计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):11−18 [doi: 10.15888/j.cnki.csa.006091]

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

基于 HTML5 的浏览器端多线程下载技术①

任双君[1,2], 周　旭[1], 任勇毛[1], 李灵玲[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘　要: 针对传统浏览器单线程下载效率低下、过度依赖目标服务器的问题, 研究提出了基于 HTML5 的浏览器端多线程下载技术. 基于 HTML5 Web Workers 技术, 实现了浏览器端多线程下载功能; 利用分段下载技术, 实现了单一文件的多源下载; 利用 HTML5 File System API 加 Blob 对象的技术, 实现了浏览器端文件片段的合并功能. 实验结果表明, 本文提出的方法对于大文件下载, 或者高延迟、高丢包率的网络下载环境, 效率明显优于单线程下载技术.

关键词: HTML5; 多线程下载; 多源; Web Workers; HTML5 File System

引用格式: 任双君,周旭,任勇毛,李灵玲.基于 HTML5 的浏览器端多线程下载技术.计算机系统应用,2017,26(11):11–18. http://www.c-s-a.org.cn/1003-3254/6091.html

Multi-Thread Downloading Technology Based on HTML5

REN Shuang-Jun[1,2], ZHOU Xu[1], REN Yong-Mao[1], LI Ling-Ling[1]

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

Abstract: This paper mainly studies the technology of multi-thread downloading in browser, which aims to solve the problem of low efficiency of single thread downloading and over reliance on the target server. Based on HTML5 Web Workers technology, we propose and implement a novel multi-thread downloading technology which runs in the browser. Though segmenting file to chunks, we download a file from multiple sources. We use ArrayBuffer array and Blob objects to merge the file fragments in the browser. The results show that this method is superior to the single thread downloading technology in the large file downloading, large network delay or high packet loss rate.

Key words: HTML5; multi-thread download; multi-source; Web Workers; HTML5 File System

目前传统的浏览器只能提供面向目标服务器的单线程下载技术, 但是由于网络环境的复杂性, 在资源下载的过程中经常存在高网络延迟、高丢包率、带宽限制等问题, 单一线程下载效率低下. 而且由于只向单一服务器请求资源, 这样就存在过度依赖目标服务器的问题, 当目标服务器出现问题, 下载只能被迫中断.

为了提高下载效率以及下载系统的鲁棒性, 产生了迅雷、BitTorrent、eMule 等多线程下载工具. 但是现有的多线程下载工具大多基客户端或者浏览器插件, 需要用户下载安装相应的客户端或者插件[1,2], 难以满足用户对下载过程简便性的要求, 不具有普适性.

随着 Web 开发技术的发展, HTML5 技术也逐渐发展成熟. 和传统的应用程序相比, HTML5 应用程序具有跨平台、无需安装、低成本开发、速度快等优点[3,4], 用户无需下载并安装软件, 一个简单的浏览器和可供访问的网络设备就可以满足用户访问应用程序简便性

① 基金项目: 国家自然科学基金(61601443, 61602463, 61672490); 北京市自然科学基金(4174110); 中科院计算机网络信息中心所级项目(ZXRW-201601)
收稿时间: 2017-02-20; 修改时间: 2017-03-27; 采用时间: 2017-03-31

的需求.

HTML5 的诸多特性使得浏览器端的多线程下载成为了可能. 首先, HTML5 Web Workers[5,6]接口规范为浏览器实现多线程并发访问提供了可能性, 该接口允许浏览器在后台 I/O 运行的同时不影响前端网页对用户的响应. Web Workers 允许在 Web 程序中并发执行多个 JavaScript 脚本, 每个脚本执行过程都作为一个线程, 各个线程之间彼此独立, 由 JavaScripte 引擎负责管理. 此外, Web Workers 还可以使用 XMLHttpRequest 执行 I/O. 一旦创建, 一个 worker 可以将消息发送到创建它的 JavaScript 代码, 通过将消息发布到该代码指定的事件处理程序[7].

此外, HTML5 提供了可以从浏览器访问本地文件系统的接口即 File System API[8,9], 它的主要作用是将本地文件以文件对象的形式提供给 Web 应用程序进行访问, 为浏览器端应用程序的开发提供了无限可能[13,14]. 在传统浏览器端应用程序中, 出于安全因素考虑, 浏览器无法直接和本地文件系统进行交互, 在这种情下浏览器本身的功能被极大的限制, HTML5 file system API 的出现为浏览器端应用程序可以更加便利的操作本地文件.

本文针对传统浏览器单线程下载效率低下、过度依赖目标服务器的缺点, 基于 HTML5 的 Web Workers 多线程技术, 提出了一种浏览器端的多线程下载技术, 实现了浏览器端文件的分段、多线程请求服务器资源、文件块的合并与文件下载功能. 传统 Web Workers 应用大多利用子线程完成复杂、耗时的计算任务, 本文创新性地提出了利用 Web Workers 实现了浏览器端多线程下载技术, 本方案通过浏览器主线程创建多个子线程, 由多个子线程向多个服务器并发请求文件数据, 提高文件的下载效率, 最终实验证明了该技术的可行性及高效性, 为浏览器端下载行为提供了更多的可能性.

1 系统概述

浏览器端多线程下载系统以资源内容为核心, 通过资源索引服务器将资源内容和资源地址进行分析映射, 将基于目标地址的单一地址扩展为基于资源内容地址列表的多源地址; 通过浏览器端主线程对资源进行分段, 子线程完成资源片段的下载, 最终由浏览器端主线程完成整个文件的合并, 写入到本地文件系统中, 实现了浏览器端多线程下载功能.

系统核心主要分为资源索引服务器和浏览器端两部分, 其中资源索引服务器主要包括资源分析模块, 浏览器端主要包括资源下载模块和写操作模块.

资源分析模块: 对目标地址进行分析, 通过对目标地址和目标资源内容进行映射, 根据目标资源内容查询资源索引服务器获得该资源对应的可用地址列表, 最终将面向用户资源的单一下载地址扩展为多个下载地址组成的资源地址列表. 该模块主要解决了系统在进行多线程下载之前, 如何通过现有下载地址找到其他可用下载地址的问题.

资源下载模块: 采用基于 HTML5 Web Workers 浏览器端多线程技术, 主线程主要负责子线程创建及管理、资源片段分配工作, 子线程主要负责和目标服务器交互及资源片段下载工作, 通过主线程与子线程的协同工作完成资源的并发下载. 该模块主要解决了系统在多线程下载过程中, 如何建立和管理浏览器端与可用下载源的并行连接的问题.

写文件模块: 采用了基于 HTML5 File System[10,11]的浏览器端本地文件系统写操作技术, 通过对资源片段进行分析, 由主线程将资源片段按照资源片段编码进行排列存储, 当所有资源片段都下载完成, 主线程将资源片段整体存入到本地文件系统中, 完成下载过程. 本系统主要解决了系统在多线程下载完成后, 如何将从各源站下载的文件片进行合并、拼接和保存的问题.

系统的整体设计框架如图 1 所示, 该图以三个可用下载站点为例.

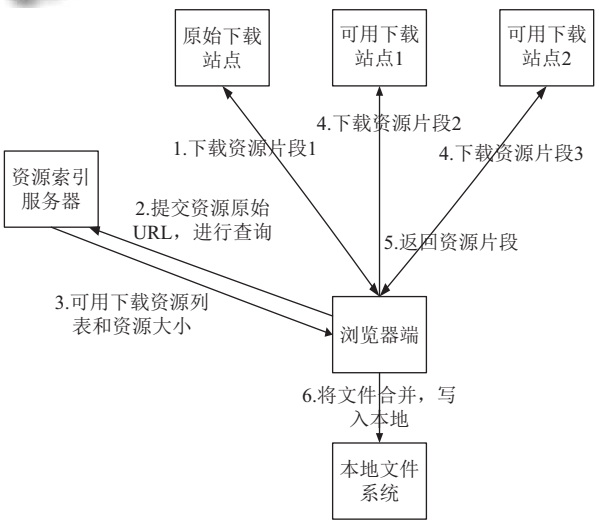

图 1 系统整体设计框架

系统的整体设计工作流程如下:

(1) 用户点击下载后, 浏览器端首先去原始下载站点请求并下载一部分资源片段.

(2) 浏览器端向资源索引服务器提交原始下载 URL, 查询其他可用下载站点服务器地址.

(3) 资源索引服务器将可用资源列表以及资源文件大小返回给浏览器端.

(4) 浏览器端对文件进行分段, 创建子线程, 并子线程, 各个子线程分别到自己的目标服务器下载资源片段.

(5) 下载站点服务器将资源片段返回给浏览器端.

(6) 浏览器端对资源片段进行合并后写入到本地文件系统, 下载结束.

2 系统设计与实现

2.1 资源索引服务器

资源索引服务器端的主要功能由资源分析模块完成, 该模块的主要功能包括可用资源地址查询和资源索引服务器的动态更新.

资源分析模块的工作流程如图 2 所示.

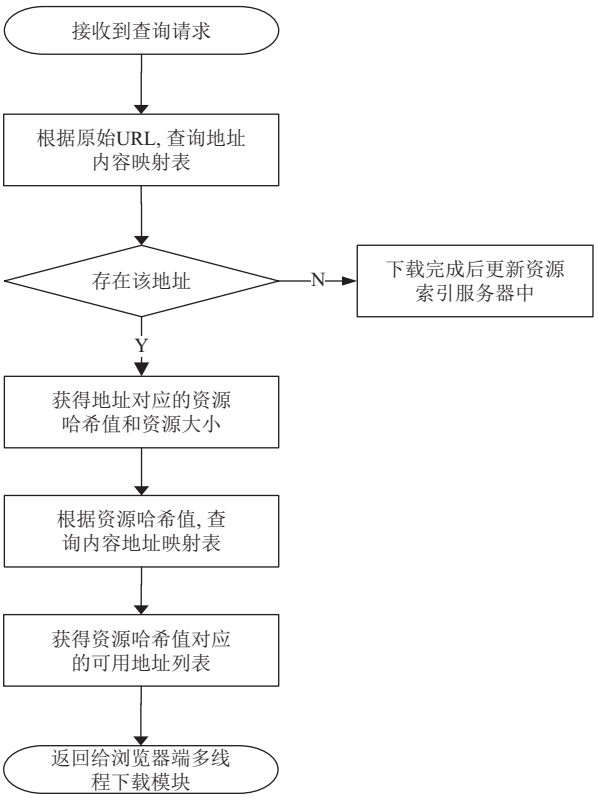

图 2 资源分析算法流程图

系统以资源内容为基础, 构造了资源索引服务器, 通过在文件下载阶段动态更新资源索引服务器, 获得资源在网络中的分布情况. 资源索引服务器主要包含两个表, 分别为地址内容映射表和内容地址映射表.

地址-内容映射表: 存放原始资源下载地址和其对应的资源内容哈希值, 以及该资源的大小.

内容-地址映射表: 存放资源内容哈希值和其对应的可用资源服务器地址.

资源索引服务器在接收到浏览器端发送的查询请求后, 首先根据目标 URL 查询地址内容映射表, 获得该地址对应的目标资源哈希值和资源大小. 然后根据该资源哈希值, 再次查询内容地址映射表, 获得目标文件对应的可用下载地址列表, 最终将单一目标地址扩展为多个目标地址组成的资源地址列表, 为多线程下载提供了基础.

对于在资源分布列表中没有命中的文件, 在下载完成后需要在资源服务器中进行更新, 在地址内容映射表和内容地址映射表中添加相应的记录, 通过在文件下载过程中不断对资源服务器进行扩展, 实现了对网络资源分布情况的动态更新. 资源服务器的更新以增量方式进行的, 避免了冗余信息造成的服务器负担与资源浪费.

2.2 浏览器端

2.2.1 资源下载模块

本模块的主要算法如图 3 所示.

本文基于 Web Workers 的特性设计了浏览器端多线程下载模块. 主线程负责与用户交互、目标资源分段、多个并发子线程的创建、子线程的监听与管理. 子线程只需要接收主线程的下载请求、下载对应资源片段、将下载完成的资源片段返回给主线程. 在下载过程中多个子线程并发运行, 向多个服务器请求自己需要的资源片段, 实现了浏览器端的多线程下载功能.

在多线程下载阶段, 主线程首先根据资源列表和文件大小确定子线程的数量, 对资源进行分段, 并且设置每一个资源片段的开始偏移和结束偏移. 然后主线程利用 Web Workers 创建子线程, 通过 postMessage 函数将文件片段信息和目标 URL 发送给子线程, 并通过 onMessage 函数监听子线程的消息, 获得子线程下载完成的资源片段.

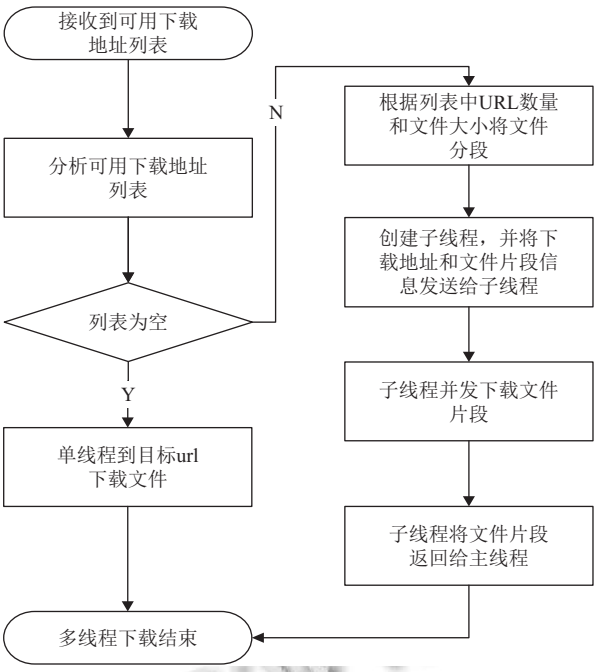

图 3 资源下载模块工作流程图

多线程下载阶段主线程算法流程如下:

```
Begin
  If (urlList is not empty) {
    Split file to chunks;
    Create workers;
    PostMessage (<url, offset>);
    If (onMessage) {
      Receive chunk;
    }
  } else {
  Download file with simple thread;
  }
End
```

子线程被创建完成后, 首先通过 onMessage 函数获得主线程发送的下载信息, 对该信息进行分析处理, 然后根据该信息构造 XMLHttpRequest 对象, 并将该对象发送给服务器.

XMLHttpRequest[12]是 javascript 的内建对象, 使用它可以在浏览器后台实现与服务器交换数据. XMLHttpRequest 对象允许用户通过添加一系列的头信息对当前请求进行限制, 其中 Range 元素使得用户可以设定自己请求的具体数据信息, 包括数据的开始偏移、结束偏移、数据长度等. 服务器通过分析请求对象头信息即可得到用户请求的资源片段信息, 然后将相应的资源片段返回给前端.

由于每一个子线程都是相互独立的, 它们彼此之间通过主线程进行通信, 所以各个子线程之间的下载是并发进行的. 当子线程的文件片段下载完成后, 需要向主线程发送已经下载完成的消息.

多线程下载阶段子线程算法流程如下:

```
Begin
  If (onMessage) {
    Create XMLhttpRequest Object XHR;
    Post request to server;
    If (XHR.status is 200 or 206) {
      Get chunk from response content;
    }
    PostMessage (chunk);
  } else {
    Waiting
  }
End
```

2.2.2 线程控制与异常处理

随着互联网的发展, 分布在互联网中的资源急速增长, 对于同一个目标资源可能存在很多可用的下载服务器, 但是如果直接向所有可用地址都请求资源片段, 不仅会增加线程管理的成本, 而且会增加资源分段与合并的时间开销, 反而会降低下载效率.

为了避免线程过多影响资源整体的下载效率, 在进行正式的资源下载之前, 首先对可用资源地址列表进行判断. 如果可用下载服务器的数量大于阈值, 系统首先会向所有的可用下载服务器请求一个很小的资源片段. 当资源片段下载完成后, 子线程会将该资源片段和数据传输速度数据一同返回给主线程, 该数据表示了资源片段传输时间内的平均数据传输速度. 主线程接收到子线程的数据后, 会根据该数据对可用下载服务器进行排序, 然后根据资源大小择优选择一定数量的服务器请求剩余的资源数据.

由于网络环境的复杂性, 在浏览器端与服务器的数据传输过程中很有可能出现超时、服务断开等问题, 为了保证下载过程的健壮性和高效性, 我们根据当前的网络状态对资源片段进行动态分配. 为了达到这个目的, 本文对资源分段的时候尽量把资源分成小段, 保证每一个资源片段可以在尽快的时间内请求完成, 然

后根据子线程的资源下载速度动态分配线程负责的资源片段.

(1) 在进行资源多源多线程下载之前, 首先对资源进行分段, 并且将资源片段的信息放入到资源片段池中.

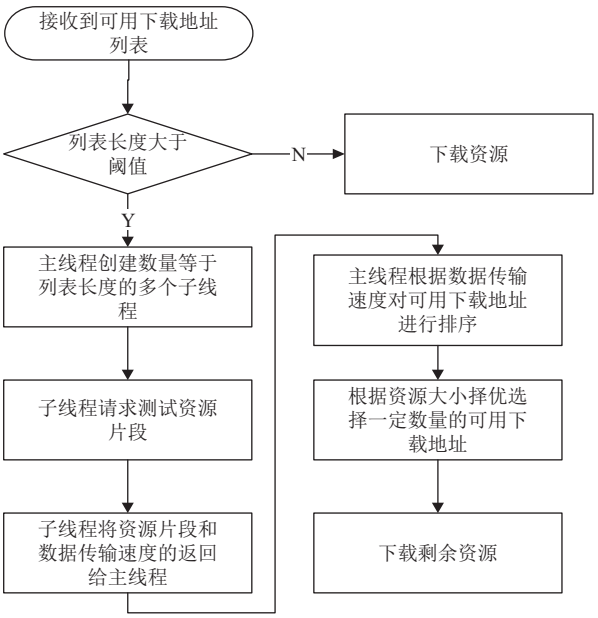

图 4 线程控制流程图

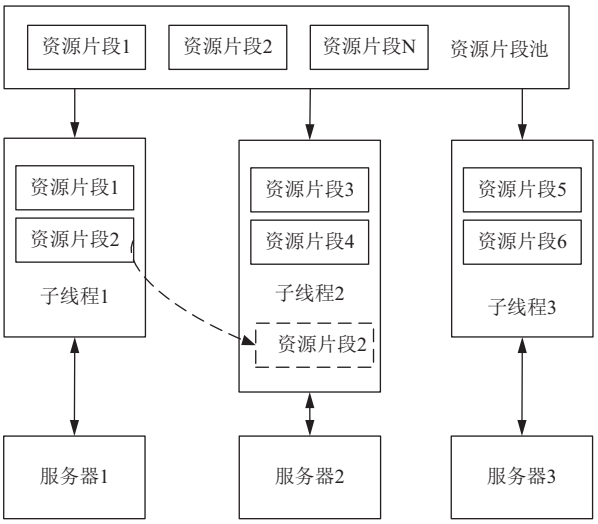

图 5 下载过程异常处理

(2) 在多源多线程下载之后, 每一个子线程首先去资源片段池请求几个资源片段, 在开始请求阶段, 每一个子线程请求的资源片段偏移量是连续的, 可以看做每一个线程请求了一个大的资源片段.

(3) 在子线程下载资源片段的过程中, 需要对资源片段下载情况进行汇报, 例如资源下载速度、网络状况或者目标服务器连接状态等.

(4) 当资源片段下载过程发生异常时, 子线程会将未下载完成的资源片段信息发送给主线程.

(5) 当主线程接收到子线程发送过来的资源片段信息后, 可以根据其他子线程发送过来的资源下载速度选择将该资源片段交给其他子线程, 该子线程向资源传输速度较快的服务器请求数据.

在上述策略中, 当某一个子线程在下载过程出现超时、服务器中断等异常情况, 主线程会将该子线程请求的资源片段转向其他服务器请求, 保证了下载过程的健壮性.

2.2.3 写操作模块

多线程下载模块下载完数据后需要把数据传送给写文件模块, 为了解决多线程写操作冲突问题, 系统在写操作模块采用了 ArrayBuffer 数组的方式.

本模块具体的工作流程如图 6 所示.

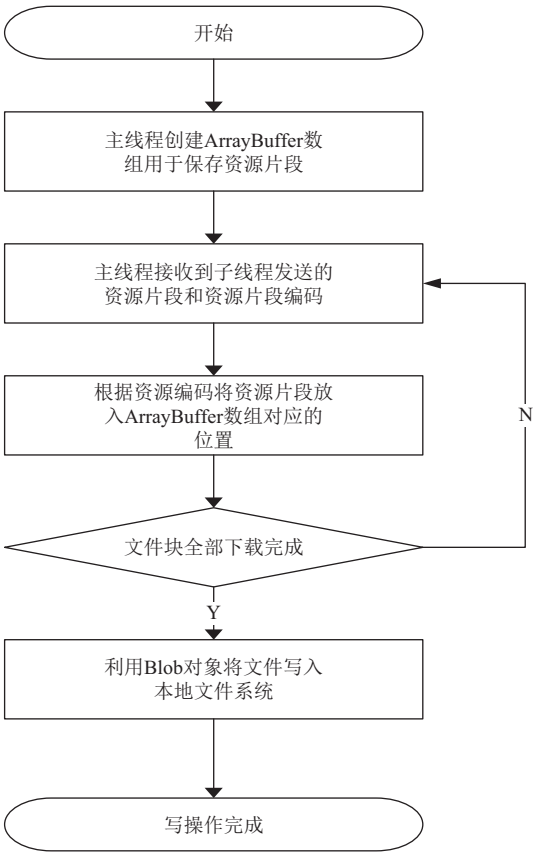

图 6 写操作模块工作流程图

(1) 主线程在对资源进行分段的时候, 对于每一个资源片段需要进行编号, 文件片段的编号顺序和文件

片段的开始偏移顺序一一对应, 例如开始偏移为 0 的资源片段的编号为 0, 它的下一个资源片段编号为 1, 以此类推.

(2) 子线程将资源片段下载完成后, 首先返回给主线程, 并将该资源片段的开始偏移和资源片段编号发送给主线程.

(3) 主线程需要在子线程创建之前保存一个 ArrayBuffer 数组[15]用来存储文件片段, 当主线程接收到子线程发送过来的资源片段后, 根据该资源片段的编号将其放入到 ArrayBuffer 中对应位置.

(4) 当所有的子线程都下载完成后, 主线程通过 Blob 对象将资源写到本地, 完成下载.

3 实验

为了验证系统的高效性, 本文进行了相关实验, 分别测试了系统在不同文件大小、不同网络环境中, 单线程和多线程下载文件所用的运行时间, 并进行了对比分析.

3.1 实验环境

本文的实验环境如图 7 所示. 在实验阶段, 本文使用了三台服务器构建虚拟多服务器下载网络, 包括一台资源索引服务器和两台资源下载服务器, 并将其中一台资源下载服务器作为原始下载站点, 另一台资源服务器作为可用下载站点. 在基本实验环境中, 浏览器端与原始下载站点的网络链路带宽为 100 Mbps, 链路往返延迟 RTT=4 ms; 浏览器端与可用下载站点的网络链路带宽为 100 Mbps, 链路往返延迟 RTT=4 ms.

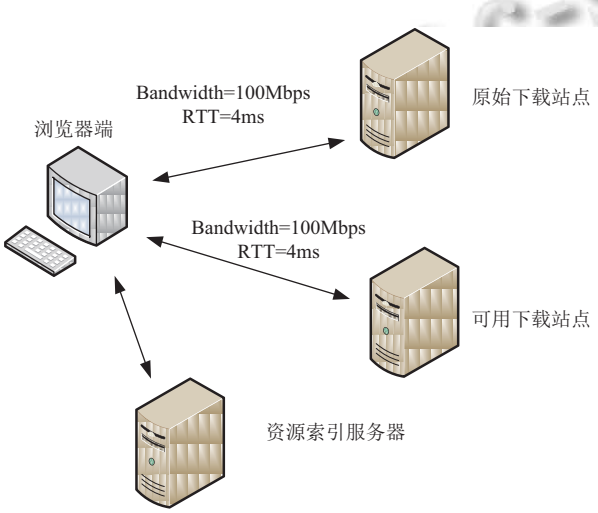

图 7 实验环境拓扑图

为了测试单线程下载和多线程下载对于不同大小文件的影响, 本文实验阶段在两个服务器中都部署了内容相同的 50 个文件, 文件大小从 100 KB 到 200 MB 不等, 文件大小等比递增. 为了验证系统对于不同格式文件的影响, 测试文件格式也采用了多种格式, 包括 Word 文件、文本文档、zip 压缩包、exe 文件等, 最终下载结果表明上述格式的文件均能被正确下载, 所以下载过程的正确性可以保证.

3.2 实验结果

3.2.1 基本实验

图 8 显示了在网络畅通, 不存在大网络延迟和高丢包率的实验环境中, 对于不同大小的文件, 单线程和多线程下载时间对比.

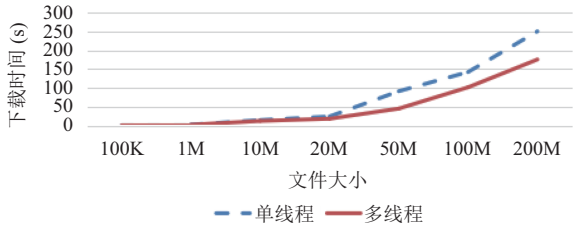

图 8 一般网络状况下单线程与多线程下载效率对比

由图 8 可看出, 在网络正常并且畅通的情况下, 在小于 10 M 的文件下载过程, 多线程和单线程的下载效率相近, 单线程的下载效率略优于多线程的下载效率; 但是在大于 10 M 的文件下载过程中, 多线程的下载效率要明显优于单线程.

3.2.2 高延迟网络环境实验

在本组实验中, 我们使用 clumsy 软件模拟了浏览器端到原始下载服务器存在 100 ms 的高网络延迟.

图 9 显示了在浏览器端到原始下载站点存在 100 ms 的网络延迟环境中, 对于不同文件大小, 单线程和多线程下载平均速率对比.

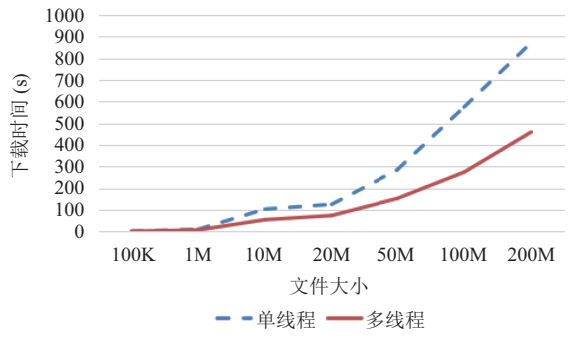

图 9 网络延迟 100 ms 状况下单线程与多线程下载效率对比

由图 9 可看出,在网络存在高延迟的情况下,在小于 1M 的文件下载过程, 多线程和单线程的下载效率相近, 单线程和多线程之间的下载效率没有明显的差异; 但是在大于 1 M 的文件下载过程中, 多线程的下载效率要明显优于单线程, 下载效率接近于单线程的两倍.

3.2.3 高丢包率网络环境实验

在本组实验中, 我们使用 clumsy 软件模拟了浏览器端到原始下载服务器存在 1% 的高丢包率.

图 10 显示了在浏览器端到原始下载站点存在丢包率为 1% 的高丢包率网络环境, 对于不同文件大小, 单线程和多线程下载平均速率对比.

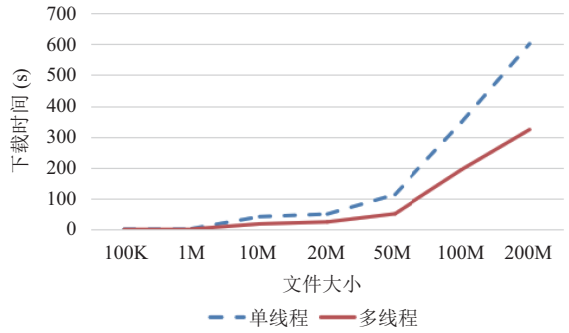

图 10 网络丢包率 1% 状况下单线程与多线程下载效率对比

图 10 显示, 在大于 1 M 的文件下载过程中, 多线程的下载效率要优于单线程, 而且当文件长度大于 10 M 的情况下, 多线程的下载速率要远远优于单线程.

3.2.4 内存开销对比

为了单线程下载和多线程下载的内存开销进行分析, 本文在进行基础实验的过程中, 分别记录了单线程下载和多线程下载的内存占用情况, 具体数据如图 11 所示.

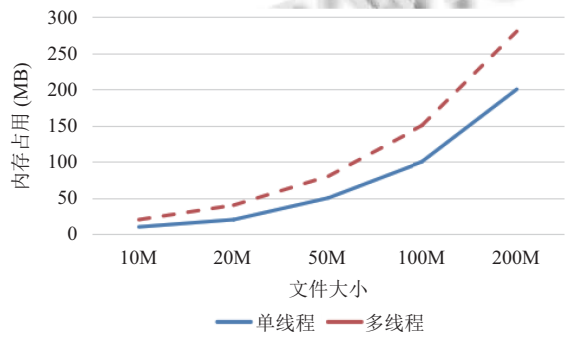

图 11 单线程与多线程内存占用对比

图 11 显示, 在资源下载的过程中, 单线程的内存占用要小于多线程. 这是由于本文在写操作模块采用 Blob 对象加 ArrayBuffer 的方式, 在所有的资源片段都下载完成之前, 需要先将已下载的缓存在内存中, 所以多线程下载增加了内存开销.

3.3 结果分析

通过对实验结果进行分析, 可以发现在大文件下载、大网络延迟以及高丢包率的情况下, 多线程下载平均速率要明显高于单线程下载速率.

在大文件下载过程中, 文件的传输时间要远远高于文件分段以及文件合并的处理时间, 所以当使用多线程下载技术时, 每一个子线程需要下载的文件大小极大缩短了, 子线程的下载时间也极大缩短, 这样使得多线程的下载平均速率要高于单线程.

在大延迟网络环境中, 由于 TCP 协议本身的特性, 一旦发生网络拥塞导致丢包, TCP 拥塞控制窗口需要一个漫长的恢复时间, 在窗口恢复的过程中, 网络带宽不能被充分利用[16], 这样就造成了单线程下载速率低. 而多线程可以在某一线程网络阻塞的情况下, 其他子线程依然可以充分利用带宽高速下载, 保证了下载的高速率.

在高丢包率的环境中, 同样由于 TCP 协议本身的特性, 数据包一旦丢失, 不但拥塞窗口的恢复需要时间, 而且需要对丢失的数据包进行重传, 对于单线程下载而言, 相当于文件长度被增大, 同样会降低下载效率. 而对于多线程而言, 每一个子线程数据包的丢失数量要远远小于单线程, 最终确保多线程下载有较高的效率.

以上多种环境下的实验结果表明, 我们提出的基于 HTML5 的浏览器端多线程下载技术优于单线程下载技术, 显著提升了文件下载效率.

4 结束语

传统浏览器存在单线程下载效率低下、过度依赖目标服务器的问题. 本文提出以资源内容为核心, 将用户的单线程下载行为扩展为多线程下载行为, 实现了浏览器端的多线程下载技术. 在下载过程中, 通过对资源索引服务器进行更新, 动态获取资源在网络中的分布情况. 本系统克服了传统浏览器需要安装客户端或者插件的缺点, 用户只需要支持 HTML5 的浏览器即可实现随时随地的使用多线程下载功能. 在多组环境下的实验证明, 本文提出的基于 HTML5 的浏览器端多线程下载技术明显优于单线程下载技术, 显著提升了文件下载效率.

由于 JavaScript 本身的局限性, 在浏览器端多线程下载的过程中需要首先将资源片段放入 ArrayBuffer 数组中进行合并, 然后才能将文件写入本地, 这样对于小文件而言, 多线程的下载效率并不是特别理想; 对于超大文件而言, 无法将所有的文件片段直接放入数组中; 目前工作中采用的文件分段策略也比较简单, 在下一步工作将对以上问题进行优化.

参考文献

1 俞嘉地. BitTorrent 对等网文件共享系统关键技术研究[博士学位论文]. 上海: 上海交通大学, 2007.

2 李蕊. 基于网络爬虫技术的多源下载系统的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2011.

3 Herhu S, Hudson RL, Shpeisman T, *et al*. Parallel Programming for the Web. HotPar, 2012.

4 Bidelman E. The basics of Web workers. Html5 Rocks Tutorials, 2013.

5 Harjono J, Ng G, Kong D, *et al*. Building smarter web applications with HTML5. Proc. of the 2010 Conference of the Center for Advanced Studies on Collaborative Research. Toronto, Ontario, Canada. 2010. 402–403.

6 Tu JM, Luo XG, Zhao WJ, *et al*. Disk Cache mechanism of tile data based on HTML5 technology. Proc. of the 23rd International Conference on Geoinformatics. Wuhan, China. 2015. 1–4.

7 Mozilla Developer Network. Using Web workers. https://developer.mozilla.org/en-US/docs/Web/API/Web_ Workers_API/Using_web_workers. [2017-01-27].

8 W3C Working Editor's Draft. File API: Writer. https://dev.w3.org/2009/dap/file-system/file-writer.html. [2012-03-07].

9 Bidelman E. Using the HTML5 Filesystem API: A True Filesystem for the Browser. O'Reilly Media, 2011: 25–76.

10 W3C Working Group Note. File API: Directories and system. https://dev.w3.org/2009/dap/file-system/pub/FileSystem/. [2014-04-24].

11 Crowther R, Lennon J, Blue A, 等. HTML5 实战. 张怀勇, 译. 北京: 人民邮电出版社, 2015: 82–96.

12 W3C Web Platform Testsuite. XMLHttpRequest. https://xhr.spec.whatwg.org/. [2017-01-24].

13 Levent-Levi T. How to send a file using WebRTC data API. https://bloggeek.me/send-file-webrtc-data-api/. [2013-08-28].

14 Mozilla Developer Network. Blob. https://developer.mozilla.org/en-US/docs/Web/API/Blob. [2017-01-24].

15 Mozilla Developer Network. ArrayBuffer. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer. [2016-12-27].

16 任勇毛, 秦刚, 唐海娜, 等. 高速长距离光网络传输协议性能分析. 计算机学报, 2008, 31(10): 1679–1686. [doi: 10.3321/j.issn:0254-4164.2008.10.002]