一种 Web 中快速传输大型文件的方法①

唐 威, 景奕昕

(武汉华工安鼎信息技术有限责任公司, 武汉 430223)

摘 要: 实现了 WEB 应用程序中快速传输大型文件的机制. 浏览器端采用 HTML5 标准中文件系统的访问特性, 对上载的文件首先进行分片实现并行上传. 服务器端采用CGI程序接收浏览器端上传的文件. 后台管理服务对上传的分片文件进行管理. 下载文件时通过 CGI 请求后台管理服务, 按序发送文件分片. 整个过程均不涉及文件分片的重组过程, 减少了磁盘 IO 请求. 与其它文件上传下载方式相比, 本方法具有上下载文件大小无限制, 传输速度高及不需要第三方控件等特点.

关键词: HTML5; CGI; 大尺寸文件传输; 异步传输; 文件分片

Approach for High-speed Transfer of Big File in Web

TANG Wei, JING Yi-Xin

(Wuhan Huagong Andin Information Technology Co. Ltd, Wuhan 430223, China)

Abstract: The paper implemented a mechanism for transferring big size file in high speed. The browser leverages Html5 to slice big file into pieces and upload files to web server asynchronously. The web server adopts CGI to handle incoming file pieces, and sends file pieces in order when the file is downloaded. The mechanism reduces I/O cost due to avoiding reunifying file pieces. Comparing other approaches, this method breaks through the limit to file size, transfers files in high speed, and features in no need of third-party plug-ins.

Key words: HTML5; CGI; big file transfer; asynchronous transfer; file split

由于云端服务的不断发展, 越来越多的用户将本地文件通过浏览器上传至云端服务器以实现云共享[1]. 然而现有 WEB 应用文件传输机制, 例如 IIS/Apache 环境下, 由于服务器配置等原因对文件上传的大小, 连接时限上有诸多限制, 通过浏览器传输上百兆甚至上千兆的大型文件速度慢、易超时, 严重影响用户体验. 采用浏览器插件的方式又存在配置不自由、安全性问题[2]. 本文利用浏览器支持的 HTML5 特性, 采用 AJAX 技术实现文件的分片并发上传, 借助 epoll 和 WEB 服务端的 CGI 程序配合, 突破了文件传输的尺寸限制, 极大提高了文件上传下载速度, 同时实现了文件分块和多线程的处理.

1 epoll

epoll[3]是 Linux 下多路复用 I/O 接口 Select/poll 的增强版本. 它能显著提高程序在大量并发连接中只有少量活跃情况下的系统 CPU 利用率. 它的 I/O 效率不会随连接数的增加而下降. epoll 模型并不采用传统的轮询方式扫描整个 socket 集合, 而只给用户返回活跃的 socket. 另外, epoll 通过内核与用户空间 mmap 同一块内存加速内核与用户空间的消息传递. 文件上传下载时, 由于存在大量并发连接, 采用 epoll 能保证其 I/O 不会在活跃情况降低时消耗过多 CPU 资源.

2 流程概述

文件上传下载采用了异步通信的方式, 在不刷新页面的同时对文件进行分块上传. 后台服务器响应浏览器端的请求, 对文件块进行存储并管理. 图 1 描述了文件上传下载的流程. 本节对文件上传下载总体流程进行概述.

① 收稿时间:2012-04-10;收到修改稿时间:2012-05-07

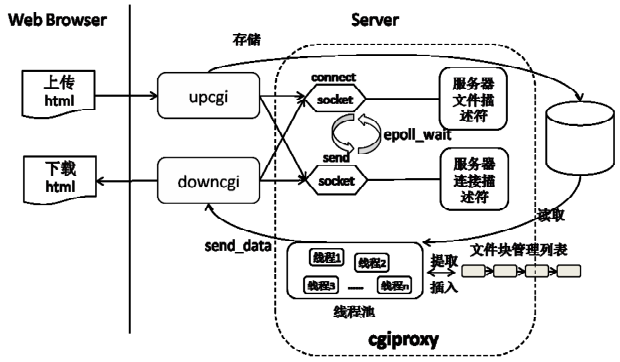

图 1 文件上传下载流程

在 WEB 服务器端运行一个 cgi 代理, cgiproxy. 该代理首先创建一个服务监听, 用于接受来自 CGI 程序的命令请求. cgiproxy 采用 epoll 轮询事件 event 并管理一个线程池来处理监听到的 event.

当上传在 web 界面中被触发时, 客户端的 JS 脚本程序获取文件, 调用 webkitSlice()或 mozSlice()方法, 将需要上传的文件切割成多个文件块 chunk. 文件块的大小可由程序指定, 默认为 1MB. 同时, 浏览器计算出 chunk 在所属文件的 chunk 序号, chunk 的偏移地址和所属文件总共的 chunk 数, 记为<fseq, foffset, fcks>. 然后, 浏览器新建一个 XMLHttpRequest 对象, 将文件控制块<fuid,fname,fseq, foffset, fcks>置于请求头内, 将 chunk 置于请求体内, 以 POST 方式请求 WEB 服务端的 upcgi 进行处理.

服务器端 upcgi 收到上传页面的 POST 请求后, 从请求中提取出对应的参数<fseq, foffset, fcks>文件控制块和文件 chunk, 对 chunk 直接进行存储. 将文件控制块发往 cgiproxy 所监听的端口, 即产生一个通知 event. cgiproxy 在收到 upcgi 的 event 后, 将上述信息提取出来存入文件管理数据库中.

下载的过程与上传相似. 不同的是 cgiproxy 在监听到 downcgi 发送的下载请求 event 后, 从文件块管理列表中按序依次获取各文件块的地址, 然后通过 socket 将文件块直接发送给 downcgi.

不论上传还是下载, 都不涉及文件分片的重组问题, 显著减少磁盘 IO 数量.

3 Cgiproxy

3.1 传输参数结构

由于 cgiproxy 统一处理上传下载命令, cgiproxy 需要根据命令种类不同读取不同的命令参数. 以上传过程为例. upcgi 发送给服务器 socket 命令格式如图 2 所示. __comm_bk.cmd 被赋值为 UP. __comm_bk.fcb 即为文件上传的参数结构体__upfctl_bk. 其中, 文件块序号和文件 id 会被用来生成文件块存储时使用的块文件名. __comm_bk.len 的值为__upfctl_bk 的长度, 即 sizeof(struct __upfctl_bk). 对于 downcgi 来说, __comm_bk.cmd 的值为 DOWN, 但__comm_bk.fcb 的类型为文件下载的参数结构体, __comm_bk.len 也为下载参数结构体的长度. __comm_bk 会被通过 socket 发送至 cgiproxy.

struct __comm_bk	
int	cmd
int	len
struct __upfctl_bk	fcb

struct __upfctl_bk		
short	cport	客户端端口
short	fseq	文件块序号
short	fcks	文件总块数
int	caddr	客户端地址
unsigned int	foffset	文件偏移开始
unsigned int	flen	文件块长度
char	ffid	文件id

图 2 文件上传命令结构

从 upcgi 发送的__comm_bk 最终可被 cgiproxy 读入数据缓冲区. 在 cgiproxy 端同样也定义了__comm_bk 和__upfctl_bk 结构体. 所不同的是, __comm_bk 结构体只含有 cmd 和 len 两个属性. cgiproxy 从 socket 根据 len 属性进行读取数据, 再依据 cmd 的值, 将读入的数据置换成相应的结构体. 例如 cmd 值为 UP 时, 数据被置换为__upfctl_bk.

3.2 cgiproxy 处理

cgiproxy 一直运行在后台, 负责在初始阶段注册 epoll 监听事件, 并响应 upcgi 和 downcgi 的请求. cgiproxy 的主流程如图 3 所示. Cgiproxy 最初监听服务器描述符的 socket 请求. 在收到 upcgi/downcgi 调用 connect()函数后产生的连接请求后, cgiproxy 调用 socket 的 accept()函数生成连接描述符, 并重新在 epoll 文件描述符上注册. 连接建立后, upcgi/downcgi 的传输事件即会被 epoll_wait()收集. 由于现在 epoll 上注册的描述符已不是服务器描述符, 事件会被加入 event 队列.

在 cgiproxy 启动时, 初始化了 10 个线程在等待处理 event 队列中的事件. 一旦队列中有事件进入, process_queue()方法会取出 event, 并从 event->data.fd 中读取服务器连接描述符 cs, 并进一步从 cs 中读取结

构体为__comm_bk 的数据. 后续操作如 4.1 所述. 在上传文件的情况下, process_queue()会建立 map=<ffid, list(<fseq, fck>)>对同属一个文件的文件块进行管理. 在文件下载的情况下, process_queue()会在 map 中查找 ffid 相对应的 fseq 值, 并与 ffid 组合得到下载文件的块文件名, 进而执行文件块读取, 通过连接描述符 cs 送往 downcgi.

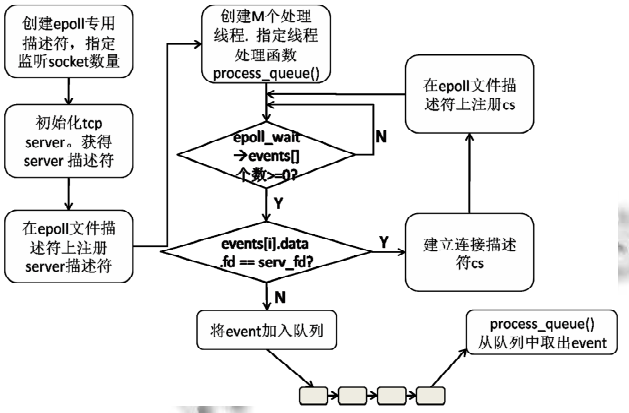

图 3　Cgiproxy 处理流程

4　效率评估

从上述的流程可发现, 大尺寸文件在浏览器端被分割后, 在服务器端被直接存入文件系统. 相比 Apache 服务器先存储为临时文件, 再整体移动到目标路径的方式[4], 本文的机制更为高效. 为了评估本文方法的上传效率, 我们针对不同文件大小, 分别采用本文方法(记为 html5)、jQuery[5]、和 Flash 三种方式, 在 Chrome 和 Firefox 中进行上传试验(Firefox 不支持 Flash 上传). 上传测试环境为服务器端 CPU3.3GHz, 内存 512M, 百兆网卡, Apache2.2; 客户端 CPU3.3GHz, 内存 3.4G, 百兆网卡. 图 4 显示了不同体积的文件, 在不同浏览器, 采用不同方式时的上传速度. 从图中可见, html5\jQuery\Flash 三种上传速度比较中, html5 分块并行对于不同大小的文件上传速度始终较快, 而且保持平稳, 大约为 11.5MB/S, 但超过 1G 的文件 Firefox 浏览器不能上传, 仅 Chrome 支持.

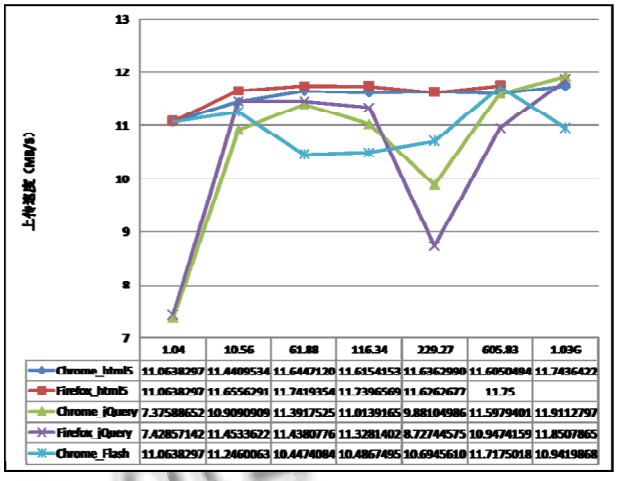

图 4　文件上传速度比较

5　总结

由于云环境下电子文档普遍采用集中存储, 大尺寸文件上传常常受限于 Web 服务器的能力而不能实现高效存储和访问. 本文采用 Html5 结合 CGI 的方式, 实现了超大文件的上传下载. 实验结果表明, 对于上百兆甚至超过 1G 的文件, 本文的方法能保持很高的传输速率. 但是, 由于文件分片需要浏览器支持 Html5, 对客户端机器有一定要求, 以后的研究方向会关注浏览器端的优化问题.

参考文献

1 何海东,张文秋.基于 Web 的网络硬盘的设计与实现.四川理工学院学报(自然科学版),2010,2:175–177.

2 Hayes F. Developers worried about safety of Java and ActiveX. Computers and Security, 1996,15(8):687–687(1).

3 白英.Linux-高档微机 UNIX 操作系统(上).计算机系统应用,1996,15(3):52–55.

4 Apache. http://commons.apache.org/fileupload/apidocs/

5 jQuery File Upload. http://blueimp.github.com/jQuery-File-Upload/.