面向国内直播行业的分布式弹幕爬虫研究

王雪瑞 刘 渊
(江南大学数字媒体学院 江苏 无锡 214122)

摘 要 近年来,依托视频行业与直播行业的兴盛,弹幕服务迅速发展。然而主流弹幕服务的弹幕环境一直以来缺乏监管,主播与用户违规行为屡禁不止,对直播弹幕的学术研究稀少,亟需开发针对弹幕的采集处理方案。针对国内知名弹幕服务的技术特征,设计一种分布式直播弹幕爬虫系统方案。分析并提出相应房间连接的建立机制与弹幕采集机制:对开放 API 的服务直接采用轻量级客户端实现;对基于 Adobe Flash 且不开放 API 的服务,用基于 Chromium 浏览器的 Electron 模拟浏览直播间网页,并改写其 PPAPI 插件界面实现,旁路 Flash 网络流量从而实现抓取。在某知名弹幕平台上进行了验证性实验,表明该系统能够调度 IP 地址资源进行较大规模抓取,且性能较好,能够处理平均 134 条每秒、峰值超过 1 000 条每秒的弹幕流量。

关键词 直播弹幕 爬虫 浏览器模拟 PPAPI 旁路

中图分类号 TP393.09 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.02.025

RESEARCH ON DISTRIBUTED DANMAKU CRAWLING FOR CHINESE LIVE-STREAMING SERVICES

Wang Xuerui Liu Yuan
(*School of Digital Media, Jiangnan University, Wuxi 214122, Jiangsu, China*)

Abstract Danmaku services have grown in popularity recently thanks to the blooming of online video and live-streaming industries. However, the environment is seriously under-regulated, with frequent rule violations from both casters and audience, resulting in negative publicity of the industry, hence a need for research on danmaku crawling and processing is realized. A distributed live-streaming danmaku crawler scheme was devised according to technical characteristics of popular Chinese danmaku services. Light-weight clients were developed for services implementing open standards. For services based on Adobe Flash and proprietary interfaces, Chromium browser-based Electron with modified PPAPI plugin interface implementation was used to simulate browsing of streaming room pages, side-channeling Flash network traffic for crawling. Experiments on a well-known danmaku service showed that the system was able to dispatch IP addresses for large-scale crawling and the performance was good with achieving an average speed of 134 per second and peak speed of more than 1 000 per second for the danmaku traffic.

Keywords Live-streaming danmaku Crawler Browser simulation PPAPI side-channeling

0 引 言

近年来国内直播产业迅速发展,涌现出一大批直播服务,普遍提供弹幕功能,即实时覆盖视频内容之上呈现的文字评论。弹幕作为网络语言与社交的载体之一,无论从语言学或监管角度看,均有潜在价值可供挖掘。尤其值得注意的是,围绕直播平台与网络主播群体的负面新闻不断,很多直播间的语言风格也较为负面,这表明弹幕环境同网络社交平台舆论环境一样需要适当监管。目前国内针对直播弹幕的学术研究尚属空白,无论是弹幕语料库,还是面向弹幕的自然语言处

收稿日期:2017-03-22。国家科技支撑计划课题(2015BAH54F00);国家重点研发计划项目(2016YFB0800305)。**王雪瑞**,硕士生,主研领域:数字媒体技术,自然语言处理。**刘渊**,教授。

理技术都急需建立与研究。

在国外,围绕西方世界最大的电子竞技类直播网站 Twitch. tv 及其弹幕社区已有少量相关研究,例如 Claypool 等[1]与 Nascimento 等[2]通过视频流、弹幕流与主播开关播时间等维度的分析研究直播视频与直播社区。但 Twitch. tv 采用成熟而开放的 IRC 协议传输实时评论,并提供开发者 API 使 IRC 机器人方便接入[3],没有专门为之开发爬虫的必要。因此,国外也就没有开展弹幕爬虫方面的研究。

国内的弹幕服务现状,与国外恰好相反。出于历史与商业原因等国情,国内流行的弹幕服务数量较多,与西方 Twitch. tv 一家独大的状况不同。这些服务无一采用现行开放标准构建,并且只有少数开放相应 API,这使得弹幕信息难以轻易获取。国内的弹幕研究遂集中于视频点播平台的弹幕服务,而没有能够覆盖直播弹幕,这表明直播弹幕爬虫的研究有其必要性。

对不开放 API 的弹幕服务,我们需要模拟用户访问服务的过程来获得弹幕数据。考虑到我们感兴趣的弹幕服务均有 Web 客户端,可以使用浏览器模拟的方式解决。浏览器模拟是在动态 Web 爬虫领域已有广泛应用的成熟技术。早期研究中,彭轲等提出并测试了基于浏览器服务的爬虫[4],刘兵基于 JavaScript 模拟执行进行链接分析[5];此后,张媚[6]、管翠花[7]、段子飞[8]、钱程等[9]均在 Ajax 爬虫方向进行了研究,魏少鹏等直接使用 Chrome 浏览器扩展方式实现爬虫[10],郭津丞等则仅仅改动 WebKit 渲染引擎实现以辅助抓取[11]。然而,不开放 API 的弹幕服务其客户端均以 Adobe Flash 技术实现,页面上 JavaScript 仅仅作为 Flash 播放器插件的辅助而存在。因此上述浏览器模拟技术有的不能直接应用,有的因为只能在 JavaScript 层面工作而效率低下,不能实现有效抓取。

虽然此前有针对 Flash 内容爬虫的相关研究[12],但此 Flash 组件并非内容载体,仅仅与弹幕服务器连接并动态解析、渲染弹幕,因此常见的 Flash 爬虫技术如文献[12]等也不能应用,需要找到更为底层的机制以绕过 Flash 黑箱的限制。

本文即针对国内市场份额较大的弹幕服务,介绍了一种分布式弹幕爬虫方案,具备支持多种开放与否弹幕协议的能力,以及可横向伸缩的承载规模。

1　分布式弹幕爬虫

1.1　原　理

弹幕服务虽然具有实时通信、传输弹幕信息的共性,但其应用层协议各不相同,有些甚至还引入认证机制与防爬虫机制。考虑到这些服务为了吸引用户,无一例外地允许匿名用户观看所有弹幕,因此这些机制的引入原因只可能是出于商业考虑,而不可能出于技术原因,并且所有此类机制必然放行匿名用户。正如任何数字限制管理(Digital Restriction Management)方案都不可避免地存在所谓“模拟漏洞”(analog hole,任何非交互性的受保护数字内容要为人类用户所感知,必须最终被转化为模拟形式,而这一最终模拟形式不可被任何机制限制)一般。这意味着我们完全可以通过模拟观看直播的匿名用户,实现针对任何不开放 API 弹幕服务的爬虫。额外地,由于当前主流弹幕服务的使用协议均于用户注册时生效[13-16],而无论是对不开放 API 进行逆向工程还是实际抓取均不需要注册用户帐号,因此此种机制不存在法律问题。

除规避防爬虫机制的考虑之外,我们不需要特意关心高并发连接对源服务可能造成的影响。因为上述工作原理意味着即使在爬取全站的极端情况下,爬虫在源服务的“足迹”也仅相当于每个房间多出一名匿名用户。这对任何已经能够承载正常数量用户的服务而言都可以忽略不计。

同样地,在不触发目标服务防爬虫机制的前提下,我们也不需要刻意追求请求来源 IP 的分散化,虽然这样做自有好处。这是因为从弹幕服务器的角度来看,它无法区分来自同一个 IP 的并发连接到底是多名穿过 NAT 的正常用户,还是理想情况下在任何方面行为都与正常用户相同的爬虫连接。何况部署了反爬虫或反 DDoS 机制的正常 HTTP 服务也不能做到上述区分,以至于真正攻击者的存在往往会使同一网段或内网的其他用户无故被限制访问。

退一步讲,即使弹幕服务器确实设置了来源 IP 的并发限制,我们总是可以通过引入负载均衡,给爬虫系统分配更多 IP 地址资源的方式来解决。

1.2　系统架构

系统整体架构如图 1 所示,图中黑色矩形节点代

表不同程序模块或节点;黑色边代表数据流。

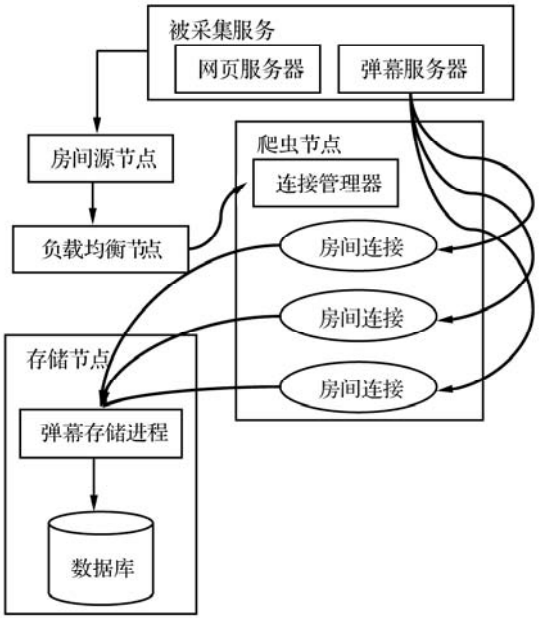

图 1 分布式弹幕爬虫系统架构

图 1 中展示了 1 个支持的弹幕服务、1 个房间源节点、1 个爬虫负载均衡节点、1 个爬虫节点以及 1 个存储节点;实际上除爬虫负载均衡节点之外,其余节点均支持任意数量的扩展。弹幕爬虫工作分为两个阶段:首先采用常规 Web 爬虫技术请求所有指定弹幕服务的网页端,得到可用房间列表;接着根据房间信息启动相应协议的客户端实现,以匿名身份连接房间,将接收到的弹幕发送至弹幕存储组件,异步存储入库。组件之间靠消息队列互相通信。

1.3 负载均衡机制

考虑使爬虫节点的负载尽可能一致,房间连接的负载均衡直接采用 round-robin 方式。规定每个爬虫节点均使用 1 个 IP 地址,针对不同弹幕服务对单 IP 最大连接数的限制,记录所有节点每个服务已建立的连接数量。当某弹幕服务的新房间连接请求到来时,将该请求分配到当前时刻承载相应服务连接数最少的节点。节点与弹幕服务不一一对应,所有节点均支持建立到所有弹幕服务的连接,以保证系统易于横向伸缩。具体算法如算法 1、算法 2 所示。

算法 1 节点连接管理器

1. NodeID = 随机字符串。

2. 向消息队列订阅 NodeID 频道。

3. 启动心跳线程,周期性刷新内存数据库中 NodeID 的健康状态。

4. 进入事件循环:

a) 退出消息:退出循环。

b) 连接请求(Site, Room):创建房间连接控制线程 ControlThread(Site, Room)并启动。

5. 退出心跳线程。

6. 删除内存数据库中 NodeID 状态。

算法 2 房间连接请求分发(Site, Room)

1. 向内存数据库查询健康节点列表。

2. 对每个健康节点:

a) 向内存数据库查询其已经建立的服务 Site 的房间连接数量;

b) 若达到或超过服务 Site 的每节点最大连接数,忽略该节点;

c) 记录并更新连接数量最小的节点 NodeID。

3. 通过消息队列,向节点 NodeID 发送连接请求(Site, Room)。

1.4 房间连接建立机制

房间连接的建立,针对开放 API 的弹幕服务,直接采用各自开放 API 所述的协议连接即可,如算法 3、算法 4 所示,使用算法 5 转交接收的弹幕报文到存储节点。而对不开放 API 的弹幕服务,连接方法于 2.3 节介绍。

算法 3 ControlThread(Site, Room)

1. 判断服务 Site 的接口种类:

a) 开放 API:创建房间连接工作线程 WorkerThread(Site, Room)并启动;

b) 不开放 API:创建浏览器模拟控制线程 ShimThread(Site, Room)并启动。

2. 向消息队列订阅(Site, Room)频道。

3. 进入事件循环,阻塞,直到接收到退出消息。

4. 设置 WorkerThread(Site, Room)线程的退出 flag 为真。

5. 等待 WorkerThread(Site, Room)退出。

算法 4 WorkerThread(Site, Room)

1. 设置退出 flag 为假。

2. 初始化数据缓冲区 Buffer,一个循环缓冲区 RingBuffer。

3. 无限循环:

a) IP, Port = 解析服务器地址(Site);

b) Socket = connect(IP, Port);失败则 continue;

c) 使内存数据库中本节点已建立连接数原子自增 1;

d) 在 Socket 上执行服务 Site 的认证序列;

e) 在 Socket 上执行服务 Site 的进入房间序列;

f) 无限循环,Length = recv(Socket, Buffer, sizeof(Buffer)):

i. 如退出 flag 为真,跳出最外层循环;

ii. 执行 ProcessIncomingDanmaku(Site, Buffer, Length, RingBuffer)。

4. 如循环中任何 Socket 操作失败,或退出循环:

a) close(Socket);

b) 使内存数据库中本节点已建立连接数原子自减 1;

c) Delay = random(最长重试间隔时间);

d) 等待 Delay 的时间;

e) continue(退出循环则 return)。

算法 5　ProcessIncomingDanmaku(Site, Buffer, Length, RingBuffer)

1. Time = 当前时间;

2. 将 Buffer 中前 Length 个字节放进循环缓冲区 RingBuffer;

3. 循环从 RingBuffer 中读服务 Site 的完整一帧报文,存放于 Data,直到没有完整报文:将(Time, Site, Data)通过消息队列传入存储节点。

1.5　异步弹幕分类存储机制

由于接收到的弹幕数量极为庞大,有必要对弹幕进行异步存储,以防止数据库写入性能成为瓶颈。考虑到上游弹幕协议随时可能发生变化,为了最大限度保存原始报文信息,我们直接存储原始弹幕报文入库,留待下游处理组件详细解析。

同时由于一些非聊天信息的实时事件,如用户进入直播间消息、赠送礼物广播消息等通常也使用弹幕方式进行传输,为降低下游处理组件的复杂度及处理负担,以及减少数据库单个表的写入压力,我们依然在存储前按照相应的弹幕协议反序列化每个报文,并按照弹幕类型分为聊天弹幕与非聊天弹幕两类,分别批量插入两个表中。具体算法如算法 6、算法 7 所示。

算法 6　存储节点

1. 建立队列 ChatQueue 与 ControlQueue。

2. 向消息队列订阅原始报文频道。

3. 创建 StorageThread(ChatQueue, ControlQueue)线程并启动。

4. 进入事件循环,接收消息:

a) 弹幕消息(Time, Site, Data):

i. 用服务 Site 的协议反序列化 Data;

ii. 取出 Data 的弹幕类型;

iii. 若 Data 为聊天弹幕,将(Time, Site, Data)放入 ChatQueue,否则放入 ControlQueue。

b) 退出消息:

i. 设置 StorageThread 线程的退出 flag 为真;

ii. 等待 StorageThread 线程退出;

iii. return。

算法 7　StorageThread(ChatQueue, ControlQueue)

1. 设置退出 flag 为假。

2. 无限循环,直到退出 flag 为真:

a) 等待一固定时间;

b) 从 ChatQueue 中原子地取出所有项,批量插入聊天弹幕表;

c) 从 ControlQueue 中原子地取出所有项,批量插入非聊天弹幕表。

1.6　基于 PPAPI 旁路的浏览器模拟

考虑到非开放 API 的弹幕客户端以 Flash 技术实现,Flash 又以插件的形式与浏览器连接,我们即可在浏览器的插件界面附近引入旁路,直接从 Flash 插件与浏览器的交互过程中抓取弹幕信息。目前主流的浏览器插件机制有两种,一种为上世纪 Netscape 浏览器使用的 NPAPI,另一种为 Google 主导的相对更完善的 PPAPI[17]。而 NPAPI 由于年代久远且安全性欠佳,其支持逐渐稀少,故我们选用 PPAPI 接口实现信息旁路,即为 PPAPI 旁路。如图 2 所示。

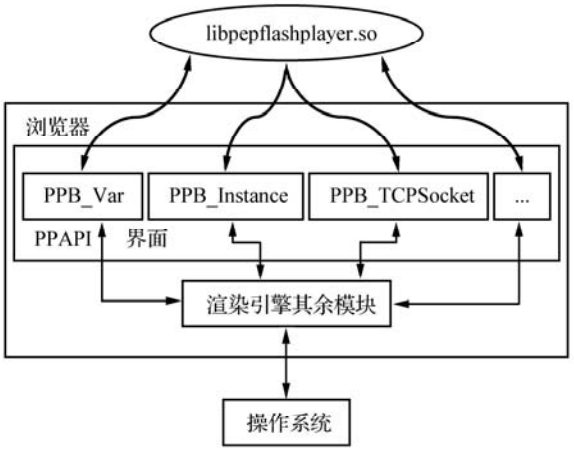

图 2　PPAPI Flash 插件与宿主交互示意

图 2 中,在 PPAPI 插件架构中,浏览器渲染引擎负责向插件(此处为 PPAPI Flash,存在于共享库文件 libpepflashplayer.so 中)暴露其可能需要的一切系统功能界面,如 JavaScript 对象互操作(PPB_Var 接口)、TCP 套接字(PPB_TCPSocket 接口)以及浏览器界面集成(PPB_Instance 接口)等。这主要是为了使不受信任的插件代码能在严格受限的沙盒进程中正常工作,同时也为信息旁路的实现提供了极大便利。我们在 PPB_TCPSocket 接口的实现中可以便捷地截获 Flash 插件发起的所有 TCP 网络活动。由于只有 Flash 插件的网络活动才会流经该接口,其中无关数据包比例明显较链路层抓包时为小,有利于提高处理效率。实际实现的 PPAPI 旁路机制如图 3 所示(图中省略了无关部分)。

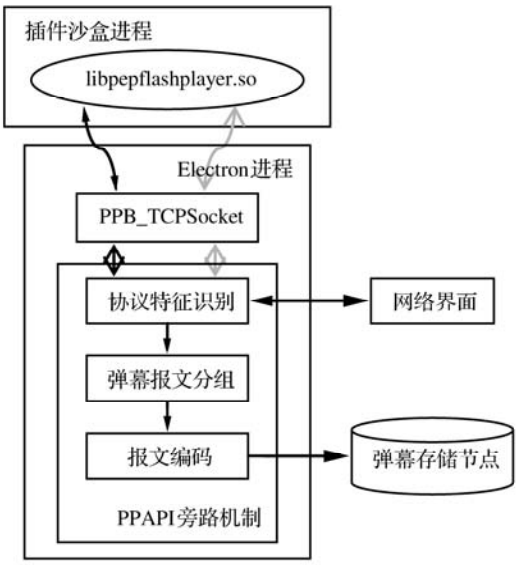

图 3 PPAPI 旁路数据流示意

如图 3 所示,修改渲染引擎实现,在 PPB_TCPSocket 接口的 Read 接口实现中监视插件的网络流量,对其中属于目标弹幕协议的报文进行解析,并以内部协议转发解析结果给弹幕存储组件。而 Flash 插件并不能感知这些修改,使得此机制无需担心被识别进而被限制。

由于完整浏览器的代码规模十分庞大,我们基于支持 PPAPI 并且规模适中的 Electron 项目进行修改。Electron 是基于 Blink 渲染引擎的桌面应用框架(Blink 是 Google 公司的 WebKit fork),支持通过 WebView 方式嵌入网页,并能与网页交换数据。考虑到服务器环境没有图形界面,无法直接启动浏览器实现乃至 Flash 插件,我们使用 Xvfb 作为 X11 显示服务器以承载模拟浏览器窗口。由此,每个房间连接在网页以及 Flash 看来即犹如 Linux 下 Chrome 浏览器用户的正常浏览过程,实现了对可能的前端防爬虫机制的规避,何况系统信息也可通过修改浏览器相应接口的方式对网页端 JavaScript 脚本予以欺骗。PPAPI 浏览器模拟的实现如算法 8 – 算法 10 所示。

算法 8 ShimThread(Site, Room)

1. 设置退出 flag 为假。

2. 无限循环:

a) 启动 ShimProcess(Site, Room)子进程。

b) 每隔固定时间检测退出 flag:

i. 为真:终止 PID,退出循环;

ii. 为假:检查子进程状态,若异常退出则 continue 外层循环以恢复连接,否则什么也不做。

算法 9 ShimProcess(Site, Room)

1. 初始化循环缓冲区 RingBuffer。

2. URL = 生成房间网址(Site, Room)。

3. 将内置 WebView 指向 URL,等待 Flash 插件自动加载。

4. 重载 WebView 资源请求逻辑,拒绝所有图像、声音、视频流的请求以节省流量,只放行播放器初始化所需脚本与 Flash 播放器本身。

并在渲染引擎的 PPB_TCPSocket 接口实现中:

修改 Read 接口实现,接口函数指针原型:int32_t (*Read)(PP_Resource tcp_socket, char *buffer, int32_t bytes_to_read, struct PP_CompletionCallback callback)[17]。

a) 正常执行 Bytes_Read = recv(tcp_socket, buffer, bytes_to_read);

b) 添加:调用 SideChannel(Site, buffer, Bytes_Read, RingBuffer);

c) 正常回调 callback。

算法 10 SideChannel(Site, Buffer, Length, RingBuffer)

1. 检查 Buffer 头部是否符合服务 Site 协议特征,不符合则 return。

2. 执行 ProcessIncomingDanmaku(Site, Buffer, Length, RingBuffer)。

2 实验与分析

2.1 实验环境

我们在国内某知名弹幕服务进行了实验,该服务有开放 API 可供利用,并提供了相关协议文档。出于实验目的,我们也选用该服务作为“不开放 API”的测试服务。我们用 Python 实现了开放 API 的轻量级客户端以及系统的其余部分,用 JavaScript 实现了不开放 API 弹幕服务到爬虫系统内部协议的转换模块,用 C 语言实现了该服务弹幕协议的识别与解析模块。

测试环境为 3 个 Linux 节点,其中两个为物理节点,一个为虚拟节点,所有节点各有一个公网 IPv4 地址。爬虫系统中其他组件分别为消息队列 RabbitMQ、Redis 以及数据库 PostgreSQL。这些服务均以 Docker 容器的形式部署。

2.2 开放 API 抓取实验

实验前,预先使用单节点单 IP 地址进行了测试,发现单个 IP 地址最多只能建立 200 个房间连接。为确认连接数限制发生在何处,同时进行了由校网络中心内节点到校外服务器的高并发心跳连接测试,结果表明能够维持至少 500 条并发连接且无一中断。这排除了校网络中心对所有出站连接数量进行限制的可

能,说明了该弹幕服务的开放 API 仅允许单 IP 保持 200 条连接。由于总共有 3 个 IPv4 地址可供使用,本次实验最多可同时维持该服务 600 个房间的连接。

为节省连接数,我们同时注意到没有必要抓取小流量直播间的弹幕数据,因为这些语料将被大流量房间的海量数据所淹没,不具备统计学意义,因此为系统增加了自动断开连续空闲 5 分钟的房间连接的机制。

我们抓取了位于该服务所有房间列表第一页的直播间。本次实验进行了 26.9 h,总共获取了 7 931 385 条聊天弹幕与 5 077 451 条非聊天弹幕,抓取的原始报文如图 4、图 5,运行状态如图 6、图 7。平均抓取速度达到每秒 134 条,实际峰值速度达到每秒 1 000 条。

```
5881,2016-11-21 13:59:53.650104+00,site_a,58428,type
@=chatmsg/rid@=58428/ct@=2/uid@=1277211/nn@=让我一个
人讲/txt@=这tk对线好菜/cid@=07e539080f2447519c6e0000
00000000/ic@=avatar@S001@S27@S72@S11_avatar/level@=4
/el@=/
5882,2016-11-21 13:59:07.766293+00,site_a,1065655,ty
pe@=chatmsg/rid@=1065655/ct@=1/uid@=9699046/nn@=白猫
与巧克力/txt@=你们下路炸了/cid@=0a5377d44add4234e8d0
0a0000000000/ic@=avatar@S009@S69@S90@S46_avatar/leve
l@=14/gt@=2/rg@=4/dlv@=3/dc@=2/bdlv@=3/el@=eid@AA=15
00000005@ASetp@AA=1@ASsc@AA=1@ASef@AA=0@AS@S/
5883,2016-11-21 13:59:10.910592+00,site_a,1065655,ty
pe@=chatmsg/rid@=1065655/ct@=1/uid@=2603759/nn@=as89
2499285/txt@=这个肉狗/cid@=0a5377d44add4234fbd00a000
0000000/ic@=avatar@S002@S60@S37@S59_avatar/level@=2/
el@=/
```

图 4　聊天弹幕示例

```
1767,2016-11-21 14:02:44.911173+00,site_a,lolff,type
@=uenter/rid@=319721/uid@=14204857/nn@=417650010/lev
el@=11/ic@=avatar@S014@S20@S48@S57_avatar/rni@=0/el@
=eid@AA=1500000005@ASetp@AA=1@ASsc@AA=1@ASef@AA=0@AS
@S/
1768,2016-11-21 13:58:58.259471+00,site_a,58428,type
@=spbc/sn@=小太Yang丶/dn@=B3RB3R/gn@=火箭/gc@=1/drid
@=1404046/gs@=6/gb@=1/es@=1/gfid@=196/eid@=7/bgl@=3/
ifs@=0/rid@=58428/gid@=-9999/bid@=30095_1479736738_2
34/sid@=77675543/cl2@=0/
1769,2016-11-21 13:59:01.6166+00,site_a,58428,type@=
uenter/rid@=58428/uid@=32186392/nn@=zhaown14569/leve
l@=17/gt@=2/ic@=avatar@S032@S18@S63@S92_avatar/rni@=
0/el@=eid@AA=1500000003@ASetp@AA=1@ASsc@AA=1@ASef@AA
=0@AS@Seid@AA=1500000004@ASetp@AA=1@ASsc@AA=1@ASef@A
A=0@AS@Seid@AA=1500000005@ASetp@AA=1@ASsc@AA=1@ASef@
AA=0@AS@S/
```

图 5　非聊天弹幕示例(全站礼物广播、用户进入房间)

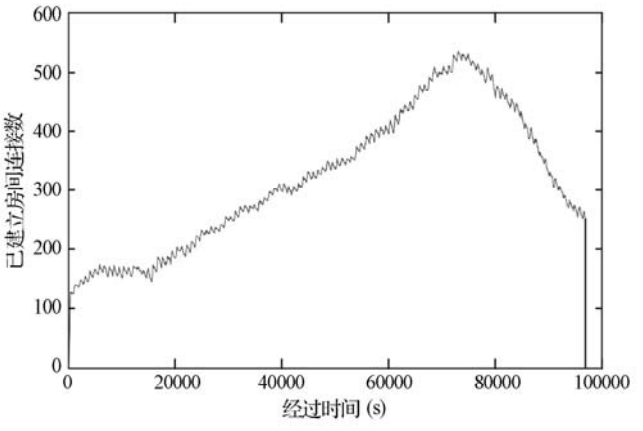

图 6　已建立房间连接数

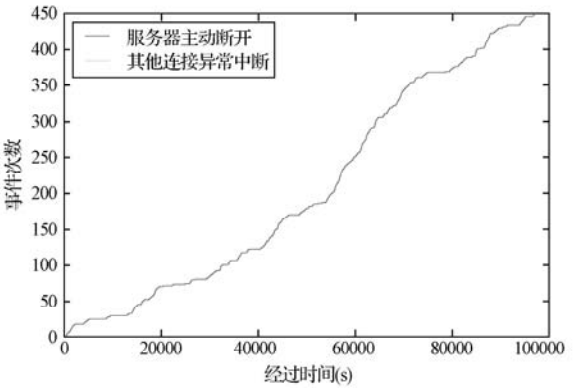

图 7　断开连接原因

如图 6 所示,拥有足够 IP 地址资源的系统确实能够承载相应的连接数量,并有效控制自身所维持的连接数。如图 7 所示,其他原因造成的连接中断消失,服务器主动断开连接的频率也大大降低,每分钟 0.277 次的断开连接频率已经与使用官方客户端观看直播的弹幕服务器断开频率接近,完全满足稳定运行的要求。

2.3　PPAPI 旁路接口抓取实验

虽然上述弹幕服务已经提供了开放 API,出于验证目的,我们仍然用其文档辅助实现了 PPAPI 旁路的弹幕协议识别与解析模块,并以此种方式进行了抓取。实际应用中,面对未提供协议文档的弹幕服务,这部分知识将通过流量嗅探、逆向工程等方式获取。由于内存占用的关系,我们仅仅选择 3.2 节中使用的房间列表的前 10 个房间进行抓取。

由于实验规模很小,且结果与开放 API 情形十分相似,实验数据从略。值得注意的是,因为每个直播间均对应一个模拟浏览器实例,且 Flash 插件运行需要虚拟 X11 显示服务器支持,内存占用十分惊人。实际测量单个房间连接的内存使用与 Chrome 浏览器平均每个标签页类似,为数十 MB 级别,这与轻量级客户端数十 KB 的内存占用差距甚大,而且属于浏览器模拟机制的固有特性,不便进一步降低。这说明不开放 API 弹幕服务的抓取应为权宜之计,长远来看,要对弹幕进行大规模的外部研究乃至监管,依然需要弹幕服务本身配合。

3　结　语

对于弹幕信息处理而言,实现高效可靠地采集是第一步,而目前国内外此方面研究尚属空白。本文针对我国直播行业国情,创新地提出并实现了一种分布式直播弹幕爬虫方案,并在真实生产环境条件下进行了实验,表明该爬虫具有良好的工作性能,能够实时抓取峰值至少每秒 1 000 条量级的弹幕数据,并实现了

与官方客户端相近的连接中断频率,完全可以用于生产环境的数据抓取。

展望未来研究方向,我们仍然需要开发相应的数据处理流水线,如此才能将原始弹幕语料转化为有效信息以支持各类下游应用。我们希望本研究能够在弹幕信息处理的领域抛砖引玉,通过收集弹幕信息,汇编弹幕语料库,以使更大范围的直播弹幕研究成为可能,进而促进弹幕信息处理手段的发展。

另外值得指出的是,虽然弹幕信息研究的需求日渐上升,各大弹幕服务对弹幕信息利用仍然缺乏开放性,法律地位大多也仍然不甚明确。因此本技术在产业化过程中必然面临需要向各服务单独获取授权及相应接口的境况。对于未来仍将继续发展的弹幕产业而言,各大弹幕服务有必要认识到构建开放生态的重要性。

参 考 文 献

[1] Claypool M, Farrington D, Muesch N. Measurement-based analysis of the video characteristics of Twitch. tv[C]// IEEE Games Entertainment Media Conference. 2015.

[2] Nascimento G, Ribeiro M, Cerf L, et al. Modeling and Analyzing the Video Game Live-Streaming Community [C]// Latin American Web Congress. 2014:1 - 9.

[3] Twitch. tv. justintv/Twitch-API[EB/OL]. 2016(2016/06/29)[2016/06/29]. https://github. com/justintv/Twitch-API/tree/0d06204.

[4] 彭轲, 廖闻剑. 基于浏览器服务的网络爬虫[J]. 硅谷, 2009(4):53 - 54.

[5] 刘兵. 基于 JavaScript 等多链接分析的主题爬虫设计实现[J]. 许昌学院学报, 2010, 29(2):87 - 90.

[6] 张媚. Ajax 友好的网络爬虫设计与实现[D]. 暨南大学, 2011.

[7] 管翠花. 支持 Ajax 技术的 Deep Web 网络爬虫模型研究[D]. 大连海事大学, 2011.

[8] 段子飞. 支持 Ajax 的 Deep Web 网络爬虫系统的设计与实现[D]. 华南理工大学, 2015.

[9] 钱程, 阳小兰. 一种支持 Ajax 框架的网络爬虫的设计与实现[J]. 计算机与数字工程, 2012(4):69 - 71.

[10] 魏少鹏, 夏小玲. 基于 Chrome 扩展的爬虫系统设计与实现[J]. 软件导刊, 2016, 15(3):76 - 80.

[11] 郭津丞, 冯超, 张磊. 基于 WebKit 的网络爬虫[J]. 现代电子技术, 2013(18):62 - 64.

[12] 钱晨, 张晓静. 网络 Flash 爬虫搜索方法比较研究[J]. 中国教育技术装备, 2014(14):32 - 34.

[13] 杭州边锋网络技术有限公司. 战旗直播注册协议[EB/OL]. 2016(2016/04/23)[2016/04/23]. http://www.zhanqi. tv/common/agreement. html.

[14] 武汉斗鱼网络科技有限公司. 斗鱼—用户注册协议[EB/OL]. 2016(2016/04/23)[2016/04/23]. http://www.douyu. com/protocal/member.

[15] 广州华多网络科技有限公司. 多玩通行证——用户协议[EB/OL]. 2016(2016/04/23)[2016/04/23]. http://zc.yy. com/license. html.

[16] 上海熊猫互娱文化有限公司. 熊猫 TV 用户注册协议[EB/OL]. 2016(2016/04/23)[2016/04/23]. http://www. panda. tv/agreement. html.

[17] Google. Pepper API Reference (Stable)[S/OL]. 2016(2016/04/23)[2016/04/23]. https://developer. chrome. com/native-client/pepper_stable.

(上接第 64 页)

夫距离更为简单,专家权重能够随着评估数据的变化而动态调整,从而使得专家群决策的作用更加明显;根据模糊理论中的模糊合成法则,可求取失效模式的模糊风险优先数,需要指出的是,三角模糊数的去模糊化方法较多,但在实际应用中,需要根据专家评价的模糊区间数具体的形态选择合适的去模糊化方法,否则将使得最终排序结果发生变化。数值计算的结果表明该方法在工艺模式失效评估中具有良好的指导意义,未来将针对风险因子权重确定提出优化模型,并进一步研究维度取值变化对最终排序结果的敏感度问题。

参 考 文 献

[1] Joshi G, Joshi H. FMEA and Alternatives v/s Enhanced Risk Assessment Mechanism[J]. International Journal of Computer Applications, 2014, 93(14):33 - 37.

[2] 刘卫东, 胡坤, 郑慧萌,等. 多品种小批量定制生产模式的工艺失效模式及影响分析[J]. 计算机集成制造系统, 2016, 22(6):1485 - 1493.

[3] Liu H C, You J X, You X Y. Evaluating the risk of healthcare failure modes using interval 2-tuple hybrid weighted distance measure[J]. Computer & Industrial Engineering, 2014,78:249 - 258.

[4] 王晓峰, 申桂香, 张英芝,等. 基于群体决策和多种赋值方式的加工中心关键部件 RPN 分析[J]. 吉林大学学报(工), 2011, 41(6):1630 - 1635.

[5] 王浩伦, 徐翔斌, 甘卫华. 基于三角模糊软集的 FMEA 风险评估方法[J]. 计算机集成制造系统, 2015, 21(11):3054 - 3062.

[6] 聂文滨, 刘卫东, 胡坤,等. 基于群决策和广义豪斯多夫距离的工艺失效风险评估[J]. 计算机集成制造系统, 2015, 21(9):2484 - 2493.

[7] 耿秀丽,张永政. 基于犹豫模糊集的改进 FMEA 风险评估方法[J]. 计算机集成制造系统,2017,23(2):340 - 348.

[8] 冉静学. 三角模糊数排序方法的研究[J]. 中央民族大学学报(自然科学版), 2011(4):37 - 42.