. 因为下载过程客户端文件的版本将低于云端服务器文件版本, 即客户端的文件版本是服务器端文件的一个历史版本. 由 2.2 中存储模型可知, 文件的所有历史版本都可以在服务器端底层存储中获得, 而且新旧版本是增量存储的. 因此, 通过将两个版本的文件的元数据在服务器端进行比较, 就能够确定哪些文件块需要传输到客户端. 此时, 可以将服务器端计算强弱校验码的开销转移给客户端程序.

综上所述, 云存储服务的多终端文件同步可以通过 Rsync 算法进行传输优化, 本论文提出了对 Rsync 算法的内存空间复杂度和磁盘访问频度的优化, 针对云存储服高并发的特点, 利用服务器端文件历史版本的元数据信息, 进行下载流程优化, 减少计算量; 新版本存储过程优化, 实现存储冗余去重, 提升存储空间利用率.

4　模拟实验及评价

本文提出了基于改变量的增量传输算法, 算法相

较全文本的文件同步算法性能有大幅度提升. 算法相较传统滚动校验算法在内存使用大小、磁盘读写性能、增量文件下载同步的流程等方面进行了优化. 为了更好地描述本论文的增量传输算法与传统滚动校验算法在数据传输方面的差异, 我们做如下实验,

(1)针对客户端文件做少量修改, 文件以 RAR 类型文件为例, 将新版本上传到服务器. 单个文件普通传输和增量传输的上传时间效率对比:

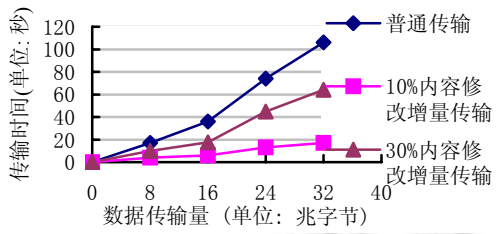

图 7 普通传输和增量传输时间开销对比

增量传输的时间开销由数据检测和数据传输两部分时间组成. 由上图可知, 在文件少量修改的情况下, 能够有效的侦测出文件数据改变量, 时间开销较小. 但在增量数据增大的情况下, 时间开销也会逐步增加.

(2)增量上传文件时, 不同的数据读取方式下, 滚动校验算法的时间开销的对比:

表 1 不同数据读取方式下滚动校验时间对比

文件大小 (M)	分块大小(k)	滚动校验时间对比			文件增量 (%)
		从磁盘读取 (s)	全存放内存 (s)	滑动窗口缓存	
10	10	4.089	0.442	0.544	2
10	30	6.485	0.847	0.959	3
10	50	6.982	0.380	0.503	4
20	10	13.828	0.792	1.067	3
20	30	16.060	0.553	0.784	4
20	50	17.584	0.502	0.726	5
30	10	26.787	1.145	1.457	5
30	30	64.703	1.671	2.035	12
30	50	90.295	0.887	1.365	19

由上图可知, 随着文件大小增大, 文件内存全部存放内存时的滚动校验算法的时间开销很小, 而每次从磁盘读取文件数据的时间开销越来越大. 本文采用的滑动窗口缓存技术的时间开销接近于全部存放内存, 远小于从磁盘读取的开销, 同时保证了较小的空间开销.

(3)共享同步时, 针对不同大小的文件, 分别计算优化同步下载流程前后的服务器进程计算开销:

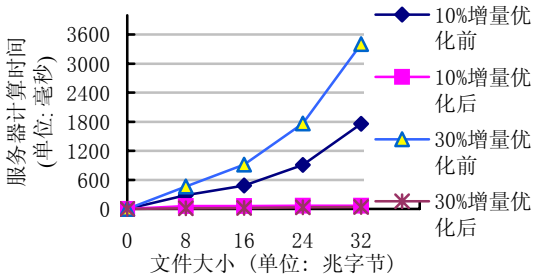

图 8 是否优化下载流程对服务器计算时间的影响

由上图可知, 共享同步的场景下, 1.优化下载流程前, 服务器计算时间随着传输增量的增大而增大; 2.流程优化后, 服务器计算时间基本保持在一个较小的值. 优化流程能够有效的减少服务器进程的计算时间, 达到让客户端为服务器分担计算量的目的. 服务器扮演分块元数据查询的角色, 下载过程的滚动由各个任务的客户端执行.

5 结语

在高并发的场景下, 同步服务器和客户端文件数据的过程, 如何减少内存、网络带宽、磁盘读写、存储等资源使用, 是云存储服务的关键问题. 本文提出了基于改变量的增量传输算法, 利用文件历史版本的相关性, 在资源使用和传输流程上进行优化. 实验证明, 优化后的基于改变量的增量传输算法在云存储环境下, 能够有效提升传输性能; 相同计算机资源的情况下, 支持更多的并发传输; 在存储过程中, 为存储冗余去重提供思路.

下一步的研究工作主要包括: 1)针对不同大小的文件提供大小不同的分块策略, 提供更灵活更精确的相同数据检测技术; 2)目前算法支持历史版本文件增量传输, 下一步研究对任意文件上传, 是否可以实现增量传输的效果.

参考文献

1 赵磊,郭晶,连思思.云存储系统技术综述.中国电子商情:通信市场,2013,4.29–33.
2 Tridgell A, Mackerras P. The Rsync algorithm. Tech. Rep. TR-CS-96-05, Australian National University, 1997.
3 Rasch D, Burns RC. In-place Rsync: File synchronization for mobile and wireless devices. USENIX Annual Technical Conference, FREENIX Track. 2003. 91–100.
4 Andrew T, Barker P, Mackerras P. Rsync in http. Auugn, 1999: 31–35.
5 Jung HM, et al. Energy efficient file transfer mechanism using deduplication scheme. Convergence and Hybrid Information Technology. Springer Berlin Heidelberg. 2011. 421–428.
6 敖莉,舒继武,李明强.重复数据删除技术.软件学报,2010,21(5):916–929.
7 汤晓迪,马晓旭,宁静,刘晓洁.远程文件差异同步系统的设计与实现.计算机工程与设计,2010,31(20):4389–4392.
8 胡晓勤,卢正添,刘晓洁,李涛,赵庆华,赵奎.远程文件快速同步方法.电子科技大学学报,2008,37(4):594–597.
9 Dropbox, Inc. Dropbox Website. http://www. dropbox. com.