·多媒体技术及应用· 文章编号：1000-3428(2015)01-0256-05 文献标识码：A 中图分类号：TP37

基于状态机的 HTTP Chunked 流并发解析

李明哲[1,2]，陈 君[1]，王劲林[1]，陈 晓[1]

（1. 中国科学院声学研究所国家网络新媒体工程技术研究中心，北京 100190；2. 中国科学院大学，北京 100190）

摘 要： 某些流媒体服务器需要对 HTTP Chunked 编码数据流进行并发解析，朴素静态解析算法难以应用于高效灵活的事件驱动并发模型，且会造成长延迟和多次数据拷贝，导致内存和计算资源开销都较高。针对上述问题，提出一种基于有限状态机的解析策略。将一次接收和一次解析操作构成一个任务片，从而适应事件驱动模型，对收到的数据包进行即时处理和释放，不需要缓存整个 HTTP 报文，减少一次内存拷贝开销。在数据处理过程中，通过有限状态机保存解析状态，能够在任务片退出后恢复之前的解析状态，从而解决事件驱动模型下的字段断裂问题。实验结果表明，相比于静态解析算法，该策略能够明显地降低解析过程的处理时间和占用的内存。

关键词： 流媒体；HTTP Chunked 编码；并发解析；事件驱动模型；有限状态机；内存拷贝

中文引用格式： 李明哲，陈 君，王劲林，等. 基于状态机的 HTTP Chunked 流并发解析[J]. 计算机工程，2015，41(1)：256-260.

英文引用格式： Li Mingzhe，Chen Jun，Wang Jinlin，et al. HTTP Chunked Stream Concurrence Analysis Based on State Machine[J]. Computer Engineering，2015，41(1)：256-260.

HTTP Chunked Stream Concurrence Analysis Based on State Machine

LI Mingzhe[1,2]，CHEN Jun[1]，WANG Jinlin[1]，CHEN Xiao[1]

（1. National Network New Media Engineering Research Center，Institute of Acoustics，Chinese Academy of Sciences，Beijing 100190，China；2. University of Chinese Academy of Sciences，Beijing 100190，China）

【Abstract】 In some streaming media applications，a server needs to parse HTTP Chunked encoding messages concurrently. The naive static parsing algorithm does not fit in the efficient event-driven concurrency paradigm，and incurs long delay and several memory copies，leads to high memory and computing overhead. To tackle this problem，a finite state machine based parsing algorithm is presented. One receiving and its following parsing operations are combined into a task slice so that the event-driven model can be applied. Data packets are parsed immediately without caching the whole HTTP message，and therefore one memory copy is reduced. Parsing status is saved into state machines to that the context can be restored after the task slice is over，which solves the broken-field problem. Test results show that this method can significantly reduce memory and computation overhead compared with static parsing.

【Key words】 streaming media；HTTP Chunked encoding；concurrence analysis；event-driven model；finite state machine；memory copy

DOI： 10.3969/j.issn.1000-3428.2015.01.048

1 概述

HTTP 协议[1]被广泛用于互联网流媒体服务中[2]，包括视频点播（Video On Demand，VOD）[3-4]和实时直播节目[5-6]等。基于 HTTP 的流媒体服务充分利用了已经广泛部署的 Web 服务设施，尤其是利用了内容分发网络（Content Distribution Network，CDN）中的 HTTP 缓存提高了服务的可扩展性。HTTP 协议的消息格式包含消息首部和消息体。首部部分定义了一些属性字段，向报文接收者提供了重要的信息。其中，Content-Length 属性字段表示消息体的长度。然而，对于某些动态生成的数据，消息生成者可能无法立即知悉消息体的总长度，从而不能及时计算 Content-Length 字段值。使用 Chunked 编码 HTTP/1.1 可以回避这一问题。Chunked 编码是协议的一种传输编码方式，将待传输数据切割为

基金项目： 国家“863”计划基金资助项目（2011AA01A102）；中国科学院战略性先导科技专项课题基金资助项目（XDA06010302）。

作者简介： 李明哲（1988－），男，博士研究生，主研方向：流媒体技术，网络处理器应用；陈 君（通讯作者），副研究员、博士；王劲林，研究员、博士生导师；陈 晓，研究员。

收稿日期： 2013-11-01 **修回日期：** 2013-12-22 **E-mail：** chenj@dsp.ac.cn

多个块,报文只需提供当前已知块的长度。

Chunked 编码被用于流媒体传输过程中。文献[5]利用 Chunked 编码有效地缩小了直播节目的播放延时。中国下一代数字互动电视标准[7]也规定,节目流从 CDN 到流服务器的传输过程使用 HTTP Chunked 编码,由后者进行解码。文献[8]根据该标准,在多核网络处理器平台对流服务器进行了实现。

服务器的并发性有 2 种实现模型:事件驱动和多线程[9-11]。相对于多线程模型,事件驱动模型有更高的效率和更大的灵活性,缺点是编程开发过程较为复杂。虽然多线程的效率日益提高,然而对于一个支持数千条流的服务器系统,其增加的总开销依旧不能忽略。流媒体服务一般有实时性要求,应用程序必须对各个数据流持有更大的控制权,这也是缺少灵活性的多线程模型不能满足的。另外,一些嵌入式运行环境并不支持线程,如上述流服务系统所基于 Cavium 公司 OCTEON 处理器的 Simple Executive 操作系统环境[8]。因此,本文采用事件驱动模型来实现服务器的并发性。在这种模型下,实时流媒体并发服务器采用分时方式处理各个数据流,每个流的处理过程被分割成小粒度的任务,应用程序按照实时性要求对各个流的 CPU 时间进行统一调度。

Chunked 编码虽然简单直观,然而流媒体具有数据量大、传输时间长的特点,流服务器将会对大量数据进行 Chunked 解码操作,其效率对于流服务器的整体性能至关重要。对于基于事件驱动模型的高并发的流服务器,使用 CPU 分时的方式处理并发数据流,Chunked 编码的各个字段可能会发生断裂,使得解码过程变得复杂。目前几乎没有文献针对事件驱动模型下高并发媒体流 Chunked 编码的解析算法展开研究。为此,本文将探讨在这种场景下如何对 Chunked 编码的并发数据流进行高效解析,提出一种基于有限状态机[12]的解析算法。

2　Chunked 编码的静态解析

Chunked 编码后的消息体一组连续的编码块(Chunk),每个编码块包含了块长度字段(chunk size)及块载荷字段(chunk data)两部分,各部分用回车换行符(CRLF,由 CR 和 LF 2 个字符组成)隔开。文献[7]未涉及 Chunked 编码的扩展字段字段,本文探讨的解码算法将对其进行忽略。为方便讨论,认为块长度字段还包括其前后两处的 CRLF 字符,而块载荷尾部的 CRLF 属于下一个块。这样,一个块完全由块长度和块载荷字段构成。注意第一个块也会有前导的 CRLF,即 HTTP 头部和编码块报文的交界处。

作为对 Chunked 编码的一种最简单的解码算法,静态解析方法先将整个 HTTP 报文获取到本地缓冲区,再在本地依次读取各个编码字段,将解析出的有效数据复制到另一缓冲区,其流程可参考文献[1]。

静态解析算法的缺点包括:

(1)静态解析算法需要先接收整个 HTTP 报文,会导致较大的启动延迟。

(2)静态解析法难以直接应用在事件驱动模型中。一条流的处理过程分为网络接收和解码 2 个环节。其中,网络接收环节可以基于 Select、Epoll[10]等异步消息通知的方式进行任务分割,而解码过程则难以分割。如果整个解析过程当作一个原子性任务被完全处理,则会大大损害其他流的实时性和平滑性。

(3)静态解析法导致解码过程中会产生一次内存拷贝,再加上 HTTP 接收过程、流化数据发送过程中数据在协议栈缓冲区和应用缓冲区之间的拷贝,总共是 3 次拷贝,造成运行效率低下。

3　基于 FSM 的解析算法

3.1　事件驱动模型下的 Chunked 解码

考虑将网络接收环节和解码环节交替进行,一次接收操作和一次解码操作构成一个任务片,从而适应了事件驱动模型。在一个任务片中,立即对本次接收操作得到的网络数据进行解析,提取出有效内容,这样就可以及时将接收到的数据进行丢弃,而不必拼接成完整的 HTTP 报文,从而减少了一次内存拷贝。

任务的分割是由单次接收操作的数据量来决定。应用程序以非阻塞的方式调用这些网络数据接收接口,如 POSIX.1 定义的 read 和 recv 函数,可以控制单次接收量的上限,应用程序就控制了任务分片长度的上限。对于非阻塞读操作,可能未能读到任何数据,导致该任务片为无用的空片。对异步消息通知机制的利用,可以使应用程序仅在明知该数据流有新数据到达时,才切换到该流进行处理,一般至少能读取到一个传输层报文。这样,任务分片长度的下限也可以得到控制,避免了空片。

读取操作会依次读到每个块的块长度和块载荷字段。当读完块长度字段后,即可确定块载荷字段的长度,进而将接下来的块载荷数据直接读入有效内容缓冲区。解码过程中块长度字段只能被读入临时缓冲区。用 sread 和 dread 分别表示读取块长度和块载荷的操作,分别将协议栈内核缓冲区的数据读入应用的临时缓冲区和有效内容缓冲区,统称 read 操作。$m = read(n)$ 表示应用程序以非阻塞方式要求读取 n Byte 数据,实际成功读取了 m Byte。

对于每个流,其块长度和块载荷字段都可能因任务分片而不能完整读取,造成字段截断。因此,每个数据流在任务分片结束时保存其当前解析状态,在下一个任务分片读取保持的状态信息,以恢复之前的工作。

3.2 基于有限状态机的 Chunked 解码算法

有限状态机是一个数学工具,用于处理有限数量子程序(状态)的发展变化。每个时刻根据当前的状态和本时刻的输入,可以决定应执行的动作,包括确定下一个状态以进行转移。利用 FSM 设计了一个 Chunked 编码解析方法,将解析过程分为多种状态。每个状态都会进行 sread 或 dread 操作。在 read 操作返回之后,分块读取的完成程度决定了当前的状态。状态的定义状态的名称如表 1 所示。

表 1 FSM 状态定义

状态名称	解释
cr1	下一个字节是块长度的第 1 个 CR
lf1	下一个字节是块长度的第 1 个 LF
cs	下一个字节是块长度的十六进制字符
cr2	下一个字节可能是块长度的第 2 个 CR
lf2	下一个字节是块长度的第 2 个 LF
cd	下一个字节属于块载荷

算法启动时处于 cr1 状态,表示下一个字节是本块的第一个 CR 字符。通过 read 操作读入新的数据,算法会在各状态间转换,如图 1 所示。其中,符号 S 表示块长度字段中的一个十六进制字符;D 表示块载荷中的一个字符;星号 * 表示前面那类字符被读入一次或多次;括号中的符号指示下一个待读入字符的类型。

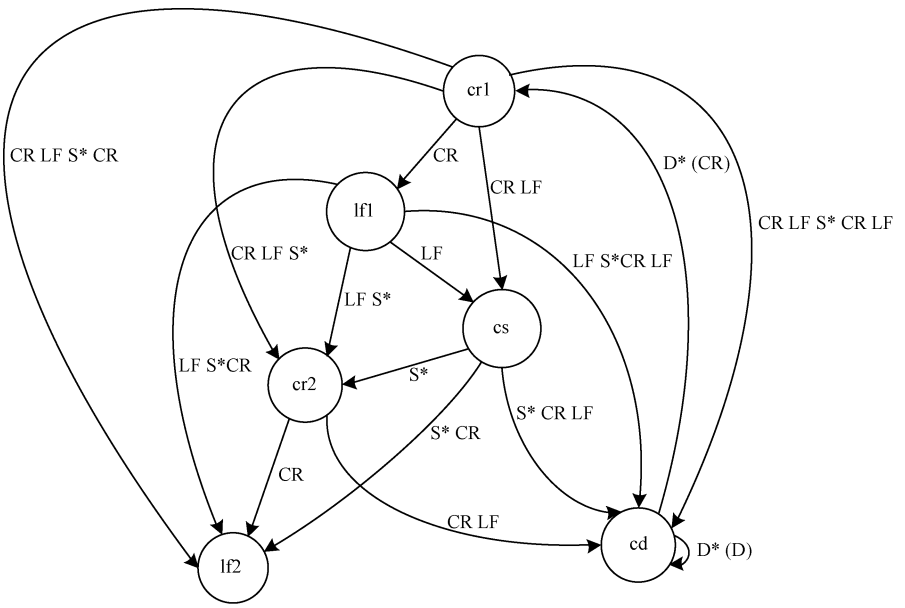

图 1 FSM 状态转移

在 cr1 状态下,通过 sread 操作读入 CR 后,进入 lf1 状态。在读入 LF 后,进入 cs 状态,期待即将读入块长度中的十六进制字符,块长度字段可能只有一个这种字符。在读入第一个十六进制字符后,不确定下一个字符是十六进制字符,还是 CR,记此时进入 cr2 状态。记块长度字段对应的整数值为 chunklen,解析前猜测为 0,在 cr2 状态,sread 操作会不断读入新的十六进制字符,用于更新 chunklen 的猜测值。具体做法是:将新字符转换为整型数 d,则 chunklen 更新为 chunklen × 16 + d,其中,“ × ”操作可以用移位操作实现。直到读入的字符是 CR,进入 lf2 状态,此时,chunklen 的值可以确定,反映了块载荷字段的长度,算法进入 cd 状态,表示下一个待读取的字节属于块载荷,此时,开始使用 dread 操作读取有效内容。在 cd 状态下,随着 dread 操作,chunklen 表示尚未被读取的块载荷字节数,不断减小。直到 chunklen 最后变为 0,此时,本编码块所有的块载荷数据均已被读取,算法进入 cr1 状态。

上述算法在 sread 操作过程中每次只读一个字节,可以加以改进,以减小读操作次数。原则是 sread 操作应尽可能多地读取数据,但避免读取到块载荷数据,因此,sread 之前应猜测块长度字段还剩余多少字节未读,取最小的可能值,作为 sread 参数。,如在 cr1 状态,块长度字段的未读字节数至少为 5。而在 lf1 状态,则为 4。根据 sread 成功读到的字节数,确定下一个转移状态。另外,在 cd 状态下,dread 的参数未必取 chunklen,可以用每个任务分片的最大读取字节数 maxSliceLen 进行限制。这样,每

个状态下应执行的动作如表 2 所示。

表 2　状态的动作

状态名称	操作
cr1	sread(5)
lf1	sread(4)
cs	sread(3)
cr2	更新 chunklen,sread(1)
lf2	确定块载荷长度,如为 0 则退出;sread(1)
cd	rlen = dread(min(chunklen,maxSliceLen)); chunklen = rlen

基于 FSM 的算法能够很好地适用于事件驱动模型,同时相对于静态解析可以减少一次内存拷贝。

3.3　算法执行实例

以“\r\n10\r\n0123456789abcdef”编码块为例说明上述算法的执行过程,实际的流媒体服务的编码块长要远远大于这个例子。根据表 1 的定义,它可能会经历的状态如表 3 所示。

表 3　算法执行实例

已接收报文	状态名称
无	cr1
\r	lf1
\r\n	cs
\r\n1	cr2
\r\n10	cr2
\r\n10\r	lf2
\r\n10\r\n	cd
\r\n10\r\n0	cd
\r\n10\r\n0…e	cd
\r\n10\r\n0…ef	cr1

如果流在接收到前 3 个字符“\r\n1”后遇到截断,则无法确定块长度数值。此时,需要保存的信息包括处于状态 cr2,chunklen 值为 1。当这条数据流进入第 2 个任务分片时,首先查看已保存的信息,得知目前为 cr2 状态,按照表 2 的说明,执行 sread(2),得到字符“0\r”,于是 lf2 状态,确定 chunklen 为 16。再执行 sread(1),进入 cd 状态。如果分片要求读入的数据的长度不能超过 10 Byte(maxSliceLen),于是执行 dread(10),成功读取了 10 Byte,chunklen 更新为 6,此时,任务分片结束。在第 3 个任务分片,根据当前的 cd 状态和 chunklen 与 maxSliceLen 的值,执行 dread(6)。如果成功读取了 6 个字节则进入 cr1 状态,准备解析下一个分块。否则,FSM 以 cd 状态退出本分片。

4　测试评估

对上述静态解析算法和基于 FSM 的解析算法进行测试,以评估两者的处理时延和内存消耗。测试主机具有 2 颗 Intel Pentium(R) Dual-Core E5700 CPU,2 GB 内存,运行 Linux 操作系统,内核版本号为 3.8.0-32。HTTP 报文发送端和接收端都基于 Python2.7.4 实现,作为在同一主机上运行的 2 个进程,使用 socket 进行通信。发送端以 1 KB 长度为编码块单位,发送不同长度的编码报文。而接收端分别运行静态解析算法和基于 FSM 的解析算法,对报文进行提取,提取出的有效内容随即丢弃,以对应流服务器的发送操作。而静态解析算法需接收的整个 HTTP 报文则存放于主存中。2 个算法均不涉及磁盘 I/O 操作。将算法的处理时延定义为整个报文的接收和解析的总的进程时间,通过 Python 的 timeit 模块进行统计。内存消耗通过单独开启进程运行 ps 命令进行统计,ps 命令通过读取/proc 目录下的文件获得各个进程的相关信息。Python 的 command 模块对进程间通信提供了一种实现。它将接收端程序的进程传递给 ps 进程,又将 ps 运行的结果传递给接收端进程。接收端进程在解析算法执行开始前和结束后分别开启一个 ps 进程获得自身的相关数据,通过两者的比较获得解析过程中的内存消耗。

图 2 表明,基于 FSM 的解析算法明显缩短了处理时间。原因是其处理过程在最后一个数据包接收后很快就能结束。而静态解析算法则在整个报文接收完成后还要回到报文的开头,对整个报文再进行一遍扫描,并复制有效内容。静态解析算法执行过程中 HTTP 报文临时存放于主存中,可以预期,如果报文存放于磁盘中,则静态解析算法的处理时间会进一步增加,进而更加显示出基于 FSM 的解析算法的优势。

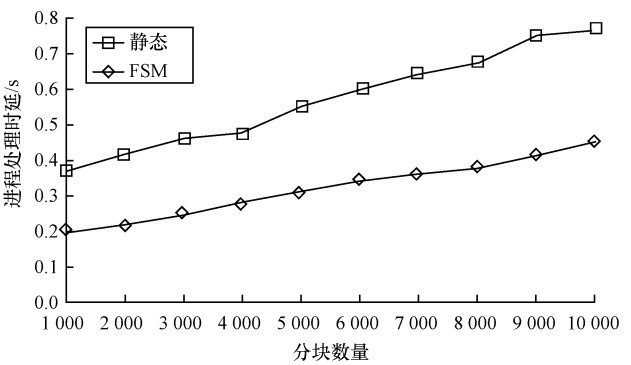

图 2　进程处理时延对比

如图 3 所示,基于 FSM 的解析算法大大降低了内存损耗,由于不需要拼接整个 HTTP 报文,程序的内存占用主要包括当前任务分片所对应的应用层缓冲区,并不随报文长度的增加而有明显增长。

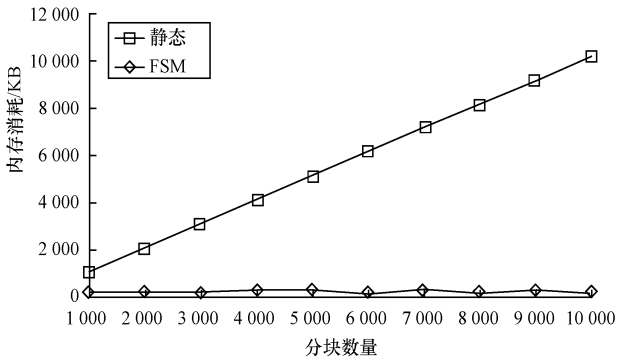

图 3 内存消耗对比

5 结束语

本文针对高并发流媒体应用中 HTTP Chunked 编码的静态解析算法导致的长延时、高开销等问题，提出一种基于有限状态机的并发解析策略。在单一执行环境下交替处理每一条数据流，利用状态机恢复各数据流的执行现场。对于每个驱动事件对应的数据，接收后立即解析，从而消除了静态策略中报文拼凑过程导致的内存拷贝开销。通过基于 Linux 主机的实验测试，验证了本文方案能够明显缩短处理时延，并大大降低内存消耗。今后将针对流媒体应用中的其他传输协议研究相关优化策略。

参考文献

[1] Fielding R, Gettys J, Mogul J, et al. Hypertext Transfer Protocol——HTTP/1.1[S]. RFC 2616, 1999.

[2] Plissonneau L, Biersack E. A Longitudinal View of HTTP Video Streaming Performance[C]//Proceedings of the 3rd Multimedia Systems Conference. New York, USA: ACM Press, 2012: 203-214.

[3] Begen A C, Akgul T, Baugher M. Watching Video over the Web: Part 1: Streaming Protocols[J]. IEEE Internet Computing, 2011, 15(2): 54-63.

[4] Akhshabi S, Begen A C, Dovrolis C. An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over Http[C]//Proceedings of the 2nd Annual ACM Conference on Multimedia Systems. [S.l.]: ACM Press, 2011: 157-168.

[5] Swaminathan V, Sheng Wei. Low Latency Live Video Streaming using HTTP Chunked Encoding [C]// Proceedings of MMSP'11. Hangzhou, China: [s.n.], 2011: 1-6.

[6] Roverso R, El-Ansary S, Högqvist M. On HTTP Live Streaming in Large Enterprises[C]//Proceedings of SIGCOMM'13. Hong Kong, China: [s.n.], 2013: 489-490.

[7] 全国广播电影电视标准化技术委员会. GY/T258-2012 下一代广播电视网(NGB)视频点播系统技术规范[S]. 2012.

[8] Li Jun, Chen Jun, Li Mingzhe, et al. A Multi-core Architecture for Video Streaming[EB/OL]. (2010-11-21). http://www.scientific.net/AMM.411-414.960.

[9] Erbad A, Hutchinson N C, Krasic C. DOHA: Scalable Real-time Web Applications Through Adaptive Concurrent Execution[C]//Proceedings of the 21st International Conference on World Wide Web. Lyon, France: ACM Press, 2012: 161-170.

[10] Wang Xin. Technical Analysis of High-capacity and Concurrency Server Groups[C]//Proceedings of ICEEE'10. Zhengzhou, China: [s.n.], 2010: 1-4.

[11] Kimpe D, Carns P, Harms K, et al. Aesop: Expressing Concurrency in High-performance System Software[C]// Proceedings of NAS'12. [S.l.]: IEEE Press, 2012: 303-312.

[12] Anderson J A, Head T J. Automata Theory with Modern Applications[M]. New York, USA: Cambridge University Press, 2006.

编辑 刘 冰

(上接第 255 页)

[5] 3GPP. TS 23.206-2007 Voice Call Continuity (VCC) Between Circuit Switched (CS) and IP Multimedia Subsystem (IMS); Stage 2 (Release 7)[S]. 2007.

[6] 徐德平，耿鲁静. 浅析 LTE 系统 CSFB 话音解决方案[J]. 电信工程技术与标准化，2013，(1)：55-59.

[7] 3GPP. TS 23.216-2012 Single Radio Voice Call Continuity (SRVCC); Stage 2 (Release 11)[S]. 2012.

[8] 3GPP. TS 23.237-2009 IP Multimedia Subsystem (IMS) Service Continuity; Stage 2 (Release 10)[S]. 2009.

[9] 刘 晨. SRVCC 优化方案研究及改进[D]. 北京：北京邮电大学，2010.

[10] Koshimizu T, Tanaka I. Improvement on the VoLTE (Voice Over LTE) Domain Handover with Operator's Vision[C]//Proceedings of World Telecommunications Congress. Miyazaki, Japan: [s.n.], 2012: 1-5.

[11] 3GPP. TR 25.913-2010 Requirements for E-UTRA and E-UTRAN (Release 9)[S]. 2010.

[12] Namakoye J, Olst R V. Performance Evaluation of a Voice Call Handover Scheme between LTE and UMTS[C]// Proceedings of IEEE AFRICON'11. Livingstone, Zambia: IEEE Press, 2011: 1-4.

[13] 许慕鸿. SRVCC 增强技术分析[J]. 电信网技术，2010，(12)：10-16.

[14] 3GPP. TR 23.856-2010 Single Radio Voice Call Continuity (SRVCC) Enhancements; Stage 2 (Release 10)[S]. 2010.

[15] 刘 静. LTE 系统中的切换与切换自优化方法研究[D]. 西安：西安电子科技大学，2009.

编辑 刘 冰