计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(9):317−321 [doi: 10.15888/j.cnki.csa.008097]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

生物数据分析平台中的大文件多线程传输方案①

史若曦[1,3], 马俊才[2], 田园静[2], 张荐辕[2]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院 微生物研究所, 北京 100101)
[3](中国科学院大学, 北京 100049)
通讯作者: 史若曦, E-mail: shiruoxi@cnic.cn

摘　要: 随着云计算技术的飞速发展, 越来越多的数据相关服务需要通过云平台来实现. 国家科学微生物数据中心目前已建立了以云计算为基础的生物信息分析的数据平台, 为了解决平台用户进行文件上传时可能面临的效率、安全、稳定等一系列问题, 本文在分片上传、断点续传等技术的基础上, 提出一种根据带宽时延积选择并发线程数的传输方案, 解决了受意外中断时文件需要全部重新上传的问题, 提升了多线程上传时的带宽资源利用效率, 提高了大文件的传输效率.

关键词: 大文件传输; 多线程; 分片上传; 带宽时延积

引用格式: 史若曦,马俊才,田园静,张荐辕.生物数据分析平台中的大文件多线程传输方案.计算机系统应用,2021,30(9):317–321. http://www.c-s-a.org.cn/1003-3254/8097.html

Multi-Thread Transmission Scheme of Large Files in Biological Data Analysis Platform

SHI Ruo-Xi[1,3], MA Jun-Cai[2], TIAN Yuan-Jing[2], ZHANG Jian-Yuan[2]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](Institute of Microbiology, Chinese Academy of Sciences, Beijing 100101, China)
[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: With the rapid development of cloud computing technology, more and more data-related services need to be implemented through cloud platforms. The National Microbiology Data Center has established a data platform for bioinformatics analysis based on cloud computing. To address the problems related to efficiency, security, stability, etc. that users of the platform may face when uploading files, this paper presents a transmission scheme for selecting the number of concurrent threads according to the bandwidth-delay product based on the technologies of piecewise upload and broken-point continuingly-transferring. This scheme solves the problem that all files need to be re-uploaded when accidentally interrupted and improves the utilization efficiency of bandwidth resources during multi threaded upload and the transfer efficiency of large files.

Key words: large file transfer; multi-thread; slice upload; bandwidth delay product

随着微生物数据的快速增长, 研究者越来越重视微生物大数据的高效管理和分析. 国家科学微生物数据中心建立了以云计算为基础的生物信息分析的数据平台[1], 用户可以登陆网站, 根据需求在线使用生物信息分析工具. 由于网站面向所有用户开放, 一般生物文件较大, HTTP 协议下云服务器端对文件接收大小也有

① 基金项目: 国家重点研发计划 (2016YFC0901702, 2017YFD0400302, 2016YFC1200804, 2017YFC1201202); 中国科学院战略性先导科技专项 (A 类) (XDA19050301)
Foundation item: National Key Research and Development Program of China (2016YFC0901702, 2017YFD0400302, 2016YFC1200804, 2017YFC1201202); Category A Strategic Priority Research Program of Chinese Academy of Sciences (XDA19050301)
收稿时间: 2020-12-15; 修改时间: 2021-01-18; 采用时间: 2021-02-02; csa 在线出版时间: 2021-09-02

一定的限制, 用传统的插件技术、表单直传等方法进行文件上传时效率过低, 网络波动或浏览器异常等突发情况导致上传中断时也需要对整体文件进行重复上传[2]. 所以, 本文设计了一个与分片传输技术相结合的多线程传输方案, 保证用户数据可以安全、快速的传递至云平台.

1 现状分析

1.1 相关技术

传统的基于 Web 文件上传的方法有 Form 表单上传, 该方法要求表单里包含一个类型为 file 的输入文件框, 与 JavaScript 等技术结合, 实现文件的上传操作. 插件技术是另一种实现 Web 端文件上传的方法, 主要利用 ActiveX、Huploader 等一系列插件实现文件上传, 但是该技术容易因浏览器的设置导致插件运行失败, 因此只适合在内网此类安全的内部环境使用[3].

除此之外, 文献 [4] 提出了一种断点续传方案, 大文件在 Web 端分片后, 通过计算出每片的 MD5 值来做为其唯一标识符. 当网络异常发生传输中断时, 由唯一标识符来确定断点续传从分片相应部分开始. 文献 [5] 提出了一种多线程控制的方式. 该方法在传输多个不同文件时, 为每个文件开启一个线程来控制内容文件和配置文件, 虽然在传输多个文件时有明显优点, 但针对单个文件时效率不高. 文献 [6,7] 采用双线程分别记录文件内容和内容偏移量的方法, 该方法在实现断点续传的情况下牺牲了传输效率. 文献 [8] 在线程选择上采取逐步增加的方式, 虽然提高了传输效率, 但是逐步增加试探, 在网络承载率大的情况下仍然有一定程度的资源浪费.

1.2 存在的问题

尽管上述技术已经对断点续传、重复文件上传等问题进行了一定程度的解决, 但针对如何提升大文件传输速率这一问题仍存在不足. 部分技术为了实现断点续传, 采用多线程来记录文件传输时产生的相关配置文件, 造成了带宽资源的浪费. 一些技术虽然提出利用多线程来实现文件并发传输, 但并没有具体提出如何选择并发线程数来保证尽可能高效的利用网络带宽.

因此本文提出了一种根据带宽时延积 (BDP) 来选择并发线程数的文件传输方案, 与分片上传、断点续传技术相结合, 满足生物数据分析平台用户的需要, 实现生物数据的稳定上传.

2 基于 BDP 的多线程方案设计

2.1 理论分析

在实际的传输过程中, 数据每份发送后无法立即得到确认, 这些在信道中传输但还未被确认的数据量通过用带宽时延积 (BDP) 来表示, 它是衡量网络链路能力和承载能力的关键指标. 窗口机制就是防止数据量超过接收端确认处理能力的一种措施, 它通过限定窗口大小的方式来进行 TCP 的流量控制和拥塞控制. 滑动窗口和固定窗口是常用的两种窗口机制, 但是由于 TCP 报文头中窗口字段大小只有 16 位, 无论哪种方式, TCP 窗口最大值都为 64 KB. 在理想的宽带利用率下, 带宽时延积应与 TCP 窗口大小一致. 在 1000 Mb/s 的网络带宽下, 只有往返时延在小于 0.448 ms 时, BDP 才会小于 64 KB, 此时能够有效利用带宽. 但是在实际的网络情况下, 要想达到这么小的往返时延几乎是不可能的. 因此, 对于并发 TCP 连接, 通过同时建立多条连接, 并发 n 条连接就相当于将窗口大小扩大了 n 倍, 从而能有效的提高传输速率[9].

2.2 理论分析

根据前文对 TCP 窗口机制与带宽时延积的理论分析可知, 为了实现高效的利用网络带宽, 减少数据等待确认时的资源浪费, TCP 发送窗口大小应与带宽时延积一致. 然而受到报文头字段大小限制, 要想在生物数据分析平台用户的网络条件下提高文件传输效率, 应采用多线程的方法建立多条 TCP 连接, 扩大传输窗口.

对于并发 TCP 来说, 理想的并发数 N 可以用式 (1) 计算.

$$N=\frac{BDP}{MaxWindowSize} \tag{1}$$

其中, BDP 为带宽时延积, 由网络带宽与传输时延 (RTT) 共同确定, 其计算方式为 BDP=带宽 × RTT, $MaxWindowSize$ 为最大窗口数.

3 方案设计及实现步骤

本文综合分片传输和断点续传等技术, 提出一种根据用户网络带宽时延积选择最佳并发线程数的传输方案. 该方案采用部分功能函数, 实现 Web 端的文件分片传输, 并根据 MD5 的计算原理, 采用分片计算后整合的方式, 最终得到原文件的 MD5 值. 在解决相同文件上传时, 使用 MD5 校验来检测服务器端是否已经存储该文件[10], 提高传输效率. 本文的关键在于设计了

一个多线程创建方式, 在用户打开网站时获得此时的网络状态, 根据此参数得到当前网络状态下用户进行上传操作时可使用的最大线程数, 进而提升了文件的上传效率. 文件发送流程如图 1 所示.

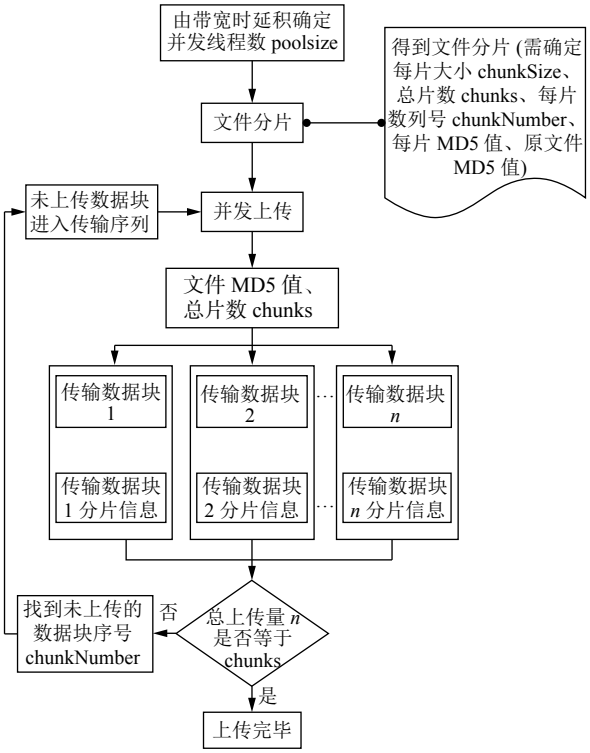

图 1 客户端文件发送流程

3.1 确定线程数

线程数的选择主要依据网络带宽和传输时延, 该参数可以在用户与服务器建立连接时测定, 通过 BpsDataPerInterval 方法, 获取连接服务器时的带宽传输时延 *RTT*, 根据上文公式, 确定并发数 *poolSize*=(*RTT*×带宽)/(64×1024), 存储在 *poolSize* 函数中, 在进行上传时, 启动与服务器的线程连接.

3.2 分片

因为服务器端会限制每次上传文件的大小, 所以需要在前端指定每片文件的值. 如果分片较小, 则分片数大, 会导致多次建立传输请求, 增大开销; 如果分片较大, 则会降低灵活度[11]. 综合数据分析平台服务器端的设置要求, 本文设置每片的大小为 2 MB, 即 chunkSize=1024×1024×2, 此外还需要的参数有, chunkNumber 表示分片序数, chunks 表示分片总数, 用于服务器端的文件合成. 其合并流程如图 2 所示.

3.3 MD5 值计算

生物信息分析平台中的分析工具所需生物数据文件大都在 600 MB 以上, 甚至大到十几 GB, 如果采用整体计算文件 MD5 值的方式, 容易导致内存占用过大, Web 端异常崩溃等情况, 计算效率也相对较低.

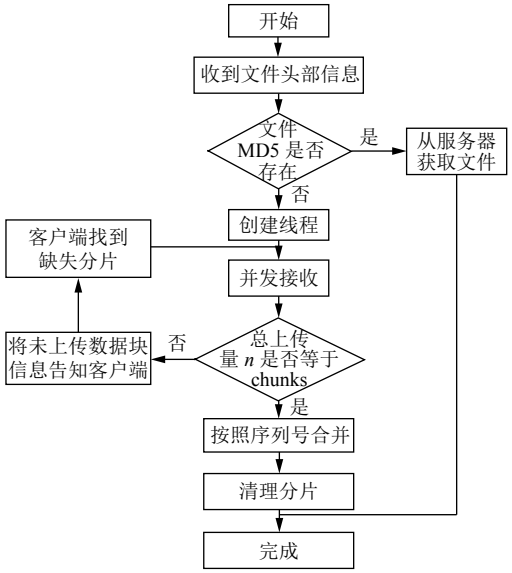

图 2 服务器端文件合并流程

由于 MD5 的计算特性, 分片计算每部分的 MD5 值后, 再进行合并不改变原文件的 MD5 值[12], 因此本文采用新的计算方式. 将文件分片后逐个传入 spark.appendBinary() 方法来计算、最后通过 spark.end() 方法输出 MD5. 这种方法节约内存开销, 在计算大文件 MD5 值效果更好. 根据前文设置, 分片大小为 2 MB, 综合文件大小得出总片数, 然后设置 file.cmd5 = true, 即文件状态改为 MD5 计算. 接着逐片读取分片信息, 并计算 MD5 值, 由 spark.end() 得出所有值后, 将总文件的 MD5 值赋给 file.5;, 作为该文件唯一标识, 为秒传和断点续传操作提供方便. 最后取消计算状态, 并开始上传文件. 其相关代码如下:

```
//计算 MD5
computeMD5(file) {
chunkSize = 2 097 152, //2 MB
chunks = Math.ceil(file.size/chunkSize),
currentChunk = 0,
spark = new SparkMD5.ArrayBuffer(),
fileReader = new FileReader();
let time = new Date().getTime();
file.cmd5 = true; //文件状态为“计算 md5…”
fileReader.onload = (e) => {
```

```
spark.append(e.target.result);
currentChunk++;
if (currentChunk< chunks) {
loadNext();
} else {
console.log('finished loading');
let md5 = spark.end(); //得到 md5
file.uniqueIdentifier = md5; //将文件 md5 赋值给文件唯一标识
file.cmd5 = false; //取消计算 md5 状态
file.resume(); //开始上传
}
loadNext();
}
```

3.4 续传实现

受异常情况中断后, 整体文件重传需要很大的代价, 本文利用已经计算得出的文件 MD5 值作为文件的特殊标识符, 在进行文件重传时, 通过 MD5 校验判断文件是否已经上传[12], 然后再进行传输操作. 本文采用 checkChunkUploadedByResponse() 函数响应后台返回的信息, 并检测分片信息是否上传完整. 分片上传前, 前端会向后端发送一个携带文件信息的 get 请求. 如果文件已经在服务器端存储, 则返回 obj.isExist, 后续上传操作不需要继续执行. 如果返回的是文件分片信息, 则表示该部分已经上传, 执行续传操作. 其相关代码如下:

```
//续传实现
checkChunkUploadedByResponse: (chunk, message) => {
let obj = JSON.parse(message);
if (obj.isExist) {
this.statusTextMap.success = '秒传文件';
return true;
}
return(obj.uploaded||[]).indexOf(chunk.offset + 1) >= 0
},
//检测断点和 MD5
public function checkFile()
{
//检测文件 MD5 是否已经存在
$rs=$this->checkMd5($identifier, $this->fileInfo['totalSize']);
if ($rs['isExist'] === true) {
return $rs;
}
//检查分片是否存在
$chunkExists = [];
for ($index = 1; $index <= $totalChunks; $index++) {
if (file_exists("{$filePath}_{$index}")) {
array_push($chunkExists, $index);
}
}
```

3.5 文件合并

本文采用的是多线程传输, 为了保证传输的准确性, 需等待所有分片传输成功再进行合并. 当传输的分片数等于总分片数 chunks 时, 会向后台发送合并请求. onFileSuccess() 方法接收从后台返回的 response 包含了是否需要合并的指令 merge, 如果 resp.merge === true, 则向后端发送合并请求. 前端将文件的唯一 ID 和拆分总数 (或要传递的更多参数) 发送到合并文件的后端. 后端受到合并指令后开始进行文件合并其相关代码如下:

```
//文件合并
public function merge()
{
$filePath=self::$tmpDir. DIRECTORY_SEPARATOR. $this->fileInfo['identifier'];
$totalChunks=$this->fileInfo['totalChunks']; //总分片数
$filename=$this->fileInfo['filename']; //文件名
$done = true;
//检查所有分片是否都存在
for ($index = 1; $index <= $totalChunks; $index++) {
if(!file_exists("{$filePath}_{$index}")) { $done = false;
break;
}
}
if ($done === false) {
return $this->message(1005, '分片信息错误');
```

```
    }
    //如果所有文件分片都上传完毕, 开始合并
    $timeStart = $this->getmicrotime(); //合并开始时间
    $saveDir = self::$saveDir.
    DIRECTORY_SEPARATOR. date('Y-m-d');
    if (!is_dir($saveDir)) {
    @mkdir($saveDir);
    }
    $uploadPath=$saveDir.DIRECTORY_SEPARATO
R.$filename;
    if (!$out = @fopen($uploadPath, "wb")) {
    return $this->message(1004, '文件不可写');
    }
    if (flock($out, LOCK_EX)) { // 进行排他型锁定
    for($index=1;$index<=$totalChunks; $index++) {
    if(!$in=@fopen("{$filePath}_{$index}", "rb")) {
    break;
    }
    while ($buff = fread($in, 4096)) {
    fwrite($out, $buff);
    }
    @fclose($in);
    @unlink("{$filePath}_{$index}"); //删除分片
    }
    flock($out, LOCK_UN); // 释放锁定
    }
    @fclose($out);
    return $res;
    }
```

4 实验结果分析

根据实际网络情况, 本文进行了带宽时延积 *BDP* 为 75 KB 和 135 KB 两种情况下的文件传输测试.

如表 1 所示, 当 *BDP* 大于 64 KB 时, 该网络情况可以进行多线程传输, 其效率相较单线程传输有一定的提高.

5 结论

本文提出利用带宽时延积和最大窗口数计算得到网络最大承载率, 来决定并发线程数的文件上传方法, 充分利用网络带宽资源, 采用 MD5 值标识已经上传过的文件, 实现生物数据分析平台用户的文件高速上传, 节约了时间成本. 该方法解决了一般分片上传过程中, 无法确定并发线程数的问题, 能够提高大文件的上传效率, 增强传输稳定性.

表 1 不同 *BDP* 下传输耗时的测试结果

BDP (KB)	文件大小(MB)	可承载线程数	传统方案耗时	本方案耗时
75	276	1	5'15"	5'29"
75	642	1	14'23"	14'12"
135	276	2	1'44"	53"
135	642	2	4'12"	2'40"

参考文献

1 亓合媛, 孙清岚, 马俊才. 微生物组大数据管理与分析. 微生物学报, 2017, 57(6): 932–941.

2 黎苑文, 程明智, 徐秀花, 等. 断点续传及多线程机制在远程传版中的应用研究. 北京印刷学院学报, 2012, 20(6): 53–56. [doi: 10.3969/j.issn.1004-8626.2012.06.019]

3 王建斌, 赵靓. Web 上传文件的三种解决方案. 计算机与信息技术, 2011, 19(S1): 65–68.

4 王莉敏, 梁正和, 段全锋. 基于 HTML5 大文件断点续传的实现方案. 计算机与现代化, 2016, (3): 91–95. [doi: 10.3969/j.issn.1006-2475.2016.03.018]

5 夏雪刚. 基于多线程文件传输关键技术研究与实现. 电脑知识与技术, 2016, 12(21): 48–50.

6 刘静敏. 广播节目传输中的断点续传与多线程技术探析. 黑龙江科技信息, 2016, (15): 98.

7 秦绪彬. 广播节目传输中的断点续传和多线程技术运用. 黑龙江科技信息, 2016, (19): 18.

8 阮晓龙, 李朋楠. 基于 Web 的大文件高效上传方法. 计算机系统应用, 2020, 29(3): 234–239.

9 邓彬. 大文件的快速传输研究与实现 [硕士学位论文]. 南京: 南京邮电大学, 2019.

10 段国云, 陈浩, 黄文, 等. 一种 Web 程序防篡改系统的设计与实现. 计算机工程, 2014, 40(5): 149–153. [doi: 10.3969/j.issn.1000-3428.2014.05.031]

11 邹鹤敏, 黄海于. 大文件分块上传和下载软件的设计与实现. 电子技术应用, 2013, 39(8): 137–139. [doi: 10.3969/j.issn.0258-7998.2013.08.040]

12 胡渝苹. 文件秒传系统在云存储环境下的设计与实现. 计算机应用与软件, 2016, 33(4): 329–333. [doi: 10.3969/j.issn.1000-386x.2016.04.076]