北京大学

博士研究生学位论文

题目：基于云计算的异构自适应内容分发

姓　　名	：	李 振 华
学　　号	：	10948866
院　　系	：	信息科学技术学院
专　　业	：	计算机科学与技术
研 究 方 向	：	云计算, 互联网, 大数据
导 师 姓 名	：	代 亚 非　教授

二〇一三年四月

版权声明

摘 要

互联网存在的最基础意义就是内容分发，即将数字内容从一个节点分发到另一个或多个节点。从2006年亚马逊公司推出“弹性计算云”和2007年苹果公司推出“iPhone”开始，互联网内容分发呈现非常明显的两极分化趋势：一方面，世界各地投入巨资兴建重量级、集成化的数据中心（简称“重云”），以追求内容分发的规模效应和成本优势；另一方面，用户终端日益轻量化、移动化（简称“轻端”），且在软硬件及网络配置上高度异构，从而对内容分发的流量、能耗、速率、时延等提出严苛要求。

针对上述“重云轻端”趋势，本文研究基于云计算的异构自适应内容分发，设计与具体应用场景相适应的更为节流、节能、高速、经济的新型内容分发技术。在科研方法论上特别注重实用性：从实际系统中发现问题、在实际环境中解决问题、并进行适度的提升与展开。研究工作立足于Dropbox（全球最大的云存储系统）、QQ旋风（腾讯公司主要的内容分发平台）、AmazingStore（教育网文件分享系统）和CoolFish（中科院视频点播系统）等系统平台。具体来说，本文的主要创新成果包括：

(1) **云存储中的内容分发**：针对云存储内容分发中普遍存在的“同步流量滥用问题”，从网络协议、操作系统两个层面解析其发生机制，提出和实现“高效批同步算法”解决该问题、节省同步流量，并通过（兼容性地）修改Linux内核从源头上根治这一问题。
(2) **云辅助的内容分发**：通过对QQ旋风和迅雷的系统性研究，提炼出“云跟踪”加速内容分发的基础模型，并设计了最大化“带宽放大效应”的“云调度”算法以高效利用云带宽，该算法在CoolFish系统进行了原型部署。
(3) **完全依赖云的内容分发**：鉴于移动用户对视频内容分发的特殊需求、尤其是节能与节流，测量和分析了大规模工业“云下载”及“云转码”系统，诊断系统性能瓶颈，提出缓存、预测和推荐等一系列优化方案。
(4) **用户构造云的内容分发**：从经济的角度，可以将高动态性和高异构性的互联网用户组织起来、构造稳定可靠的虚拟云。我们为此类构造方案设计了“稳定性最优的端用户分组策略”和“流媒体数据源最快切换策略”。

关键词：互联网，内容分发，云计算，大数据，异构，自适应。

Cloud-based Heterogeneity-adaptive Content Distribution

Zhenhua Li (Computer Science and Technology)
Directed by Prof. Yafei Dai

The most fundamental function of Internet lies in *content distribution*, *i.e.* distributing digital content from one node to another node or multiple nodes. Since Amazon's release of EC2 in 2006 and Apple's launch of iPhone in 2007, Internet content distribution has shown a strong trend of *polarization*. On one hand, great fortune has been invested in building heavyweight and integrated data centers (abbreviated as "Heavy-cloud") all over the world, in order to pursue the *economies of scale* and the *cost advantage* of content distribution. On the other hand, end user devices (abbreviated as "Light-end") have been becoming more and more lightweight, mobile and heterogeneous, thus posing rigorous requirements on the traffic volume, energy consumption, transfer rate and latency of content distribution.

Motivated by the abovementioned trend of "Heavy-cloud *vs.* Light-end", this dissertation investigates the "*cloud-based heterogeneity-adaptive content distribution*" by designing novel traffic-saving, energy-efficient, high-rate, and economical content distribution technologies that automatically adapt to specific application scenarios. Our research methodology pays special attention to *practicality*, that is to say, we discover practical problems in real systems, solve the problems in real environments, and make further enhancements and extensions. Specifically, our research stands on real-world systems such as Dropbox (*i.e.* the biggest cloud storage service across the world), QQXuanfeng (*i.e.* the major content distribution platform of the Tencent company), AmazingStore (*i.e.* a file-sharing system residing in CERNET) and CoolFish (*i.e.* a VoD system mainly serving the Chinese Academy of Sciences).

The major contribution in this dissertation can be summarized as follows:

(1) **Cloud storage content distribution**. With regard to the "data sync traffic overuse problem" that pervasively exists in cloud storage (related) content distribution, through comprehensive measurements and reverse engineering of Dorpbox-like cloud storage services, we resolve the pathological principle of the traffic overuse problem from both network protocol and operating system

perspectives. Then, we propose and implement the *update-batched delayed synchronization* (UDS) mechanism to efficiently solve the problem and save the synchronization traffic. Moreover, we extend UDS with a (compatible) Linux kernel modification that further improves its performance.

(2) **Cloud-assisted content distribution**. By means of large-scale measurements and analysis of the QQXuanfeng and Xunlei systems, we extract the basic model of "cloud tracking" content distribution. Thereby, we design the efficient "cloud scheduling" algorithm to maximize the cloud "bandwidth multiplier effect". A prototype implementation of this algorithm is deployed on top of the CoolFish system.

(3) **Cloud-relied content distribution**. Driven by mobile user devices' special requirements on video content distribution, in particular the traffic and energy effectiveness, we measure and analyze the industrial "cloud downloading" and "cloud transcoding" systems, diagnose their performance bottlenecks, and propose the corresponding optimization schemes such as cache, prediction, and recommendation.

(4) **User-formed cloud content distribution**. From an economical point of view, the highly dynamic and heterogeneous Internet end users can be organized to form a stable and reliable virtual cloud. In order to better organize the users, we design the "stability-optimal grouping strategy of end users" and the "fast source switching strategy of end users".

Key Words: Internet, Content distribution, Cloud computing, Big data, Heterogeneity-adaptive.

目 录

第一章　引言

1.1 互联网内容分发

互联网存在的最基础意义就是**内容分发**，即将**数字内容**从一个**节点**分发到另一个或多个节点。**数字内容**包括视频、音频、图像、软件、文档、网页、电子邮件、即时消息等。**节点**既可以是巨大的数据中心或服务器集群，也可以是普通的路由器、交换机、个人电脑、平板电脑或智能手机，还可以是微小的传感器、电子标签等等。结合时间先后与模式差异，互联网内容分发的传统技术大致可分为如下4类：

(1) **直接内容分发（Client/Server、简称C/S）。**互联网浪潮最早兴起于1994年的美国硅谷，典型代表是Yahoo!（雅虎）和Netscape（网景）公司。那时的互联网数字内容主要是网页和电子邮件、也包含部分图片，不管从种类、数量、容量上看都极为有限，故而采用最简单的C/S技术直接分发内容即可满足用户需求，如图1.1所示。对直接内容分发最经典的实现是1970年代的UNIX Socket（见IETF RFC-33文档），其作者之一为Vincent Cerf（“互联网之父”、图灵奖得主、现任ACM主席）。

(2) **内容分发网（Content Distribution Network、简称CDN）。**随着互联网浪潮愈演愈烈，数字内容迅速覆盖大容量的多媒体内容、特别是视频，在数量上也呈现指数级增长，直接分发内容造成互联网普遍拥堵，World Wide Web恶化为“World Wide Wait”。1995年，WWW发明人Tim Berners-

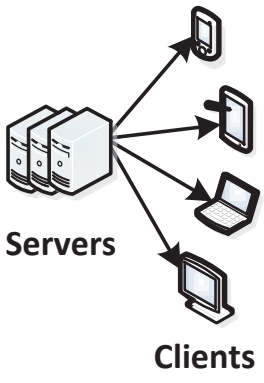


图 1.1　直接内容分发（C/S）

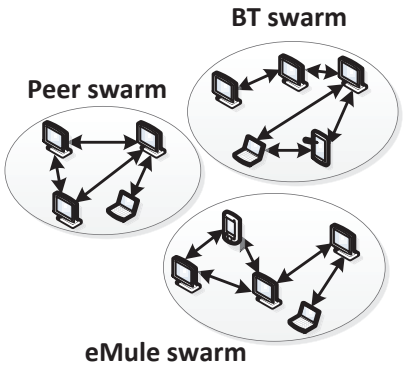


图 1.2　对等网络（P2P）

Lee向麻省理工学院的同事提出一个挑战性问题："能否发明一种全新的、从根本上解决问题的方法来分发互联网内容？"解决这个问题的正是他隔壁办公室的Tom Leighton，其团队首次提出CDN的设想——在互联网上多个位置策略性地部署一系列**边缘服务器（Edge server）**，边缘服务器中保存的内容根据用户请求的状况动态调整，以尽量保证每个用户都能从附近的服务器快速获取内容——并最终于1998年成立了全球首个CDN公司Akamai。直到今天，CDN依然是互联网内容分发最基本的加速技术，典型代表还包括Limelight、Level3以及中国的ChinaCache（即蓝汛公司，也成立于1998年）和ChinaNetCenter（即网宿科技，成立于2000年）。

(3) **对等网络（Peer-to-Peer network、简称P2P）**。CDN内容分发有其不足之处：首先，CDN公司只服务那些付费的数据源（通常是大型网站）；其次，即使对于付费的网站，CDN受限于自身的存储、带宽和ISP覆盖范围，也只能加速部分内容的分发。一个（今天看来）很自然的思路是：既然每个互联网用户都有一定的存储容量和传输带宽、且互联网用户数目庞大，有没有可能绕过CDN、用户之间直接传输数据、形成自发自足的**内容分发集群（Data swarm或Peer swarm）**？1999年，Napster音乐分享系统给予前述问题肯定且辉煌的答案，也从而开启了P2P内容分发技术（如图1.2所示）的繁盛。紧随其后一系列耳熟能详的系统：BitTorrent（简称BT）、eDonkey/eMule（电驴/电骡）、KaZaa、Skype、PPTV、PPStream、UUSee等，充分证明了互联网用户端资源聚沙成塔所蕴含的巨大能量。

(4) **混合式内容分发（Hybrid content distribution）**。P2P内容分发同样有其缺点：首先，用户端设备极不稳定（**高动态性**）；其次，用户端设备能力悬殊（**高异构性**）；最后，不管是用户还是内容都缺乏声誉/质量认证（**高风险性**）。这"三高"导致P2P内容分发的性能难以预测、更难掌控，用户的服务质量得不到保障。从而混合式内容分发技术应运而生，兼容C/S或CDN的稳定与可靠、P2P的廉价与自扩展，取长补短、相得益彰；当然不可避免的是系统实现更为复杂。目前在工业界我们已经很少看到纯粹的P2P系统，如PPTV、PPStream、UUSee、风行等都转换成了混合式架构；另一方面，以前单纯依赖C/S或CDN的系统也纷纷整合P2P，如优酷、土豆、搜狐视频、LiveSky [1] 等。

1.2 重云轻端及异构性

从2006年亚马逊公司推出“弹性计算云”（EC2）和2007年苹果公司推出“iPhone”开始，互联网内容分发呈现明显的两极分化趋势：

- 一方面，近年来世界各地投入巨资兴建重量级、集成化的大规模数据中心（简称“**重云**”），用于云计算 [2]（如亚马逊EC2、谷歌AppEngine、Apache Hadoop和Cassandra、阿里云和百度云等）、云存储 [3]（如亚马逊S3、微软Azure、苹果iCloud和Dropbox等）、虚拟化 [4]（如Vmware、Virtualbox、Xen和Hypervisor 等）及流媒体播放 [5]（如Youtube、Netflix、Hulu和iTunes等）等。基于上述广义云计算平台的内容分发系统一般将海量内容集中存储，同时大批量购买ISP或CDN带宽，还可能自主进行内容迁移和带宽调度，以追求内容分发的规模效应和成本优势。
- 另一方面，用户终端日益轻量化、移动化（简称“**轻端**”），且在软件、硬件、网络环境和内容生成方式上高度**异构**，对内容分发的开销、速率、时延、能耗提出严苛要求。iPhone、iPad的推出，特别是Android开放手机联盟的蓬勃发展，将移动设备的功能从传统的打电话和发短信迅速扩展到几乎一切互联网应用，并且移动设备的数量已远远超过个人电脑。虽然如此，当前互联网内容分发的主流技术依然是面向个人电脑的，照搬到移动场景往往水土不服、差强人意——移动设备的屏幕大小不一，所支持的音视频格式和解析度也不尽相同 [6]，并且它们的网络流量（当工作于GPRS、3G、4G模式时）、处理器能力和电池容量都极为有限 [7]，这些异构性都给移动设备的内容分发带来严苛的要求和很大的挑战。

事实上，“重云”和“轻端”是相辅相成的：正是因为云端越来越“重”（集成化、规模化），用户端才可以越来越“轻”，典型事例是iPhone每一代都比前一代更轻更薄却能完成更多功能、并且越来越多的功能依靠后台云计算；反过来，用户端对更“轻”、更异构的渴望，也必然要求云端更“重”、更强、更智能。此外，我国政府在《国民经济和社会发展第十二个五年规划纲要》（简称“十二五规划”）的“中国云规划”专项方案中指出：要进一步加快加强云计算服务平台和移动互联网、物联网、广播电视网和工业专用网的深度融合。在这一深度融合的进程中，必然会面临不同网络间高度异构性的挑战，从而进一步强调了研究“重云轻端”的异构内容分发技术的迫切意义。

1.3 相关前沿技术调研

我们对近几年出现在工业界和学术界的前沿互联网内容分发技术进行了调研，特别关注那些面向“重云轻端”趋势、并考虑到异构性问题的内容分发技术。相关代表性技术综述如下：

- **多CDN调度分发。**同时购买多家CDN的资源、再由自身集中调度以提升内容分发性能，如图1.3所示。典型代表是新兴视频网站Hulu [8]，它没有沿用传统视频网站（如Youtube [9]和Netflix [10]）的做法，而是根据其自身特点（创业公司、资金不足、但无历史负担）因势利导，同时购买了3家CDN——Akamai、Limelight 和Level3的部分资源，此外Hulu 自身还构建了一个较小规模的数据中心，用来在3家CDN原始资源调度的基础上进行“二次调度”，从而有效地弥补了单个CDN在ISP覆盖范围、实时可用带宽、收费性价比等方面的不足。
- **私有BT（Private BitTorrent）和捆绑BT（Bundling BitTorrent）。**两者都是BitTorrent经典技术的延伸与扩展。前者将BitTorrent用户群限制在一个较为狭窄、但更为积极、且往往具有较高带宽的范围内（本质是强制用户同构），通过牺牲求全责备的全互联网覆盖度换来了用户的可认证性和内容的安全性，进而极大提升了内容质量和分发性能 [11–13]；国内比较著名的代表是“北邮人PT”，在北邮校内非常活跃、广受好评。后者则故意将本来可以独立分发的多份内容**捆绑（bundling）**在一起集中分发，促使用户长期在线、为其他用户贡献更多的资源、进而激发整个系统的繁荣 [14–16]；Bundling BitTorrent最常见的做法是将一部电视连续剧的所有

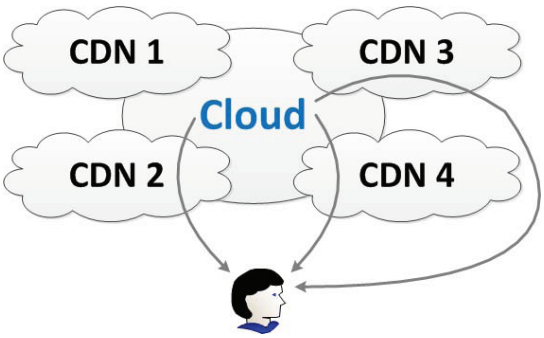


图 1.3　多CDN调度分发

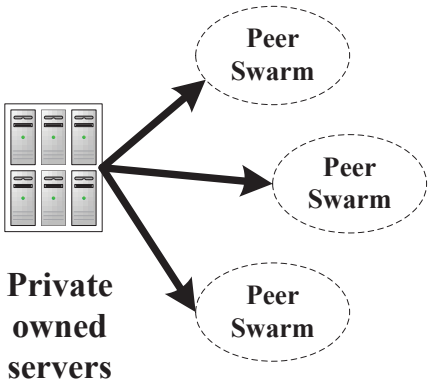


图 1.4　P2SP

剧集捆绑到一起，这个简单却有效的思想已被众多BitTorrent网站乃至P2P网站所采纳。

- **P2SP（Peer-to-Server&Peer）。**允许并引导端用户同时从P2P数据集群和内容服务器获取数据，相当于P2P和C/S的聚合，如图1.4所示。这也是一个极为简单却十分有效的思想，已被Spotify、PPTV、PPStream、UUSee、风行等一系列网站采用 [17, 18]。再进一步还有更为开放的Open-P2SP内容分发技术 [19]，它比P2SP 强大的地方在于——用户可以跨越协议、跨越系统，从完全不同的P2P数据集群（比如一个是BitTorrent集群、另一个是电骡集群）和内容服务器（比如一个是HTTP服务器、另一个是RTSP 服务器）中并行获取数据，典型代表为迅雷 [20]、QQ旋风、Flashget、Orbit和QVoD。但不可忽视的是，P2SP技术在实现上难于P2P或C/S，如果实现不好效果适得其反；而Open-P2SP 技术则更为复杂与困难，所涉及到的已不仅仅是技术问题。
- **迂回内容分发。**也称迂回路由（detour routing） [21]，该技术的出现源于互联网**三角不等式违例（Triangular Inequality Violation）**现象的普遍存在。如图1.5 所示，在互联网中要将文件f从节点A传到节点B，A到B之间的最短路径为*path1*，从理论上讲（假设当前互联网构造是理想的）沿着*path1*传送f应该最快；但实际情况是，由于当前互联网中存在太多的人为因素，选择一个合适的中间节点C，先将f沿着*path2*传送到C、再将f沿着*path3*传送到B，反而可能更快，因此在形式上出现了“三角形两边之和短于第三边”的违例情况。特别是当A、B位于不同的ISP时，“三角不等式违例”十分严重，正是迂回内容分发大行其道的场景。迂回内容分发的典型代表包括腾讯QQ的离线文件传输、迅游公司的迂回游戏加速等，它本质上是对当前互联网缺陷的纠正弥补。
- **自适应HTTP内容分发。**全称“动态自适应HTTP流媒体”（Dynamic Adaptive Streaming over HTTP），简称DASH。苹果公司也有自己的称呼即HTTP Live Streaming（简称HLS），而微软公司称之为Smooth Streaming，原理都和DASH相似。互联网上存在最多的始终是最古老的Web服务器，虽然Web服务器本身是设计用来服务网页请求的、并不适合流媒体，但由于Web服务器的庞大数量和HTTP协议的广泛流行、使得HTTP流媒体分发客观上成为可能。DASH需要对原始HTTP协议进行扩展，Web客户端也要做相应的调整、以根据实时网络状况自动切换到不同比特率的

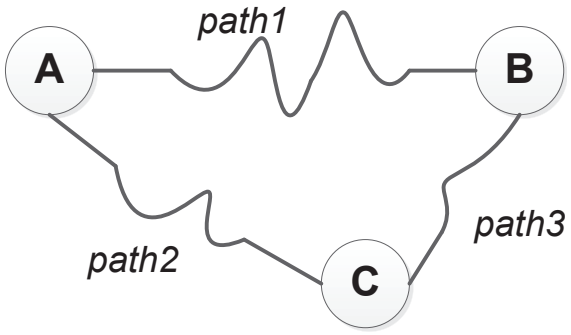


图 1.5　三角不等式违例

流媒体内容。目前DASH依然是一个活跃的研究课题、有较大的优化空间 [22, 23]。

通过对上述相关前沿技术的调研，我们认识到：面对互联网日益加剧的重云轻端趋势，传统的内容分发技术确实需要改进并且实际上正在改进，而用户端繁多的异构性正是改进的动力。作为支撑内容分发的基础设施，云计算平台必须调整架构、优化技术以自适应用户端的异构性。此外，需要改进的不仅仅是HTTP协议，改进的做法也不仅仅是私有BT这样的强制同构或迂回内容分发这样的纠正弥补——下面本文就提出了一系列改进的新架构或新技术。

1.4 本文工作简介

针对互联网内容分发的“重云轻端”趋势，本文研究“**基于云计算的异构自适应内容分发**”，设计与具体应用场景相适应的更为节流、节能、高速、经济的新型内容分发技术。研究内容覆盖**云存储中的**、**云辅助的**、**完全依赖云的**和**用户构造云的**内容分发技术，包括理论算法的探索和工业系统的实践。在科研方法论上特别注重**实用性**：从实际系统中发现问题、在实际环境中解决问题、并进行适度的提升与展开。研究工作立足于Dropbox（全球最大的云存储系统，拥有超过1亿注册用户）、QQ 旋风（腾讯公司主要的内容分发平台，日均服务用户超过500万）、AmazingStore（教育网文件分享系统）和CoolFish（中科院视频点播系统）等系统平台。

对比上一节所述的相关前沿技术，我们的研究工作除实用性外还具有**系统性**和**完备性**的特点。首先，研究立足于当前主流工业系统（特别是大规模的Dropbox和QQ旋风），体现了一定的系统性；其次，研究工作涉及互联网内容分发最主要的几大应用场景：文件存储、文件下载、文件分享和音视频流媒

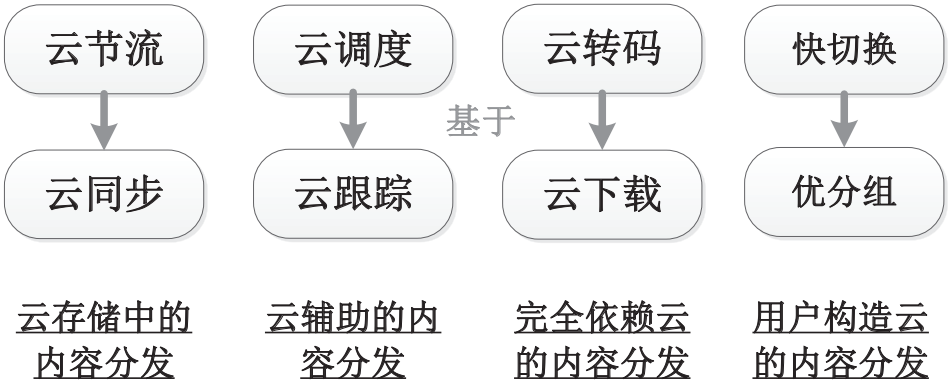


图 1.6　研究工作关系图

体，体现了较好的完备性。

具体来说，本文可总结为如下4个方面、8份工作的集合：

(1) **云存储中的内容分发**，包括
  (a) 云存储的高效批同步算法，简写为“**云同步**” [24,25]；
  (b) 云存储的节流效率研究，简写为“**云节流**” [26,27]；
(2) **云辅助的内容分发**，包括
  (a) 云跟踪系统的挑战设计与性能，简写为“**云跟踪**” [19,28]；
  (b) 最大化带宽放大效应的云调度算法，简写为“**云调度**” [29,30]；
(3) **完全依赖云的内容分发**，包括
  (a) 服务冷门视频的云下载系统，简写为“**云下载**” [31–33]；
  (b) 适应移动视频的云转码系统，简写为“**云转码**” [34]；
(4) **用户构造云的内容分发**，包括
  (a) 稳定性最优的端用户分组策略，简写为“**优分组**” [35,36]；
  (b) 流媒体数据源的最快切换策略，简写为“**快切换**” [37,38]。

上述每个方面的第2份工作(b)都基于第1份工作(a)，前者(b)往往是后者(a)的扩展和延伸，如图 1.6所示；此外，附录 A中描画了一幅“研究工作简图”，图文并茂地简介了各项工作的名称、背景系统、工作原理图和所发表论文。下面我们概要介绍每一份研究工作的动机和方案：

本文第2章第1部分（“云同步”），我们发现以Dropbox为代表的众多云存储系统在内容分发过程中普遍存在“同步流量滥用问题”，尤其当用户数据流呈现“频繁短促更新模式”时、用于维护分发过程的“控制流量”反而远远超

过实质分发内容的“数据流量”，从而给云端、客户端以及传输网络都带来沉重却并非必要的流量负担。为解决这一问题，我们首先对云存储内容分发过程进行了一系列精心设计的测量，从网络协议、操作系统两个层面分析出该问题的发生机制；然后提出并实现了“高效批同步算法”解决该问题，在不明显影响用户体验的前提下、有效避免了流量滥用。最后，我们通过修改Linux内核从源头上根治了这一问题，并且进一步提升了高效批同步算法的性能。

本文第2章第2部分（“云节流”），面对近年来云存储应用迅速流行、国际国内众多云存储系统激烈竞争的局面，考虑作为云存储服务的核心操作——“数据同步”消耗巨大网络流量，因此在不损害用户体验的前提下、云存储系统的设计必须尽量节省同步流量。在本文中，我们提出一个新的指标TUE（*traffic usage efficiency*）来衡量云存储数据同步的节流效率，并进行了一系列特殊设计的实验来全面发掘影响TUE的主要因素。我们对TUE的洞察可以帮助云存储服务提供者设计更为经济、节流的数据同步机制，同时还可以指导用户选择适合自身需求的云存储服务。

本文第3章第1部分（“云跟踪”），通过对QQ旋风、迅雷的大规模、系统性研究，提炼出“云跟踪”加速内容分发的基础模型、关键指标、核心优势和挑战问题，并结合QQ旋风系统的实际架构与性能深入阐述了设计（大规模、开放式）云跟踪系统所要注意的权衡和需要避免的误区。

本文第3章第2部分（“云调度”），首先提出“带宽放大效应”的概念及其对于“混合式CloudP2P”内容分发系统的重要意义，随后基于大规模测量数据建立起带宽放大效应的形式化模型，进而设计了能够最大化带宽放大效应的“云调度”算法、以高效利用云带宽。该算法既在QQ旋风系统数据集上进行了模拟实验，又在CoolFish系统上进行了原型部署实验。

本文第4章第1部分（“云下载”），反思当前互联网冷门视频内容分发的现状，发现传统技术（CDN和P2P）都因为各自的局限性无法提供高质量的冷门视频分发服务。之后结合QQ旋风离线下载系统的大规模测量和分析，抽象出“云下载”内容分发的基本架构，归纳出评价其性能的关键指标（特别专注于冷门视频内容分发），并详细研究了一系列相关设计机制，包括系统硬件配置、缓存容量规划、缓存替换算法、（冷门视频）分发加速策略、（冷门视频）下载成功率提升等。

本文第4章第2部分（“云转码”），鉴于移动用户对视频内容分发的特殊

需求、尤其是节能与节流，考虑到当前互联网视频内容分发依然主要面向个人电脑、照搬到移动场景往往性能不佳，专门研究“移动视频转码”这一亟待解决的问题。为此我们设计并实现了一个原型“云转码”系统，代替移动设备下载视频、转码视频、再将转码后的视频快速分发给它们，从而最小化移动设备的资源（主要是流量和电量）消耗。此外，我们还诊断出系统性能瓶颈，并提出相应的缓存、预测和推荐等优化方案。

本文第5章第1部分（“优分组”），从经济的角度，可以将高动态性和高异构性的互联网用户组织起来、构造稳定可靠的虚拟云。为此我们提出将端用户分组、每个分组作为一个更为强大和稳定的虚拟云节点以承担更为重要的内容分发角色（比如Skype系统中的中继节点、电驴/电骡系统中的超节点），并且设计了“稳定性最优的端用户分组策略”，在保证系统可扩展性的前提下、能够最大化用户分组的稳定性。我们使用人工合成数据集、AmazingStore和CoolFish两个真实系统数据集对该策略进行了模拟实验。

本文第5章第2部分（“快切换”），面向存在多个数据源且数据源经常切换的流媒体内容分发场景（如多人视频会议）提出一个基础问题：如何最小化数据源切换的时延？我们首先给数据源切换过程建立了数学模型，从而将上述问题形式化为一个数学优化问题，然后推导出此优化问题的最优解。鉴于实际系统情况的复杂性与网络环境的动态性，提出一个称为“流媒体数据源最快切换策略”的实用贪心算法，通过交错旧数据源与新数据源的数据传递来趋近理论最优解。我们基于多个真实系统数据集进行了模拟实验。

上述各项工作都以研究问题的背景、动机及工作简介作为开始，其次对该问题的国际研究现状进行综述，再次是我们对于这个问题所做的工作，最后以小结和进一步工作作为结尾。此外，本文还在第6章结束语中简单总结了全文的线索，并提出了互联网内容分发领域一些值得研究的问题。最后是参考文献和读博期间发表论文列表。

第二章　云存储中的内容分发

2.1 云存储的高效批同步算法

2.1.1 背景、动机及工作简介

作为个人存储、文件同步和数据分享的有力工具，**云存储服务**（如Dropbox、Google Drive、OneDrive、iCloud Drive等）近年来极为流行，为用户提供了泛在、可靠、安全的数据存储，并且可以方便地**同步**数据到多台设备或多个用户。在国际国内大量的云存储系统中，Dropbox出现最早也最为流行，截至2012年末已拥有超过1亿注册用户、日均存储或更新约10亿个文件 [39]。

一个云存储系统典型地由两部分构成：（1）前端客户端，安装并运行在用户设备上；（2）后端云，存放用户数据，一般由大规模数据中心组成。用户在安装客户端应用时一般需要指定一个特殊的“**同步文件夹**”，每当用户在此文件夹中创建文件或修改文件时，所做的数据更新都会被客户端应用自动同步到后端云，同步过程对用户几乎完全透明。

考虑到数以亿计的用户已经将他们的数据托付在云存储中，云存储应用如何有效地利用网络资源、进行经济高效的内容分发就变得极为重要。Dropbox类云存储系统通常采用两个方法来减少网络流量开销。首先，当发生数据更新时，客户端计算被更新文件的**二值差异（Binary diff）** [40] 并且仅仅发送此二值差异（即文件中被改变的那部分）到云端。其次，在任何更新被发送到云端之前、客户端都会压缩它。一个简单的例子是，当我们向Dropbox同步文件夹中的一个现存文件中添加100 MB的相同字符（比如说‘a’）时，二值差异的大小显然为100 MB，但此次文件修改只消耗了约40 KB的网络流量——略多于仅添加一个字符‘a’所导致的网络流量（约38 KB）。

虽然如此，我们观察到当面临**频繁短促数据更新（frequent, short data updates）**时，云存储应用所产生的网络流量仍然呈现“病态的低效”。具体来说，每当同步文件夹中发生数据更新时，云存储应用当前普遍采用的**更新触发的实时同步机制（Update-triggered Real-time Synchronization，简写为URS）**就会计算二值差异并压缩之，再将压缩后的二值差异发送到云端——这一过

程不可避免地要同时发送一些**会话维护数据（Session maintenance data）**，包括DNS查询、TCP/HTTPS连接的建立和维护、数据索引等操作所需要的数据。故而当用户数据流呈现频繁短促数据更新模式时，会话维护流量将远远超出实质更新内容的数据流量，从而给云端、客户端以及传输网络都带来沉重却并非必要的流量负担。我们称上述情况为云存储内容分发的**“同步流量滥用问题”（Data sync traffic overuse problem）**。从本质上说，流量滥用问题源自前述URS同步机制的**更新敏感性**。

我们对于流量滥用问题的研究表明：这一问题不仅在用户中普遍存在，而且在各大主流云存储系统中屡见不鲜。2012年，Drago等人对Dropbox系统进行了一次大规模测量 [41]，其公布的测量数据 [42]显示：系统中至少有8.5%的用户、其超过10%的数据流量来自于“频繁短促数据更新”。当某文件被多个用户分享时，流量滥用问题会变得更为尖锐，因为一个用户（对该文件）所产生的频繁短促数据更新将强迫所有分享的在线用户自动同步、从而给每个在线用户都带来大量的网络流量。此外，我们还观察了和Dropbox使用相似的同步机制（URS）的诸多云存储应用，比如iDriveSync、Ubuntu One、盛大网盘、360云盘等，结果发现它们都存在类似Dropbox的流量滥用问题。

为深入理解流量滥用问题，我们对Dropbox进行了广泛可控的测量。具体来说，我们人为生成一系列数据更新模式（通过改变文件更新频率和更新长度）来观察相应的同步网络流量。虽然Dropbox是一个闭源商业应用，并且它的通信数据包中绝大部分使用SSL协议加密了，但我们依然能够使用Wireshark网络协议分析器 [43]、通过“黑盒测量”来理解Dropbox的工作原理。通过分析Dropbox数据包的时序，我们量化地解释了为什么在面临频繁短促数据更新时、会话维护流量会远远超出实质数据流量。更为重要的是，我们发现了触发Dropbox URS同步机制的本地操作系统特性，从而将Dropbox同步的整个过程分解为一系列单独的步骤并分别加以探究。

在上述测量结果的指导下，我们设计了一个**“高效批同步算法”**来解决流量滥用问题，英文简称**UDS（Update-batched Delayed Synchronization）**。如图 2.1所示，UDS 相当于用户文件系统和云存储应用（比如Dropbox）之间的一个中间件，它独立于任何特定的云存储系统、并且不需要对原始云存储系统进行任何改变，从而使得UDS简单易用。具体地说，UDS首先构建一个名为“SavingBox”的文件夹以替代原始同步文件夹（的功能），一旦探测到频繁短促数据更新，就适当延迟数据更新的同步、最终将多次更新批处理成一次大的

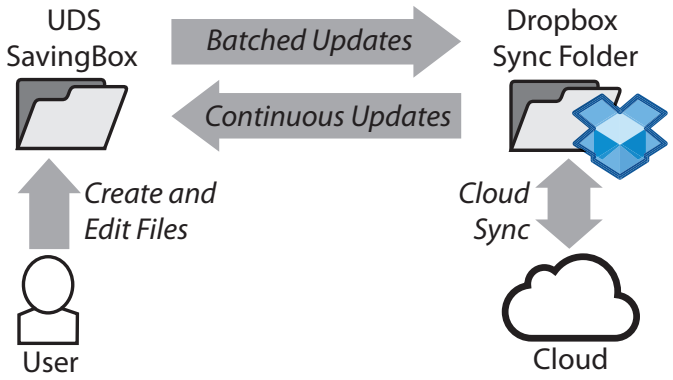


图 2.1　云存储服务的高效批同步算法（UDS）示意图

更新同步到云端。从本质上说，UDS强迫云存储应用合并了那些可能导致流量滥用问题的数据更新。在实际环境下，UDS批同步所导致的延迟仅为数秒，这就意味着用户不大可能感觉到，因此也就不会明显影响用户体验。

为评估UDS的实际性能，我们实现了Linux版本的UDS原型系统，它使用inotify [44]这一内核API来跟踪SavingBox文件夹内的文件更新、使用开源工具rsync [45]来计算被修改文件的二值差异。原型系统运行结果显示：UDS能将会话维护流量降低到实质数据流量的30%，作为对比，Dropbox（的URS机制）所产生的会话维护流量往往可以达到实质数据流量的数十倍。

除了流量滥用问题，我们还观察到URS和UDS都存在一个缺点：处理频繁数据更新时的CPU开销很高，因为两者都必须不断地重新索引被更新文件。从根源上看，重新索引操作之所以不断发生是因为inotify 只向上层应用汇报了磁盘上什么文件/目录被修改、但并没有汇报是怎么修改的——从而rsync（或一个等价的算法）就必须被不断地执行以确定文件是怎么修改的。为解决这一问题，我们修改了Linux 内核以返回文件更新的大小和位置，实际上，这些信息本来就存在于内核中，我们修改后的内核API只不过是将这些信息汇报给上层应用。利用修改后的内核API，我们实现了一个增强版的**UDS+**系统，它根本不需要调用rsync来确定文件是怎么修改的，所以其CPU开销远低于URS和UDS。

修改Linux内核具有一定风险性，我们也很清楚要说服Linux内核社区采纳新的API并不容易，但我们依然坚信对于inotify的修改是值得的。我们使用Linux的strace命令跟踪了一系列云存储应用所涉及的系统调用，可以确认诸多云存储应用（如Dropbox、Ubuntu One、TeamDrive、SpiderOak等）确实使用了inotify，因此，如果我们修改后的API能够真正融入到Linux内核中，将有众多

的云存储应用受惠于它。

2.1.2 相关工作综述

随着云存储服务的愈发流行，学术界相应的研究也越来越多。就我们所知，最先对云存储服务进行测量分析的是Hu等人，他们简单比较了4个应用：Dropbox、Mozy、CrashPlan和Carbonite的基本性能 [46]，主要目标是评估不同应用的相对上传/下载性能——他们发现Dropbox在4个应用中表现最好，而Mozy则最差。

已有数份研究专注于Dropbox。Drago等人通过在欧洲2所大学和2个居民区的大规模测量研究了Dropbox系统的细节架构、用户情况、工作性能等等 [41]。他们还公开了他们的测量数据集 [42]，这些数据集也为我们本文的工作提供了重要支持。Wang等人的研究 [47]显示Dropbox系统的可扩展性受限于其后台云存储Amazon S3与后台云计算Amazon EC2的互动关系，并提出一些建议性方案解决这个后台瓶颈问题。此外， [48, 49]研究了Dropbox的数据消重问题，而 [46, 50]讨论了Dropbox的数据安全和隐私问题。

作为Dropbox实际数据存储的后台，Amazon S3近年来也有一些量化的研究。Burgen等人从用户的角度测量了S3的性能 [51]，指出用户所感知到的性能主要取决于客户端和S3 之间的传输带宽、而不是S3云端的上传带宽。因此，云存储服务的设计者必须特别关注用户实际感知到的服务质量、而不是一味提升云端的规模和配置。Li等人开发了一个称为“CloudCmp”的工具 [3]来广泛地比较4个主流云存储系统：Amazon AWS [52]（包含Amazon S3），微软Azure [53]，谷歌AppEngine和Rackspace CloudServers的性能。他们发现不同的云存储系统性能迥异，特别是S3更适合处理大的数据对象而非小的数据对象，这和本文中我们观察到的现象是一致的。

基于两个大规模的NAS（Network-attached system）文件系统数据集（来自同一个企业数据中心），Chen等人对于数据访问模式进行了多个层面（包括用户层、应用层、文件层和目录层）的多维分析 [54]，从而得出12条“设计启发”，以指导“存储系统如何针对特定数据访问模式来定制”。Wallace等人也对一个大规模商业备份系统的备份数据流进行了广泛的研究 [55]。本文的工作和上述两份工作具有相似的研究方法论：首先研究云存储用户的数据访问模式，然后利用这些知识优化云存储系统以提升性能。

最后，还有更多相关于Dropbox类云存储系统的研究工作，比如云支撑的文件系统 [56,57]、流通知的差分压缩（stream-informed delta compression） [58]以及可靠的云存储设计 [59,60]。

2.1.3 深入理解云存储内容分发过程

本节我们概览云存储系统的架构并对云存储应用的网络开销进行细粒度测量。首先，我们审视Dropbox的高层系统架构，识别出客户端所访问的云端核心部件。通过在操作系统层面监控Dropbox客户端的活动，找出驱动网络流量的关键性本地操作。随后，我们通过一系列精心设计的测量精确地捕捉到（客户端）本地文件系统的活动和（云端）网络流量之间的关系。这些测量反映出导致云存储应用“同步流量滥用问题”的特定情境。虽然我们主要关注Dropbox（因为它是最早和最流行的云存储系统），但我们的测量发现和测量方法基本上也适用于其它云存储系统。

研究云存储应用的流量特征之前，首先必须理解云存储系统的高层系统架构。我们以Dropbox为代表来探索云存储系统是如何构建的。为了理解Dropbox应用如何作用于用户设备，我们在自己的测试计算机上安装了Dropbox客户端并使用Wireshark网络协议分析器记录了相关的通信数据包。我们观察到有相当一部分通信数据包被Dropbox加密了（使用HTTPS协议），但是我们并不试图去破解加密协议（破解HTTPS并非不可能）。反过来，我们的办法是去分析未加密的通信数据包、每个数据包的目的地、时序等等，这些信息组合到一起使得我们依然能够推断出Dropbox应用的高层操作，并且这个办法具有更高的普适性。

图2.2描绘了Dropbox系统的高层架构。客户端主要发送三种类型的消息到云端，相应地，云端主要由如下三种服务器构成。

(a) **索引服务器（Index server）**：每当用户启动Dropbox客户端、或者客户端监控到同步文件夹中发生数据更新的时候，客户端首先建立一条到云端索引服务器的TCP连接。虽然Dropbox租用了S3来存储实际文件内容，但它自身还构建了一个相对较小的“私有云”来部署索引服务器、维护文件索引信息。测量结果显示每个Dropbox用户被分配6-12台索引服务器，在每次会话中客户端仅连接到1-2台随机选择的索引服务器，从而大致维持索引服务器的负载均衡。

索引服务器主要的功能是认证用户和存储用户的索引（也称为“元数

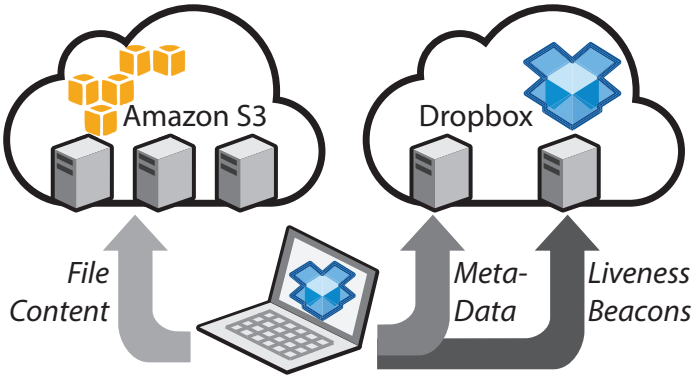


图 2.2　Dropbox系统架构示意图

据”）。元数据包括用户文件列表、文件大小和属性以及如何在Amazon S3上找到文件的实质内容。如果用户只是修改文件属性或者删除文件（而并没有修改文件内容），那么客户端只需要联系索引服务器来同步文件更新。

(b) **Amazon S3存储服务器**：上面已经说过，Dropbox租用了Amazon S3来存储实际文件内容（从而节省了大量的硬件投资、也使得Dropbox服务能够迅速扩展），索引服务器能够告知客户端如何从S3直接获取一个文件（使用S3的标准REST接口，REST即Representational State Transfer、表述性状态转移）。此外，任何文件在存储到S3之前客户端都会压缩它以节省空间（需要注意的是Dropbox分配给用户的空间“配额”是按照文件原始大小计算的），因此如果客户端从S3取回一个文件必然需要解压缩。类似地，如果一个文件被修改了，那么压缩后的二值差异会被客户端上传到S3。

(c) **信标服务器（Beacon server）**：每个Dropbox用户端都被分配一个固定的信标服务器，该服务器也位于Dropbox私有云中。客户端周期性地发送HTTP消息到信标服务器以汇报其在线状态，同时信标服务器也会推送通知消息到客户端，比如有一个共享文件在别的客户端发生了变化因此需要同步。当客户端空闲时，它大约每隔55秒联系一次信标服务器；否则，它就停止或者推迟联系信标服务器，因为它和索引服务器之间的交互已经暗含了信标的功能。

本地磁盘和网络间的关系：上文从网络协议的角度去理解Dropbox的行为，下面我们要从操作系统（更具体地是本地文件系统）的角度进行更深入的探索。为了测量Dropbox应用的细粒度行为，我们使用了一个称为“Dropbox命令行接口”[61]（Dropbox Command-line Interface、简写为Dropbox

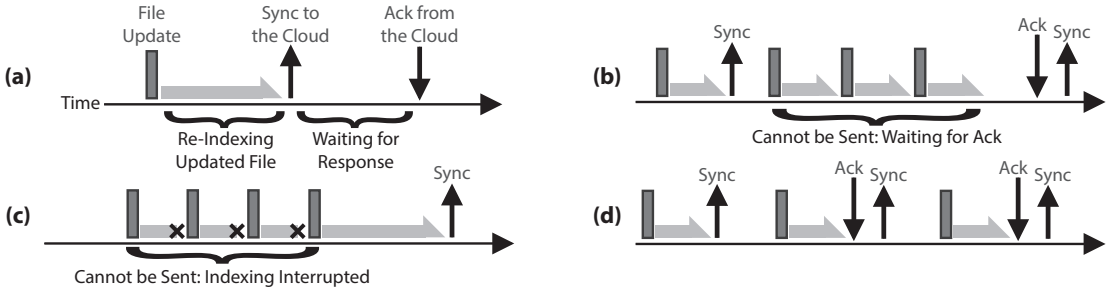


图 2.3　一次文件更新发生后Dropbox应用的底层行为。(a)描绘了Dropbox的最基本行为，(b)和(c)描绘了多个文件更新被“批处理”以后的情况，(d)则描绘了Dropbox流量滥用最严重的情形——没有任何两个文件更新被批处理。

CLI）的Python脚本，它能够监控Dropbox客户端的底层行为。有了Dropbox CLI，我们就可以编程序根据需要来查询Dropbox客户端的实时工作状态，从而领悟到Dropbox的URS同步机制的内在原理。

图2.3(a)描画了Dropbox（或者说URS机制）的最基本行为。首先，Dropbox同步文件夹中的一个数据更新引起本地磁盘（文件系统）发生变化，这可能是用户创建了一个新文件、或者是修改了一个现存文件。Dropbox客户端利用了特定操作系统的API来监控同步文件夹中的文件变化（比如在Linux操作系统中利用inotify [44]这一内核API），每当客户端监控到文件更新时，就立刻索引更新的新文件或者重新索引更新的旧文件。文件（重新）索引完成之后，客户端把被压缩的新文件或二值差异发送到Amazon S3，而对应的文件元数据则被发送到Dropbox 私有云。上述操作在图2.3(a)中被标示为“Sync to the Cloud”。文件更新被成功提交到云后，Dropbox私有云回馈给客户端一条确认消息（Ack from the Cloud），标志着该次文件同步（会话）的完成。下文中我们还会进一步研究上面整个过程中各个步骤的准确执行时间。

图2.3(a)描画的过程看上去简单、直接，但现实环境中存在诸多“隐藏”状况会令Dropbox同步过程复杂化。具体来说，并不是每一次文件更新都会触发一次（如图2.3(a)所描画的）文件同步过程：图2.3(b)和2.3(c)描绘了多个文件更新被“批处理”以后的情况。

先看图2.3(b)，在一次同步会话被触发之后（但尚未完成），文件又被修改了多次，每次修改都导致URS去触发一次新的同步会话，但没有一次同步会话

真正完成了，直到最后收到来自云端的确认消息。因此，图2.3(b)中的多次文件修改被“批处理”成一次大的文件修改了。

再看图2.3(c)，文件被多次修改、且修改速度太快以致超出了客户端重新索引文件的速度，所以URS根本不会触发任何同步会话，直到文件修改的速度降低到不超出客户端重新索引文件的速度。同样，图2.3(c) 中的多次文件修改也被“批处理”成一次大的文件修改了。

图2.3(b)和2.3(c)反映出本地磁盘活动和Dropbox网络流量之间存在复杂的互动关系。一方面，一系列具有特定时序的文件修改操作可能仅触发一次文件同步（被“批处理”了），如果它们发生的足够快以致能够“打断”客户端的重新索引操作；另一方面，一系列时序“糟糕”的文件修改操作可能触发多次文件同步，导致Dropbox“流量滥用问题”，如图2.3(d)所示。并且这种“糟糕”的文件更新模式并不是某种特例或边缘情形，在下文中我们将看到现实世界中确实存在诸多这样的“槽糕”文件更新模式。

上面我们已经明白本地磁盘活动和Dropbox网络流量之间存在复杂的互动关系，下面通过精心设计的可控场景测量进一步深入探讨这种关系，目标是量化文件更新频率、文件更新大小及Dropbox网络流量之间的关系。所有的测量都在两台计算机上完成：第一台是普通笔记本电脑，双核Intel处理器@2.26 GHz，2 GB内存，5400 RPM（Round Per Minute）转速及250 GB容量的SATA 硬盘；第二台是普通台式机，双核Intel处理器@3.0 GHz，4 GB内存，7200 RPM转速及1 TB容量的SATA硬盘。在上述各种配置中，硬盘转速尤其重要，因为它对于Dropbox客户端（重新）索引文件所需的时间有关键影响。两台计算机都运行Ubuntu Linux 12.04版操作系统、Linux版的Dropbox客户端（0.7.1版） [62]以及Dropbox CLI工具。两台计算机的互联网接入带宽都在4 Mbps左右，具有当前的代表性（带宽），且所有测量都于2012年在美国完成（注：本文第一作者于2011年9月至2012年9月期间在美国明尼苏达大学双城校区公派留学）。

(a) 场景一：创建新文件

场景一的实验检查当我们在Dropbox同步文件夹中创建新文件时所产生的网络流量。表 2.1记录了当我们在Dropbox同步文件夹中创建不同大小的新文件时、Dropbox客户端向索引服务器以及向Amazon S3发送的网络流量（对应5400

表 2.1　在Dropbox同步文件夹中创建新文件所产生的网络流量

新文件大小	索引服务器流量	S3存储服务器流量	α	同步时延
1 B	29.8 KB	6.5 KB	38200	4.0 秒
1 KB	31.3 KB	6.8 KB	40.1	4.0 秒
10 KB	31.8 KB	13.9 KB	4.63	4.1 秒
100 KB	32.3 KB	118.7 KB	1.528	4.8 秒
1 MB	35.3 KB	1.2 MB	1.22	9.2 秒
10 MB	35.1 KB	11.5 MB	1.149	54.7 秒
100 MB	38.5 KB	112.6 MB	1.1266	496.3 秒

RPM硬盘的计算机）。为便于有用数据更新流量的计算，我们使用JPEG格式的图像文件进行实验（不包括最小的1 B新文件实验），因为JPEG是一种压缩文件格式、从而Dropbox客户端不能进一步压缩它。

表 2.1显示出关于Dropbox同步流量的几个有趣的现象。首先，Dropbox客户端向索引服务器发送的元数据大小十分稳定、几乎与新文件大小无关。反过来，Dropbox客户端向Amazon S3发送的数据流量却十分接近新文件大小。之所以出现这一结果，是因为在Dropbox系统中真正的文件数据内容是存储在Amazon S3上的。

表 2.1中的α列记录了总的Dropbox流量和新文件大小的比值。最理想的情况是α接近1，因为这表示Dropbox数据同步的额外流量开销极低。对于小文件来说α很大，因为元数据的固定大小远高于文件大小。而对于大文件来说α则更为合理，因为Dropbox数据同步的额外流量被文件大小“分摊”了。

表 2.1的最后一列记录了Dropbox云同步的平均时间。这些实验显示出云同步所需时间至少4秒、与新文件大小无关。这一最小时间值取决于Dropbox的云系统架构，并且它和硬盘转速、互联网连接带宽、RTT（Round Trip Time，即往返时间）等无关。对于更大的文件，同步时延相应增加；在这些情形下，同步时延主要取决于Dropbox客户端向Amazon S3上传文件所需的时间。

(b) 场景二：短促文件更新

场景二的实验检查Dropbox处理（对一个现存文件进行）短促文件更新时的行为。每次实验首先在Dropbox同步文件夹中创建一个空文件，然后周期性地向此文件添加1个随机字符，直到文件大小达到1 KB。添加随机字符是为了确保Dropbox客户端不能压缩文件的二值差异。

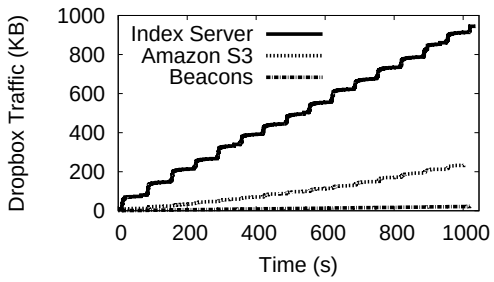


图 2.4　在5400 RPM计算机上向一个文件频繁添加1字节数据时Dropbox的同步流量

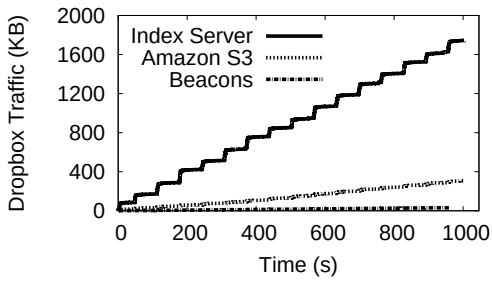


图 2.5　在7200 RPM计算机上向一个文件频繁添加1字节数据时Dropbox的同步流量

图 2.4和 2.5描绘出Dropbox的同步流量。虽然每次只添加了一个字节的数据，且文件总大小始终低于1 KB，但Dropbox所消耗的同步流量对于5400 RPM的计算机和7200 RPM的计算机分别达到了1.2 MB和2 MB。Dropbox主要的同步流量是由于客户端向索引服务器传送了大量的元数据更新。如表 2.1所示，每次索引服务器更新大约消耗30 KB流量，这远远超过被更新的文件大小。此外，客户端向Amazon S3存储服务器发送的数据流量相比于文件大小依然多很多。信标流量则是可以忽略的，因为信标消息仅仅在Dropbox客户端空闲时才会周期性发送，但在我们的实验中Dropbox客户端几乎从未空闲。总的来说，图 2.4和 2.5清晰地展现出：在极端情况下，Dropbox所产生的数据同步流量可以远远超出有用的用户数据更新大小（好几个数量级），并且更快的7200 RPM硬盘进一步加剧了这一“流量滥用问题”、使情况变得更糟。

文件更新的时序。 如图 2.3(b)和(c)所示，文件更新的时序（情况）确实可以影响到Dropbox的网络流量使用。为了检查更新时序和同步流量之间的关系，我们对上面的实验方法稍作改动、即每次实验采用一个不同的数据添加时间间隔：从100毫秒到10秒。图 2.6和 2.7分别描画了Dropbox针对5400 RPM计算机和7200 RPM计算机的同步流量，从而我们可以看到一个清晰的趋势：文件更新越快、同步流量越少。这可以从图 2.3(b)和(c)所描画的机制得到解释，也就是说，Dropbox客户端软件能够（客观上、无意识地）“批处理”连续快速的数据更新。这一“批处理”行为减少了向索引服务器发送的元数据信息流量，并且“聚合”了往文件中添加的多个字节数据成为一个二值差异、然后发送此二值差异到Amazon S3中。不幸的是，随着数据添加的时间间隔逐渐

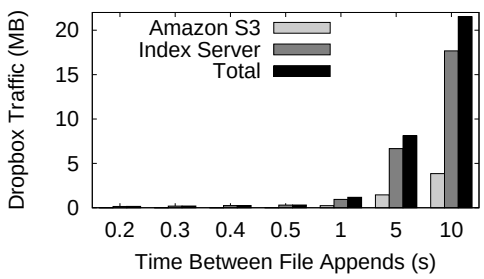


图 2.6 在5400 RPM计算机上向一个文件频繁添加1字节数据时Dropbox的同步流量

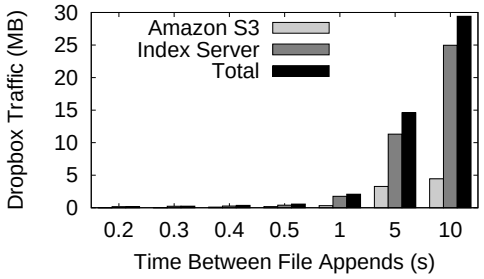


图 2.7 在7200 RPM计算机上向一个文件频繁添加1字节数据时Dropbox的同步流量

延长，Dropbox的“批处理”效果越来越弱，这在图 2.6和 2.7中所显示的5 秒和10秒间隔实验中表现得尤为明显－－这代表了图 2.3(d)中所描画的极端场景：几乎每字节的数据更新都触发了一次完整的数据同步。

(c) 文件索引时间探究

图 2.6和 2.7证明了文件更新的时序（情况）确实可以影响Dropbox同步流量。虽然如此，到目前为止我们还是不知道具体哪个因素应该（真正）对同步流量负责：到底是图 2.3(b)中所显示的网络等待间隔时间，还是图 2.3(c)中所显示的被中断的文件索引操作，或者是这两个因素的某种组合？

为回答上述问题，我们进行了微基准测试（microbenchmarks）来检查Dropbox客户端到底花多少时间来索引文件。和前文类似，我们还是使用一个空文件、往其中周期性随机添加1字节数据、直到文件大小达到1 KB。数据添加的间隔时间设定为5秒，以保证（间隔时间足够长从而）文件索引操作不会被打断（如图 2.3(c)所示）。利用Dropbox CLI（Command Line Interface [61]）工具监控Dropbox客户端的工作过程，我们可以测量出Dropbox客户端花在文件索引上的时间。

图 2.8描画了Dropbox文件索引时间的分布。5400 RPM计算机的中值索引时间大约为400毫秒，而7200 RPM计算机的中值索引时间大约为200毫秒，在所有测试中观察到的最长索引时间为960毫秒。这一结果表明：发生在200-400毫秒之间的顺序文件更新（取决于具体硬盘速度）应该会打断Dropbox的文件索引过程、导致其重启，因此这类文件更新被Dropbox“批处理”了。

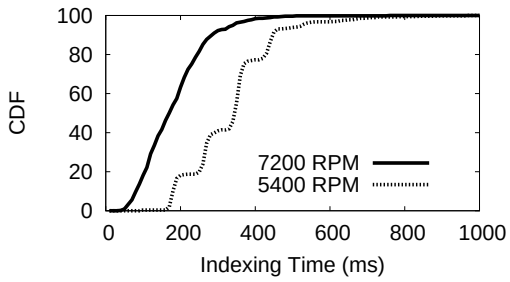


图 2.8　Dropbox文件索引时间分布（文件大小为1 KB）

将图 2.6和 2.7同图 2.8相比较，可以确信文件索引操作被打断确实减少了Dropbox的同步流量。随着数据添加间隔时间从200毫秒延长到500毫秒，Dropbox所产生的同步流量稳定增加，这同顺序的文件添加操作打断文件索引过程的概率紧密对应，如图 2.8所示。当数据添加间隔时间达到1秒时，顺序的文件添加操作几乎不会打断文件索引过程（因为我们已经观察到最长索引时间为960毫秒）。因此，1秒间隔测试所产生的同步流量比500毫秒间隔测试所产生的同步流量多一倍以上。

虽然文件索引被打断是导致Dropbox在处理频繁短促数据更新时所展现出的同步流量模式的主要原因，但这并不足以解释：为什么当数据添加间隔时间从1秒延长到5秒时，同步流量显著增加？事实上，当数据添加间隔时间为5秒时，真正影响同步流量的因素转变为图 2.3(b)中所描画的网络时延。如图 2.8所示，三分之一的Dropbox同步操作可以在1-4秒内完成，还有三分之一则在4-7秒内完成，所以延长数据添加间隔时间（从1秒到10秒）导致更多的文件更新操作可以（成功地）触发Dropbox数据同步操作，也就是说，Dropbox的“批处理”效果被极大地减弱了。

(d) 场景三：长文件更新

到目前为止，我们所有的实验都集中在频繁“短促”数据更新上。下面我们来测量Dropbox处理长文件更新时的行为。和前文类似，我们往一个空文件中逐秒添加随机数据块，直到文件大小达到5 MB为止。数据块大小介于50-100 KB之间，每隔10 KB做一次测量。

图 2.9描画了在5400 RPM计算机上的实验结果。和上文中单字节添加的实验结果不同，Dropbox在处理长文件更新时的同步流量仅仅略高于文件大小（5

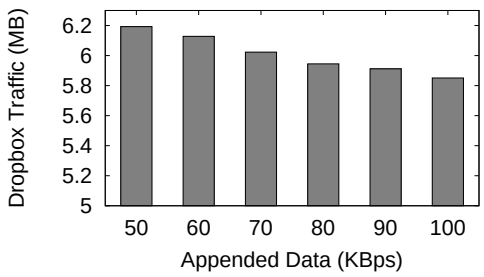


图 2.9 每秒添加的数据块大小逐渐增加时所对应的Dropbox同步流量

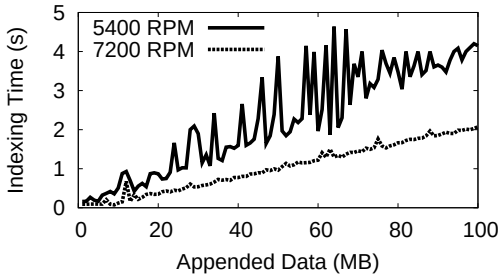


图 2.10 文件大小逐渐增加时所对应的Dropbox文件索引时间

MB）。随着每秒添加的数据块大小的增加，同步流量同文件大小之间的比值逐渐减小，这再次映证了我们前文提出的观点：Dropbox在处理大文件更新时更能有效地利用网络资源。

图 2.10探索了添加数据大小（或文件大小）和Dropbox索引文件时间之间的关系。这两个变量之间呈现明显的指数关系：随着文件大小逐渐增加，对应的Dropbox文件索引时间（从平均意义上讲）也相应延长。这一结果可以很直观地解释：Dropbox客户端需要花费更多的时间来读取更大的文件、并对文件执行二值差分算法。

图 2.10还反映出：对于更大的文件、文件索引操作将更容易被打断，由于大文件的索引一般需要更多的时间。虽然如此，这一效果对于较差的计算机（特别是硬盘转速慢的）将被表现得更为明显。因此，当处理数MB大小的文件时Dropbox可以更有效地利用网络资源。类似地，同索引服务器通信的（较为）固定的网络开销对于大文件来说也更容易分摊，这同样有助于提升Dropbox的网络流量效率（α）。

其它的云存储服务和操作系统。这一小节我们简单地考察其它的云存储服务，以确定Dropbox的“流量滥用问题”是否对它们依然存在。此外，由于一个云存储服务常常为多个操作系统开发不同的客户端软件，为了检查它们跨操作系统的情况，我们所有的测量都在下面两台计算机上完成：第一台是台式机，双核Intel处理器@3.0 GHz、4 GB内存、7200 RPM和1 TB的硬盘，安装有Ubuntu Linux（12.04版）和Windows 7 （SP1 版）操作系统；第二台是MacBook Air笔记本电脑，四核Intel处理器@1.7 GHz、4 GB内存、256 GB的SSD硬盘，仅安

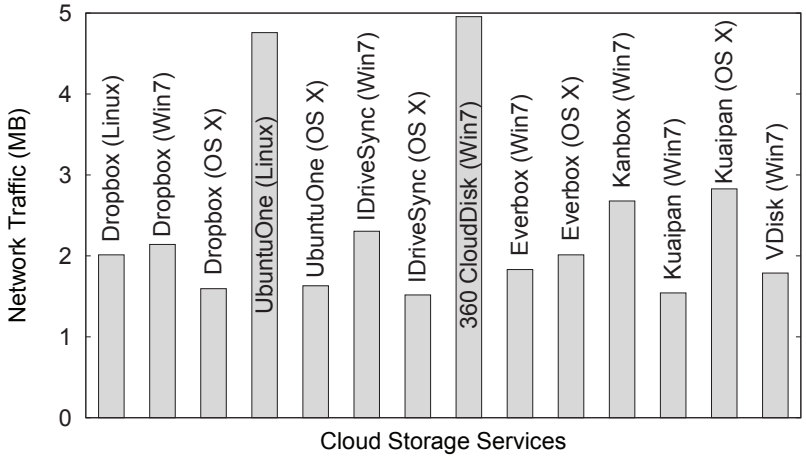


图 2.11　向一个空文件逐秒添加单字节直到1 KB后、多个不同的云存储服务各自产生的同步流量

装Mac OS X（Lion 10.7）操作系统。每台计算机的互联网接入带宽都是4 Mbps。

如图 2.11所示，在Dropbox之外我们选择了七个其它的在美国或中国流行的云存储服务，并针对每个云存储服务、在上面一段所述两台实验计算机上重新运行了单字节添加实验（共添加1 KB）。由于360 CloudDisk（360云盘）、Everbox（盛大网盘）、Kanbox（酷盘）、Kuaipan（金山快盘）和VDisk（新浪微盘）都主要服务国内用户，与它们对应的实验是在中国而不是美国完成。另外需要注意的是：有些云存储服务仅开发了其Windows版本的客户端软件。图 2.11显示：所有七个云存储服务同Dropbox类似、也产生了数MB的同步流量，远远超出文件大小1 KB，由此映证了“流量滥用问题”对诸多云存储服务依然存在、且与操作系统关系不大。

测量小结。下面我们简单总结在整个这一节中的观察和发现：

- Dropbox客户端只在满足两个前提条件的情况下才会启动数据同步过程：（1）更新后的文件在本地已经重新建立索引；（2）前一次数据同步过程已经完成。这就导致存在两种情况能够在“客观上”批处理多个文件更新，从而减少Dropbox同步流量。
- Dropbox索引一个小文件一般需要200-400毫秒，且具体耗时与本地计算机硬件配置、特别是硬盘转速相关，所以发生在200-400毫秒之间的频繁文

件更新会打断Dropbox的文件索引过程、导致其重启，从而这类文件更新被“批处理”了。

- Dropbox同步一个小文件总共大约需要4秒，所以间隔短于4秒的文件更新操作也有可能被“批处理”，因为后一次操作需要等待前一次操作（的同步过程）完成。
- “流量滥用问题”一般出现在频繁短促数据更新发生的时候，且连续的数据更新间隔一般为数秒。在这些情况下，云存储应用不能有效地“批处理”数据更新，导致数据同步流量数十倍甚至数百倍于实质的文件更新大小。
- 虽然我们绝大多数实验专注于Dropbox，但在其它多个流行云存储服务上的实验表明“流量滥用问题”具有很强的普适性、且与操作系统关系不大。

2.1.4 现实中的同步流量滥用问题

前面一节的测量结果表明：在受控实验环境下，云存储应用确实产生了大量的、远超过用户实质更新数据量的同步流量。然而到目前为止，我们还是不清楚上节的受控实验环境在现实世界中是否存在，这就导致一个新问题：现实用户真的会受“同步流量滥用问题”影响吗？这一节我们通过测量现实世界中Dropbox的同步流量回答此问题。我们进行了四个不同的模拟实验：HTTP文件下载、日志文件事件添加、数据库元组写入和协同文档编辑。在可能的情况下，我们尽量使用真实用户行为数据来驱动模拟实验。每个实验相关的文件都放在Dropbox同步文件夹中，这意味着所有文件更新（最终）都被反馈到云端。最终，每个实验所产生的Dropbox同步流量都远超过被更新文件大小、一般在11到130倍之间，从而进一步确认了“同步流量滥用问题”的现实存在性。

HTTP文件下载。Dropbox的一个重要应用场景是文件共享，比如说，本文的几位作者间就是通过Dropbox来协同编辑源文件的。在诸多情况下，我们发现从私人数据池（如Gmail邮箱、个人Linux服务器）直接下载文件到Dropbox同步文件夹很方便同其他作者共享。那么当一个文件从网上直接下载到Dropbox同步文件夹的过程中（在网络通信方面）发生了什么呢？从理论上讲答案非常简单：获取整个文件、一次写入整个磁盘、然后Dropbox再将整个文件上传到云端——但实际情况绝非如此：由于文件是分片从网上传过来的，所以是逐块写入磁盘

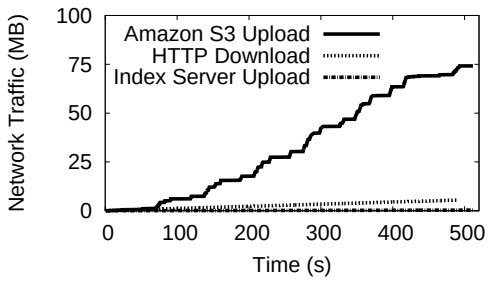


图 2.12 以HTTP方式下载一个5 MB的文件到Dropbox同步文件夹，所产生的同步流量

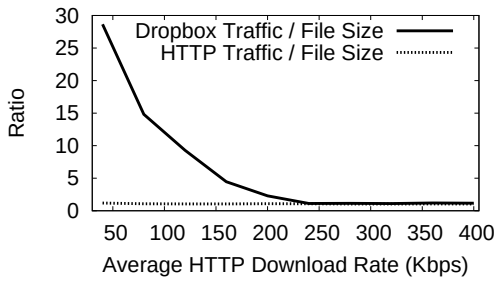


图 2.13 Dropbox同步流量以及HTTP下载流量和下载文件大小的比值

的，就好像前文所做的文件添加一样。那么，Dropbox对文件下载到底有什么样的反应呢？

为回答上述问题，我们做了一个简单的实验：使用标准的Linux命令wget从网上直接下载一个5 MB的压缩文件到Dropbox同步文件夹。实验设备还是2.1.3节提到的那台Linux笔记本电脑，所有进出的网络流量（数据包）都用Wireshark来捕捉。为便于计算实质的数据上传流量，我们使用压缩文件来实验，这样Dropbox在同步文件时就很难再压缩了。

图 2.12画出了相关于HTTP下载、Dropbox索引服务器以及Amazon S3的网络流量。在这一实验中，我们固定wget的下载速率为80 Kbps。很明显，Dropbox同步流量远超过HTTP下载流量：wget产生5.5 MB的流量（5 MB的文件大小+ HTTP数据包额外开销）,但Dropbox同步流量却高达75 MB。

图 2.12和图 2.4展现出非常相似的结果：在这两个实验中，Dropbox都传输了超出实质数据量一个数量级（即10倍以上）的流量。不同的是，在HTTP 下载实验中，额外开销主要来自客户端向Amazon S3传输数据；而在图 2.4对应的实验中，额外开销主要来自客户端向Dropbox索引服务器传输元数据。之所以有此区别，原因就在于HTTP下载文件大小达到5 MB，因为每个文件分块都比向索引服务器传输元数据的流量（约30 KB）要大；相反，在另一个实验中，整个文件大小才1 KB。

下面我们检查对应不同的HTTP下载数据率的Dropbox软件行为。图 2.13列出了Dropbox同步流量以及HTTP下载流量和下载文件大小的比值。HTTP下载流量和下载文件大小的比值恒定为1.1，而Dropbox同步流量和下载文件大小的比

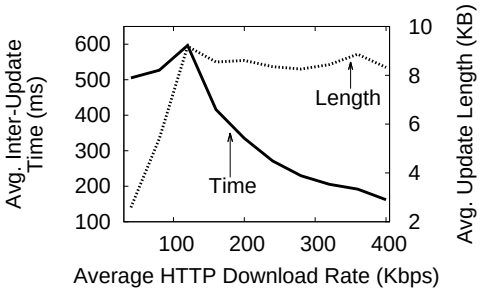


图 2.14 对应不同的HTTP下载数据率的（本地磁盘数据）平均更新间隔时间和平均更新大小

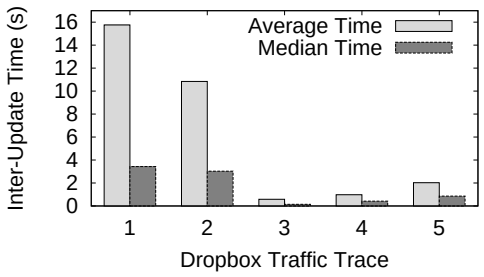


图 2.15 对应不同日志记录的（本地磁盘数据）平均/中值更新间隔时间

值介于1.1到30之间、且HTTP下载速率越高此比值越小。

为了解释为什么Dropbox同步流量随着HTTP下载速率的升高而减少，我们检查了wget和硬盘之间的互动情况，在图 2.14中记录了对应不同的HTTP下载数据率的（本地磁盘数据）平均更新间隔时间（对应左边的纵坐标和图中的实线）和平均更新大小（对应右边的纵坐标和图中的虚线）。HTTP下载数据率达到200 Kbps时，Dropbox同步流量降到最低，此时数据更新每隔300毫秒写入一次磁盘，而数据更新大小则达到了最大（每次更新约9 KB），所以Dropbox客户端能够（客观上）“批处理”多个数据更新。相反，当HTTP下载速率低的时候，数据更新间隔时间较长，而数据更新大小较小，所以Dropbox“批处理”的机会不多，导致出现流量滥用问题。

除了wget下载，我们还在Chrome和Firefox浏览器上做了HTTP下载实验并得到了相似的结果：当HTTP下载速率较低时，Dropbox依然产生远超出下载文件大小的同步流量。

日志文件事件添加。考虑这样一种应用场景：用户想要往云中持续地备份一个日志文件。在此场景下，每次事件发生后向日志中添加的数据量基本不变，而变化的是事件添加的速率。问题在于：云存储用户添加事件的速率多快才合适？

为进行日志添加实验，我们使用了最近欧洲学者的一份大规模Dropbox测量报告 [41]中所公布的数据。这些数据是在ISP层收集到的，包括带有时间戳的（用户）IP地址和Dropbox 云服务器之间的互动记录。具体来说，

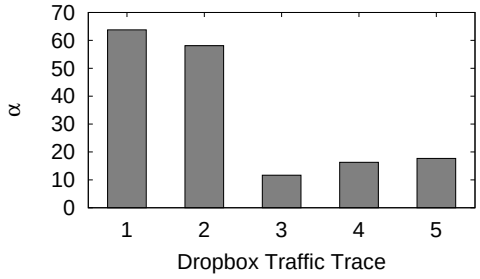


图 2.16 对应不同日志记录的Dropbox同步流量与实际文件大小的比值（α）

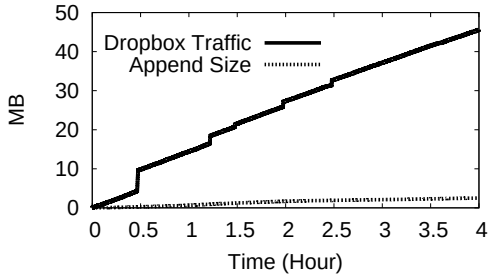


图 2.17 对应一个活跃用户日志的Dropbox同步流量与日志大小

我们使用了下面5份数据集：（1）*campus1_dataset1*、（2）*campus1_dataset2*、（3）*campus2_dropbox*、（4）*home1_dropbox* 和（5）*home2_dropbox* [42]。每份数据集包含16万到200万条事件，来自283到6609个IP地址——可以近似认为每个IP地址对应一个Dropbox用户，虽然某些情况下多个用户可能隐藏在一个NAT（Network Address Translator）之后。每条事件记录包含了105个属性，在磁盘上大概占据0.5 KB的空间。

图 2.15记录了5个日志数据集的平均/中值更新间隔时间。可以看出平均间隔时间远长于中值间隔时间，因为在每个日志中都存在一些“平静期”（比如夜里用户基本都睡觉了）、其间更新间隔时间可能长达数千秒。

在第一个实验中，我们首先在Dropbox同步文件夹里创建了一个空日志文件，然后“重放”日志数据集中所记录的事件——也就是把事件记录添加到日志文件，而事件时间戳则使得我们可以保证添加的速率和原来一致。图 2.16记录了对应不同日志记录的Dropbox同步流量与实际文件大小的比值（α），它介于11到63之间，反映出日志文件事件添加的场景同样存在流量滥用问题。

在第二个实验中，我们专注于*campus2_dropbox*数据集中一个特定的活跃用户，目标是检查单个用户的日志时间添加是否也存在流量滥用问题。实验重放了此用户使用Dropbox 4个小时的事件，平均间隔时间为2.6秒。图 2.17记录了Dropbox同步流量与日志大小，前者是后者的21倍，表明活跃的Dropbox用户同样存在流量滥用问题。

数据库元组写入。此实验研究这样一种应用场景：用户想要往云中持续

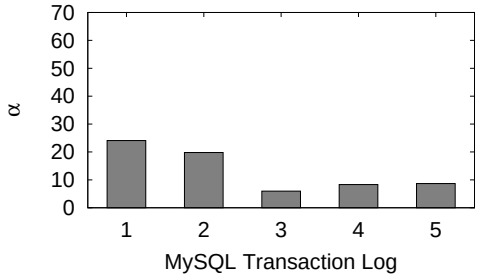


图 2.18　对应不同MySQL数据库的Dropbox同步流量与实际文件大小的比值

地备份一个数据库文件。我们安装了MySQL软件，并将其数据库文件配置到Dropbox同步文件夹中，并再次使用上文所说的Dropbox事件数据集 [42]来驱动整个实验过程。对数据库来说，每条事件代表一个元组，而事件的105个属性则对应数据库元组的属性，并且我们还为每个元组新增了2个属性：一个主键、一个次主键，共计107个属性。图 2.18描画了对应不同MySQL数据库的Dropbox同步流量与实际文件大小的比值（α），α介于5到25之间。虽然α比图 2.16中对应的值要小，但仍然数倍于实际文件大小。

协同文档编辑。Dropbox的一个常见应用场景是多个作者（并行）协同编辑一个文档，比如说本文的几位作者正是使用Dropbox来协同编辑该论文的。在这一小节，我们模拟了一个较为典型的协同编辑案例：首先，在Dropbox同步文件夹中创建一个1 MB的文件、其中充满随机ASCII字符；然后把此文件分享给第二个Dropbox用户；每个用户每隔t秒在文件位置x修改或添加l个随机字符，这里l是1到10之间的随机整数，而t是0到10之间的随机浮点数。如果用户执行的是修改操作，x是介于文件头和文件尾之间的一个随机位置；但如果用户执行的是添加操作，x就直接设定为文件尾；此外，用户执行文件修改和添加操作的概率均等。上述协同文档编辑实验共持续1小时，到实验结束时，Dropbox共产生接近130 MB的同步流量，高出文档大小上百倍。

2.1.5 UDS高效批同步算法

为克服云存储服务的流量滥用问题，我们实现了一种应用层机制以极大地减少云存储应用的同步流量，该机制被称为“高效批同步算法”（*update-batched delayed synchronization*，简写为UDS）。UDS的宏观设计如图2.1所示，直观上看，UDS被实现为普通云同步文件夹（如Dropbox同步文件

夹）的一种替代，它前动地探测和批处理发生在“SavingBox”文件夹中的频繁短促数据更新。这些被“批处理”的数据更新被合并到普通云同步文件夹、再被真正同步到云中。因此，UDS是作为一个中间件来保护云存储应用免受流量滥用问题影响的。

UDS的实现。UDS的设计受两个目标驱动：首先，它应该通过强制云存储应用批处理文件更新来解决流量滥用问题；其次，它应该兼容多个云存储服务。这第二个目标排除了直接修改现存云存储应用的办法，也不允许为特定的云存储服务重写一个客户端。

为满足这两个目标，我们将UDS实现为一个中间件，它位于用户和云存储应用（客户端）之间。从用户的角度看，UDS和现存的普通云存储服务一样，它首先在本地硬盘上创建一个名为“SavingBox”的文件夹，然后一直监控SavingBox中的文件和文件夹。当用户向SavingBox中添加新文件时，UDS自动计算新文件的压缩版本；相似地，如果一个文件被修改了，UDS则为此文件的改变部分计算一个压缩版本。对于SavingBox来说，如果t秒内不发生文件更新、或者总的文件更新大小超过一个阈值c，那么UDS就将数据更新推送到真正的云存储同步文件夹中（比如Dropbox同步文件夹）。此时，真正的云存储应用（比如Dropbox）自然会将被更新的文件同步到云端。

为验证想法，我们实现了Linux版的UDS系统，并将它同Linux版的Dropbox进行对比。UDS的实现使用了Linux内核提供的inotify系统调用来监控SavingBox文件夹中发生的任何数据更新。具体来说，UDS使用inotify_add_watch(...)函数来建立一个回调功能（callback），每当SavingBox文件夹中发生数据更新时、Linux内核就会激发此回调功能，将一系列信息，比如文件事件类型（如文件创建、文件修改等）、被更新文件路径告知UDS。根据这些信息，如果是新文件创建，那么UDS使用gzip计算出新文件的压缩大小；如果是现存文件被修改，UDS首先使用标准的Linux工具rsync来计算文件的二值差异、再用gzip计算出二值差异的压缩大小。UDS每隔一定时间就会推送SavingBox中的新文件和被更新文件到真正的云存储同步文件夹中。对于新文件，UDS将它（们）直接复制过去；而对于被更新文件，UDS使用rsync工具将其差分同步过去。

UDS内部需要维护两个变量，它们决定了多长时间推送（一次）SavingBox中的新文件和被更新文件到真正的云存储同步文件夹。直观

上看，这两个变量控制了数据更新“批处理”的频率。第一个变量是一个简单的计时器，每当一个文件被创建或修改时计时器值被重置为0，当计时器值增加到t时, SavingBox中的新文件和被更新文件就被推送到真正的云存储同步文件夹中。第二个变量是一个（字节）计数器，用来确保频繁短促文件更新被成批同步的最小容量。具体来说，UDS维护SavingBox中尚未被推送到真正的云存储同步文件夹中的所有数据更新的（压缩后）大小。如果计数值超过c，那么SavingBox中的新文件和被更新文件就立刻被推送到真正的云存储同步文件夹。需要注意的是，并非所有云存储应用都使用gzip来压缩文件，所以UDS的计数值只是一个近似的估计值，它可能不太准确，但在实践中的性能已经足够好了。

作为一项保险措施，UDS还使用了第二个计数器，它在更粗的时间间隔上推送SavingBox中的数据更新到真正的云存储同步文件夹。这项保险措施是为了防范极端“病态”的数据更新模式可能为“阻塞”UDS的数据同步。考虑这样一种极端情况（虽然在实际中极难出现）：往SavingBox中的一个文件反复添加单字节，此时如果c较大，那么需要太长的时间才能使数据更新值达到c；类似地，如果数据添加的时间间隔总是比t短，那么前文所说的（第一个）计时器在计数值没达到c前就不会有任何效果。有了这个作为保险措施的计时器（我们目前的UDS实现将它设置为30秒），极端“病态”的数据更新模式也不可能长时间阻塞UDS的数据同步。

UDS的参数配置。UDS有两个重要参数：计时器阈值t和计数器阈值c，它们代表了UDS同步流量和同步时延之间的权衡。一方面，如果我们将t和c都设置得很小，那么UDS向云端同步数据就会很勤，同步时延短了，代价则是同步流量增多；反过来，如果我们将t和c都设置得很大，那么UDS批处理的效果会显著增强，同步流量少了，但更长的同步时延可能会影响用户体验。我们真正想要的是在同步流量和同步时延之间取得一个好的权衡，为此我们做了一个实验：往SavingBox中一个空文件反复添加随机字符串，直到文件大小达到5 MB，在此过程中记录Dropbox的同步流量和同步时延。这个实验被重复多次，且每次采用不同的计数器阈值c。

图 2.19和图 2.20记录了多次实验的结果。意料之中，对于较小的c，UDS同步流量较大而同步时延较低。真正有趣的是图 2.19中同步流量快速降低然后趋于稳定的那个“拐点”：c = 250 KB；过了这个拐点，再增大c的节流效果已十

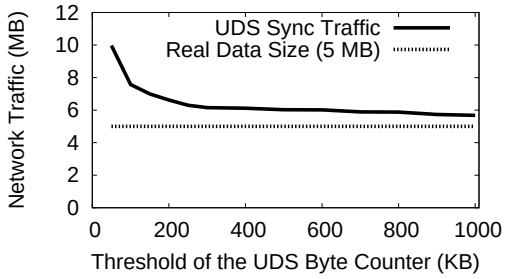


图 2.19　对应不同的UDS算法计数器阈值c的Dropbox同步流量

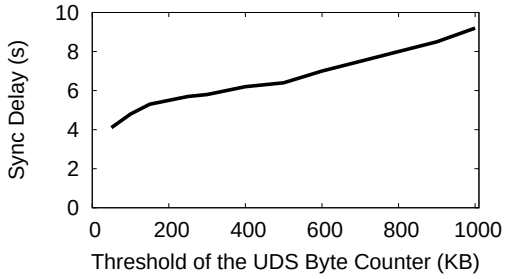


图 2.20　对应不同的UDS算法计数器阈值c的Dropbox同步时延

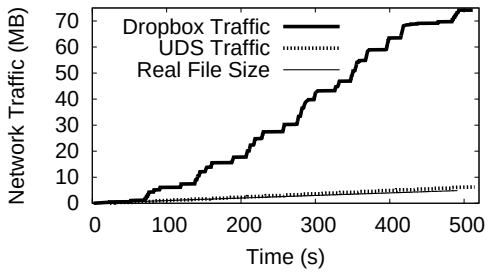


图 2.21　以HTTP方式下载5 MB的文件到Dropbox，Dropbox和UDS所产生的同步流量对比

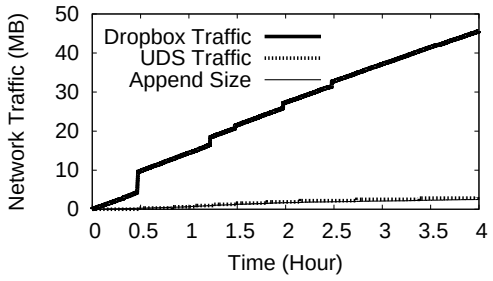


图 2.22　对应于一个活跃用户的日志事件添加过程，Dropbox和UDS所产生的同步流量对比

分微弱。另一方面，图 2.20显示c和同步时延之间呈现近似的线性关系，因此就同步时延来说不存在c的“拐点”。综上所述，我们最后选定c = 250 KB。

计时器阈值t则被设置为5秒，这主要来自于网络性能和用户体验之间的一种“定性”权衡（因为对用户体验的“定量”评估十分困难）。当t较大时，数据更新的批处理效果更明显，负面效果则是用户同步时延变长，部分用户可能产生不适感。我们个人的感觉是：t=5秒基本上可以有效缓解流量滥用问题，而对用户来说这个同步时延又不至于引起反感。

UDS和Dropbox性能对比。为了对比UDS和Dropbox的实际性能，我们重新运行了 2.1.4节的两个实验：（1）wget文件下载实验和（2）一个活跃用户的日志事件添加实验。图 2.21描画了两者的同步流量，很明显UDS要比普通的Dropbox节流得多、其流量（约6.2 MB）仅仅略多于下载文件大小（5 MB）。相似地，

图 2.22中UDS的同步流量略多于日志文件被添加的事件大小，远少于普通的Dropbox节流。这些结果清晰地证明了UDS的实用性。

2.1.6 UDS+：修改Linux内核

上一节我们描述了UDS中间件如何有效地减少了云存储应用的同步流量、解决了“流量滥用问题”。这一节我们将研究再深入一步，进一步分析云存储应用的CPU开销。我们首先分析Dropbox的CPU 开销、指出Dropbox消耗了大量CPU资源来索引被更新文件；反过来，UDS极大地减少了云存储应用的CPU开销，但这依然还不足够好。所以我们通过修改Linux内核相关的系统调用来进一步降低UDS的CPU开销，改进后的中间件称为UDS+。

Dropbox和UDS的CPU开销。我们首先评估Dropbox客户端本身的CPU开销（不使用UDS），实验场景和 2.1.3节中的类似：实验计算机为一台普通笔记本电脑，配置双核Intel CPU@2.26 GHz、2 GB内存以及一块5400 转速、250 GB容量的硬盘；首先在Dropbox同步文件夹中新建一个空文件，然后每隔200毫秒向其中添加2K个随机字符，整个过程持续1000秒（因此文件最终大小为10 MB），我们逐秒记录Dropbox进程的CPU开销。

由于Dropbox客户端是单线程的，所以它仅使用了实验计算机双核中的一核。图 2.23描画了Dropbox的CPU开销（随时间的变化），我们主要有两个发现。首先，Dropbox的CPU 利用率在400秒（对应文件大小为4 MB）和800秒（对应文件大小为8 MB）处有两次大的“跳跃”，其原因在于Dropbox客户端在索引文件时按4 MB分块 [50]。其次，Dropbox在整个过程中的平均CPU开销为54%，这实际上是非常高的。800秒以后，Dropbox的CPU开销几乎一直保持在100%。

下面我们比较UDS和Dropbox的CPU开销。实验过程同开头说的差不多，唯一的不同是被修改文件放置在UDS的SavingBox文件夹。图 2.24描画了实验结果（请注意Y轴的刻度和图 2.23中的不同），很明显UDS和Dropbox的组合比如仅使用Dropbox节省了很多CPU开销：平均CPU利用率仅为12%（实际上主要来自rsync操作）、介于6%到20%之间，而其中Dropbox 的平均CPU 利用率只有2%。之所以能大量节省CPU开销，原因就在于UDS对文件更新的批处理，从而大大减少了Dropbox的（重新索引被修改文件的）活动。随着时间的推移，文件逐渐变大，索引文件的CPU开销增加，从而UDS的CPU开销也跟着增加。

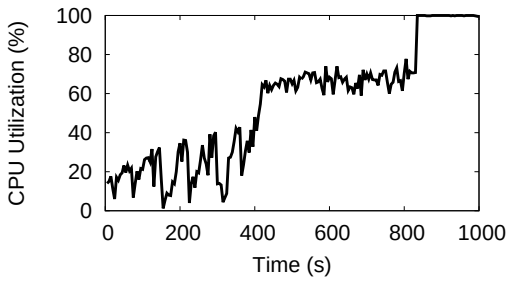


图 2.23 Dropbox的CPU利用率（不使用UDS）

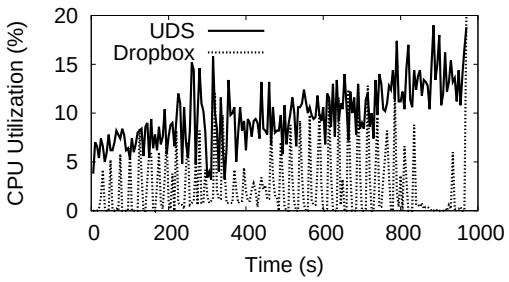


图 2.24 Dropbox和UDS的CPU利用率

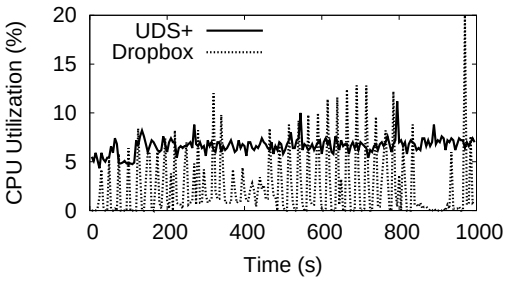


图 2.25 Dropbox和UDS+的CPU利用率

UDS+的实现和性能。虽然UDS极大地减少了云存储应用的CPU开销，但我们的问题是：还能做的更好吗？尤其是在开发和测试UDS的过程中，我们意识到Linux内核的inotify系统调用的一个缺点：它只汇报文件系统事件的发生对象，而不汇报*在哪里发生*以及*改变了多少*。这两个信息对上层云存储应用非常重要，因为如果能够获得它们、我们就能直接把握文件被修改了多少；但目前由于inotify不汇报这两个信息，上层云存储应用只能自己计算（典型地使用rsync来计算）。

我们最重要的发现是：这两个（元）信息在Linux内核中*本来*就存在，只不过当前的inotify系统调用没有向上层汇报而已。因此，让Linux内核向上层汇报这两个信息并不会给内核带来什么负担，却可以免除云存储应用计算这两个信息的工作量，从而进一步降低CPU开销。

为实现上述想法，我们修改了inotify系统调用的代码来汇报：（1）文件修改的字节偏移量；（2）被修改的字节数。这些修改涉及到inotify和fsnotify

表 2.2 UDS+相关的Linux内核函数

相关于fsnotify和inotify的Linux内核函数：
fsnotify_create_event(...) fsnotify_modify(...) fsnotify_access(...) inotify_add_watch(...) copy_event_to_user(...)
调用fsnotify和inotify的被修改的内核函数：
vfs_write(...) nfsd_vfs_write(...) compat_do_readv_writev(...)

[63]的一系列相关函数（如表 2.2所示）的修改，而fsnotify是inotify赖以实现的子系统。我们还往fsnotify_event和inotify_event两个结构体中增加了相关的偏移量和修改字节数属性。如表 2.2所示，还有一些直接调用inotify和fsnotify的内核函数也需要修改。我们总计修改了163行Linux内核代码，跨越8个内核函数。

修改好inotify系统调用后，我们利用它实现UDS的升级版（称为UDS+）。事实上，UDS+的实现比UDS简单的多，因为它再不需要使用rsync来自己计算文件被修改部分的大小了。反过来，UDS+直接利用升级版的inotify系统调用所汇报的字节偏移量和修改字节数信息来计算文件被修改部分的大小。

为评估UDS+的性能提升，我们再次运行上文中（用来比较UDS和Dropbox的CPU开销）的实验并将结果描画在图 2.25中，可见UDS+比UDS更节省CPU开销：平均CPU利用率仅为7%，而UDS是12%；更重要的是，UDS+的CPU开销比UDS更为稳定，且并不随着时间推移（文件变大）而增加，因为它再也不需要使用rsync来重新索引文件了。

2.1.7 小结和进一步工作

最近几年云存储服务愈发流行。数十家厂商在这一领域激烈竞争，其中好几家的用户数达到了千万级甚至亿级。考虑到越来越多的用户将使用云存储服务来放置和传递数据，“节流”问题变得迫在眉睫。

本文指出了云存储应用在同步本地数据更新（到云端）时可能出现的“流量滥用问题”：所消耗流量远远超出真实数据更新的大小。虽然本文的研究主要关注Dropbox（因为它最早且最流行），但我们已在文中说明了“流量滥用问题”对其它诸多云存储应用依然存在。

我们在多个人工控制的和真实世界的场景下对流量滥用问题进行了测量，以洞察此问题发生的深层机制，从而开发了UDS：一个位于用户文件系统和云存储应用之间的中间件。UDS在基本不影响用户体验的前提下无缝地批处理了本地文件更新，从而有效地避免了由频繁短促数据更新所导致的流量滥用问题，同时极大地降低了CPU开销。更进一步，我们还通过对Linux内核的修改实现了一个增强版的UDS中间件、称为UDS+，他能够避免Dropbox和UDS所需的文件索引操作、进一步降低CPU开销。

UDS和UDS+是独立而互补的。UDS是一个独立的中间件，它兼容多个云存储应用，并且不需要对本地操作系统做任何修改，未来我们希望将UDS扩展到其它多个操作系统、比如Windows、Mac OS X 和Android。UDS+对于那些追求更好的云存储应用性能的用户是一个值得推荐的扩展中间件，未来我们希望将我们对于inotify系统调用所作的修改融合进主流的Linux内核，因为一大类云存储应用将受惠于它。

2.2 云存储的节流效率研究

2.2.1 背景、动机及工作简介

近年来（个人）云存储服务（如Google Drive、OneDrive、Dropbox等）愈发流行，为用户提供了方便、可靠、安全的数据存储和分享功能。OneDrive号称已拥有超过2亿用户、存储数据量超过14 PB [64]，而Dropbox也拥有超过1亿注册用户、日均存储或更新约10亿个文件 [39]。此外，谷歌公司于2012年4月发布了其Google Drive服务，仅发布2个月就迅速吸引到1000万用户 [65]。

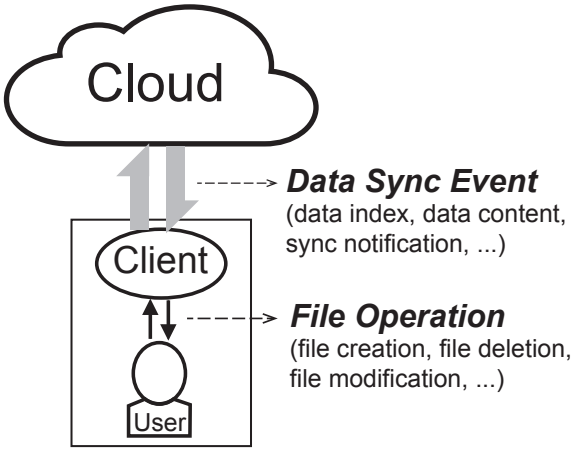


图 2.26　云存储系统的基本架构与操作

如图 2.26所示，一个云存储系统通常由两个基本部分构成：（1）前端客户端，安装并运行在用户设备上；（2）后端云，存放用户数据并且监控客户端状态信息，一般由大规模数据中心组成。用户在安装客户端应用时一般需要指定一个特殊的“同步文件夹”，每当用户在此文件夹中创建文件、删除文件或者修改文件时，所做的数据更新都会被客户端自动同步到后端云，同步过程对用户几乎完全透明。而当用户切换到另一台设备（个人电脑、平板电脑或智能手机）时，客户端将帮助他自动从云端获取最新的数据。此外，多个用户可以依靠云存储分享同一个文件（夹），其中任意一个用户的数据更新会被自动分发给其他用户（的客户端）。

虽然云存储服务在商业上取得了巨大成功，但在成功背后，云存储提供商和用户都在承担来自巨大的**数据同步流量**的“痛苦” [66–70]。为了对云存储

的数据同步流量有一个量化的理解，我们分析了欧洲学者 [41]最近在ISP层面收集到的一份大规模的Dropbox用户行为数据集 [42]。分析结果显示：（1）数据同步流量占云存储服务总流量的比例达到了90%，而云存储服务总流量则高达YouTube视频服务总流量的三分之一；（2）同步一次文件操作平均消耗约2.8 MB 的“入站”（inbound）流量加上5.18 MB的“出站”（outbound）流量，这里“入站”指的是从用户到云端，“出站”指的是从云端到用户。因此，整个Dropbox系统一天的同步流量可合理估计为$> 7.98\text{ MB} \times 1\text{ billion} = 7.4\text{ PB}$。根据Amazon S3的收费政策 [71]（因为Dropbox将所有文件内容存储在S3、而S3只收出站流量的钱），Dropbox一天就要在同步流量上花费$> \$0.05/\text{GB} \times 5.18\text{ MB} \times 1\text{ billion} = \$260{,}000$（假定Dropbox和S3 之间并无特殊内部协议价）。上面的计算表明：数据同步确实消耗了巨大的互联网流量，相应产生高昂的开支 [67]。

事实上，数据同步流量也给用户带来了诸多不便和超支。比如说，我们经常观察到移动用户、尤其是那些流量受限（*traffic cap*）的移动用户 [72]抱怨其手机的同步流量超支，有些人甚至提醒道：“*Keep a close eye on your data usage if you have a mobile cloud storage app.*” [69]（中文翻译：“如果你用了一个移动云存储应用，要密切关注它的数据量使用。”）此外，甚至是一些大型用户也深受同步流量开支之苦，一些人说到：“*The bandwidth costs from the ISP are double the storage charges from Amazon.*” [70]（中文翻译：“ISP收的带宽费是亚马逊收的存储费的双倍。”）

本文试图解决一个简单然而关键的问题：*当前工业云存储服务所消耗的数据同步流量都是必要的吗？* 换句话说，*它们所设计的数据同步机制是否有效利用了同步流量，特别是对于那些流量受限或带宽受限的用户？* 最基本的原则是：在不损害用户体验的前提下，这些云存储服务的设计必须尽最大可能节省数据同步流量。

为了回答上述问题，本文首先提出了一个新的指标“***TUE***”来标识云存储服务的（数据同步）**节流效率**（***traffic usage efficiency***）。在云计算领域，衡量节能效率的经典指标是“***PUE***”（***power usage effectiveness*** $= \frac{\textit{Total facility power}}{\textit{IT equipment power}}$ [73]），与此类似，我们定义$\frac{\textit{Total data sync traffic}}{\textit{Data update size}}$为云存储服务的*TUE*。一旦某个文件被用户*更新*，比如说被创建、被删除或被修改，*data update size*（数据更新大小）表示被更新的文件相对于云端存储文件的“变化”部分，因为从用户的角

度看，数据更新大小是对同步流量*应该*为多少的直观、自然的感知。与同步流量的绝对值相比，*TUE*可以更好地反映云存储服务本质的流量利用能力。

为寻求对于*TUE*坚实和深入的理解，我们收集了一份真实云存储用户数据集，并在此基础上进行了一系列特殊设计的实验，以研究*TUE*的关键“影响因素”和“设计抉择”。就影响因素来说，我们检查了*文件大小*、*文件操作*、*数据更新大小*、*硬件配置*、*网络环境*以及*访问方式*（包括PC客户端、Web浏览器以及移动应用）等等。（数据同步机制的）设计抉择则包括*数据同步粒度*、*数据消重粒度*、*数据压缩级别*以及*同步推迟*[1]。

通过分析上述影响因素和设计抉择，我们发现了主流云存储服务（包括Google Drive、OneDrive、Dropbox、Box、Ubuntu One和SugarSync等）在*TUE*方面的关键特性、设计权衡和优化空间，总结在表 2.3中。最重要的是，它们确证了：即使是当前国际上最主流的云存储服务，*所消耗的数据同步数据流量中很大一部分都是不必要的，并且实际上可以通过谨慎的数据同步机制设计来有效避免或减少*。也就是说，*当前最主流的云存储服务的数据同步机制还存在很大的优化空间*。此外，由于我们所有的实验都在用户端进行，不涉及到云端的内部知识，所以我们的研究方法学是普适的，而我们的研究结果也是容易重现的。

最后，总的来说，我们对云存储节流效率的研究可以帮助服务提供者开发更为经济、节流的数据同步机制；同时，还可以指导用户、特别是那些流量或带宽受限的用户如何选择一家适合他们需求和预算的云存储服务。

2.2.2 相关工作综述

有4份工作和我们的最为相关：GoodBadUgly [46]、Benchmarking [74]、CloudCmp [3]和MDTA [54]。下面我们简述每一份工作，并同我们的工作进行对比。

GoodBadUgly：就我们目前所知，学术界最早的针对云存储服务的比较性研究是胡等人的工作，他们研究了4个云存储服务的“the Good, the Bad and the Ugly”：Dropbox、Mozy、CrashPlan和Carbonite [46]。具体来说，他们测量了简单文件创建的备份和取回时间、网络流量和CPU利用率，还讨论了一些数据压

[1] *同步推迟（Sync deferment）*不同于经常看到的概念*同步时间（sync delay）*。一个文件操作发生后，*同步时间*表示这个文件操作需要多长时间才能完全同步到云端，但*同步推迟*表示相应的同步过程被客户端故意推迟了多久。

表 2.3 我们的主要发现和它们的启发

简单文件操作方面的发现	简单文件操作方面的启发
1) *文件创建*: 在我们收集的云存储用户数据集中，77%的用户文件是小文件（不超过100 KB），而同步一个小文件通常导致很大的*TUE*，因此很不节流。	接近三分之二（66%）的小文件可以被*批量同步*（batched data sync、简写为BDS）以大幅节省同步流量。然而现状是：只有Dropbox和Ubuntu One部分实现了BDS。
2) *文件删除*: 同步流量一般接近0。	用户删除文件时无需关心流量。
3) *文件修改*: 84%的文件被用户修改过。文件修改的*TUE*主要受*数据同步粒度*的影响，而*数据同步粒度*在不同的云存储服务间差别很大：大多数服务采用粗粒度的*全文同步*，也有一些服务（如Dropbox和SugarSync）使用了细粒度的*增量同步*（incremental data sync、简写为IDS）来为其PC客户端节省流量，但增量同步从未见诸于网页浏览器或者移动应用。	大多数现存云存储服务构建于RESTful（自表义、无状态的）基础设施之上，比如亚马逊S3、微软Azure以及OpenStack Swift，它们仅支持全文级的数据操作。要想显著提升文件修改的*TUE*，就必须实现IDS机制，而这一般需要云存储服务提供者搭建一个额外的中间层，将文件的MODIFY操作转换成GET + PUT + DELETE操作。
数据消重和压缩方面的发现	**数据消重和压缩方面的启发**
4) *数据压缩*: 52%的文件可以被有效压缩，但Google Drive、OneDrive、Box和SugarSync从不压缩。Dropbox是唯一的针对所有访问方式都做压缩的云存储服务。	对云存储服务提供者而言，数据压缩能够减少24%的同步流量。对用户来说，基于Web浏览器的文件同步过程一般不压缩数据，而移动应用的压缩程度一般不及对应的PC客户端。
5) *数据消重*: 虽然我们观察到18%的文件可以被消重，但令人惊奇的是，大多数云存储服务并不支持数据消重，尤其是对于基于Web浏览器的访问方式。	对云存储服务提供者而言，要想同时实现数据压缩和数据块级别的消重十分复杂。基于真实数据集的分析，我们建议同时实现数据压缩和全文级别的消重，因为这两者可以无缝结合。
文件频繁修改方面的发现	**文件频繁修改方面的启发**
6) *同步推迟*: 一个文件的频繁修改经常导致很大的*TUE*。一些云存储服务已经意识到这个问题，并通过故意设置固定同步推迟的方法来解决它，然而该方法仅适用于有限的应用场景。	对云存储服务提供者而言，我们提出一种自适应同步推迟（adaptive sync defer、简写为ASD）的方法，可以很好地克服前述固定推迟方法的缺点。
7) *硬件和网络*: 对频繁修改，较老的用户设备常可以节省更多同步流量，但并不损害用户体验；对于网络带宽较小或时延较大的用户也是如此。	对用户来说，当面临频繁文件修改时，今天的云存储服务反而给那些设备较老、网络较差的用户带来了节流的好处。

缩、数据消重、责任限制、隐私和安全问题。

Benchmarking：Drago等人比较了五个云存储服务（包括Dropbox、Google Drive、OneDrive、Amazon Cloud Drive和Wuala）的系统能力，发现每个服务在数据同步方面都有其局限性 [74]。他们的发现肯定了研究云存储数据同步机制及其节流效率的重要性，同时也指出了优化云存储数据同步机制的可能性。

我们的工作在三个方面不同于GoodBadUgly以及Benchmarking。首先，我们的工作并不探讨过多的事情，而只专注于一个问题：节流效率（*TUE*）。我们寻求对于*TUE*深度和细粒度的理解，这是我们一切实验和分析的核心动机。其次，我们专注于6个最主流的云存储服务，因为它们支配了市场，虽然其它服务也有涉及。最后，对每个服务我们进行了*系统化*的测量：我们测量了各种各样的文件操作，通过多种不同的访问方式、在各种各样的网络和硬件配置场景中的性能，*系统化*的测量比起简单随意的测量能够反映出更多坚实的结果和出人意料的发现。

CloudCmp：为了帮助用户找到适合他们需求的服务，李昂等人开发了一个名为“CloudCmp” [3]的工具，以比较商业云存储服务提供商的性能和开支，包括Amazon AWS、Microsoft Azure、Google AppEngine和Rackspace CloudServers。他们发现这些提供商的性能和开支在虚拟实例、存储服务和网络传输等方面很不相同。虽然如此，CloudCmp 所面向的“用户”主要是企业级用户而不是个人云存储服务所定位的普通用户，所以它所关心的性能指标和我们所研究的区别很大。

MDTA：陈等人的工作“MDTA”以对于企业存储系统multi-dimensional trace analysis（多维数据集分析）为特征 [54]。通过分析两个大规模的真实系统数据集，尤其是它们在文件系统层面上的各种数据更新模式，陈等人找到了一种新的途径来更好地使用企业存储系统。我们的工作研究的是个人云存储服务中的多种数据更新模式，给用户提供一系列指导原则，帮助他们设计或选择更好的个人云存储系统。

云存储服务方面还有一些其它的相关工作，比如数据消重 [48,49]、云支持的文件系统备份 [56,57]、虚拟化的影响 [47]等等，这里不再详述。

2.2.3 云存储服务的一般设计框架

从数据同步流量的角度看，云存储服务的一般设计框架涉及到诸多的影响

表 2.4 *TUE*的关键影响因素和*设计抉择*

客户端	客户端位置　客户端硬件 访问方式　文件大小　文件操作 数据更新大小　数据更新速率 *数据压缩级别*　*同步推迟*
服务器端	*数据同步粒度* *数据消重粒度* *数据压缩级别* (可能不同于客户端)
网络层面	同步流量　带宽　时延

因素和设计抉择，它们可能位于客户端、服务器端（云端）或者是网络层面。为了避免被过多琐碎或难以捉摸的问题所影响，我们主要基于下面两个准则来挑选所要研究的关键影响因素和设计抉择：

- *准则1：影响因素必须相对不变或稳定，这样我们的实验结果容易被他人重复。*
- *准备2：设计抉择应该是可以从外部测量到的，且独立于特定的云存储服务，这样我们的研究方法更具普适性。*

基于上述准则，最终我们选择了10个关键影响因素和4个关键设计抉择，见表 2.4。其中大多数概念是自解释的，少数可能引起歧义的下面进一步澄清：

- *数据同步粒度*：用户所做的一个文件操作可以通过两种不同的粒度同步到云端，要么是全文级的（被修改的文件整体上传），要么是数据块级的（仅上传被修改文件中实际被修改的数据块）。
- *数据消重粒度*：同一用户或者不同用户的文件之间可以通过两种不同的粒度进行消重，以节省同步流量，要么是全文级的，要么是数据块级的。
- *带宽*：定义为客户端和云端之间的峰值数据上传速率。
- *时延*：定义为客户端和云端之间的通信往返时间（RTT）。

2.2.4 实验方法

（a）真实用户数据集

我们的研究使用了一份真实云存储用户数据集。它收集自中国和美国的多所高校和公司，涉及到153个云存储用户以及他们保存在云存储同步文件夹中的222632个文件。数据集中记录了每个文件多

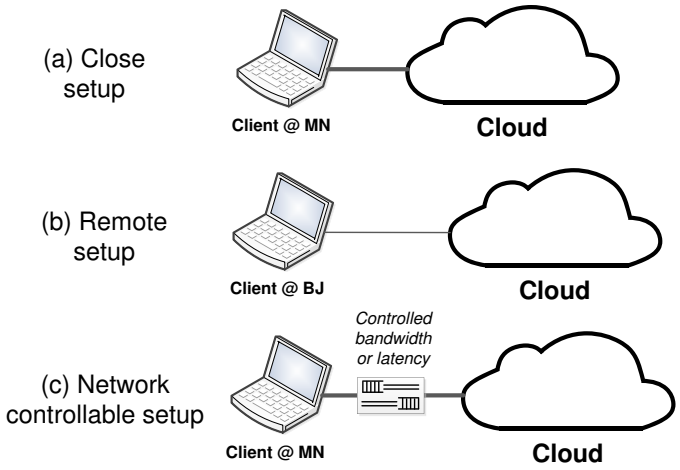


图 2.27　实验配置

方面的详细信息，包括用户名、文件名MD5、文件原始大小、文件压缩后大小、创建时间、最后一次修改时间、全文MD5以及数据块（从128 KB到16 MB）级MD5。可以通过如下链接下载到我们收集的数据集：*http://www.greenorbs.org/people/lzh/public/traces.zip*.

（b）实验配置

为重现多样化的使用场景，我们构造了多个不同的实验配置（experiment setups），使用了多台客户端计算机（包括移动设备）、位于地理差异极大的两个位置，如图 2.27(a)和图 2.27(b)所示。此外，为了细粒度地调整网络环境，我们还实现了一个网络环境可控的实验配置，如图 2.27(c)所示。

- **云存储服务**。在当今数十个流行的云存储服务中，我们重点关注最主流的6个：Google Drive、OneDrive、Dropbox、Box、Ubuntu One和SugarSync，它们要么拥有极多的用户，要么在数据同步机制或者系统架构上具有代表性。
- **客户端位置**。由于上述云存储服务主要部署在美国，我们选择了两个地理差异极大的位置：MN（即美国明尼苏达）和BJ（即中国北京）来进行每个实验。很明显，位置MN代表典型的靠近云端的位置，而BJ则代表典型的远离云端的位置。
- **客户端设备**。我们的实验使用了多台设备。在MN，用到了一台过时配置的笔记本电脑（5400 RPM传统硬盘）、一台主流配置的笔记本电

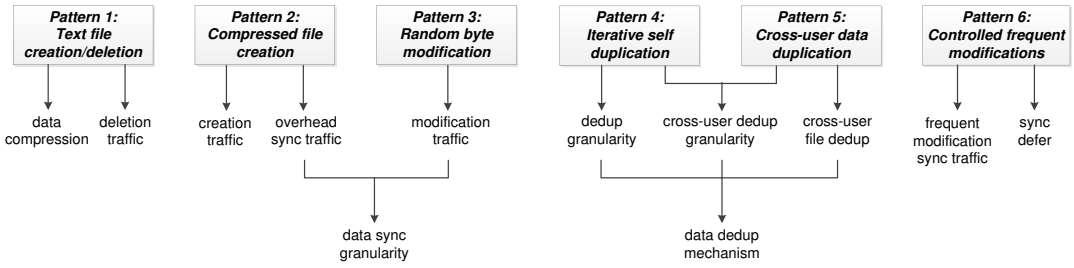


图 2.28　数据更新模式，以及它们的输出和依赖性

脑（7200 RPM传统硬盘）、一台前沿配置的笔记本电脑（SSD 固态硬盘），以及一个主流配置的Android智能手机。在BJ，非常类似，也用到了一台过时配置的笔记本电脑（5400 RPM传统硬盘）、一台主流配置的笔记本电脑（7200 RPM 传统硬盘）、一台前沿配置的笔记本电脑（SSD 固态硬盘），以及一个主流配置的Android智能手机。笔记本电脑都安装了Windows 7 SP1操作系统，客户端接入互联网的带宽都在20 Mbps左右。

- **可控网络环境**。为细粒度调整网络环境（包括带宽和时延），我们在MN的客户端和云端之间嵌入一组数据包过滤器（运行在一台Linux代理服务器之上），它们使得我们能够细粒度地调整双向的带宽和时延。具体来说，这对数据包过滤器是通过使用标准的Linux工具netfilter/iptables来嵌入的，因此它们就像常见的客户端防火墙一样工作。此外，一些云存储软件（比如Dropbox）还允许直接在软件上配置上传、下载带宽。这里我们没有使用北京的客户端，因为它们本来的带宽和时延状况就十分不佳，很难进一步限制。

（c）设计数据更新模式

如图 2.28所示，我们设计了6种不同的数据更新模式，它们之间有一定的依赖性，提供我们一条线索：如何利用这些模式的实验结果来识别出*TUE*的关键影响因素。

- *模式1：文本文件创建/删除*。在同步文件夹中创建一个10 MB的文本文件，它实际上是从互联网上下载的一部网络小说，目的是检查云存储服务是否压缩数据。此外，每个文本文件创建好后会被删除，以获取文件删除的同步流量信息。
- *模式2：压缩文件创建*。在同步文件夹中创建一个Z字节的压缩文件，以弄清*附加同步流量*（overhead sync traffic）的多少，其中Z =

$1, 2, 4, \cdots, 128M$，实际上Z越小越容易反映出附加同步流量的情况。由于我们采用了最高强度的压缩文件，所以附加同步流量的计算很简单：用总的同步流量减去文件大小Z即可。

- *模式3：随机字节修改*。在一个现存的Z字节的压缩文件中随机修改一个字节，以搞清楚数据同步粒度。根据“差分同步”的工作原理 [45]，同步粒度（也就是同步数据块大小C）可以被近似估计为：

$$C \approx \textit{总的同步流量} - \textit{附加同步流量}。 \tag{2.1}$$

- *模式4：迭代自复制*。推断数据消重粒度有些复杂，特别是当消重数据块大小B（字节）不是2的整数次幂时。基本的迭代自复制过程如下所示：

迭代自复制过程

步骤0: 设置数据消重粒度的下限$L = 0$字节以及上限$U = +\infty$字节，并猜测一个可能的消重粒度（数据块大小）B_1。

步骤1: 产生一个新的B_1字节的压缩文件f_1，把它上传到云端。待f_1完全上传之后，记录总的同步流量Tr_1。

步骤2: 通过把f_1添加到它本身（所谓“自复制”）产生第二个文件f_2，名字和f_1不同，再把f_2上传到云端。相似地，在f_2完全上传之后，记录总的同步流量Tr_2。

步骤3: 如果$Tr_2 \ll Tr_1$并且Tr_2很小（一般几十KB），B_1就刚好是消重粒度，从而整个过程结束。

否则，分两种情况：

- 情况1：$Tr_2 < 2B_1$并且Tr_2并不很小，这意味着真正的消重粒度$B < B_1$，那么将B_1设为消重粒度的上限：$U \leftarrow B_1$，并且减小猜测值B_1：$B_1 \leftarrow \frac{L+U}{2}$，进入**步骤1**。
- 情况2：$Tr_2 > 2B_1$，这意味着真正的消重粒度$B > B1$，那么将B_1设为消重粒度的下限：$L \leftarrow B_1$，并且增大猜测值B_1：$B_1 \leftarrow \frac{L+U}{2}$，进入**步骤1**。

容易看出：不管B是不是2的整数次幂，上述过程都可以在$O(log(B))$步之内完成。根据我们的实验结果，几乎所有的流行云存储服务都采用了2的整数次幂作为其消重粒度，因为这样的设置本身也便于系统的设计。

- *模式5：跨用户数据重复*。上传一个文件f到云端之后，我们使用另一个账号再次上传f到云端，通过比较两次上传的同步流量，能够明白此云存储服务是否采用了文件级的数据消重。如果确认采用，再进一步利用类似*模式4*的数据更新模式来确定跨用户的数据消重粒度。
- *模式6：受控频繁修改*。每隔Y秒添加X个随机字节到同步文件夹中的一个现存空文件，直到文件大小达到某个值（比如1 MB），这里我们添加随机字节是因为它们很难被压缩，而X是一个正整数、Y是一个正实数，并且我们经常需要使用X和Y的多个组合来进行研究。这一数据更新模式有三方面的效果：（1）观察频繁文件修改所导致的同步流量和*TUE*；（2）识别云服务是否故意设置了同步推迟；（3）如果确实设置了同步推迟，那么具体推迟了多长时间（T）。

2.2.5 实验结果和发现

（a）文件创建和删除

表 2.5和 2.6列出了在六个最主流的云存储服务中创建一个压缩文件的同步流量。由于文件已被事先高强度压缩，相应的*TUE*可以直接计算为：$\frac{同步流量}{文件大小}$，并且附加同步流量可以合理估计为：*同步流量* – *文件大小*。总的来说，从表 2.5和 2.6可以得到如下发现：

- 同步一个压缩文件创建的*TUE*主要取决于文件大小。小文件（比例高达77%）经常导致很大的*TUE*，而大文件的*TUE*则接近（但永远超过）1.0。所以对云存储服务提供方，我们建议多个相关小文件（比例高达66%）可以在*逻辑上*组合成一个*合适大小*的文件来*批量同步*（batched data sync、简写为BDS）以大幅节省同步流量。遗憾的是，根据我们的实际测量，只有Dropbox和Ubuntu One部分实现了BDS [26]。
- 上面提到文件的*合适大小*，那么它具体是多少？通过将*TUE*和文件大小的关系描画在图 2.29中，我们可以直观看出*合适大小*至少需要达到100 KB、最好超过1 MB，才能有较小的*TUE*——至少低于1.5、最好低于1.2。
- 对于一个合适大小的文件创建来说，使用Web浏览器或移动应用的同步流量一般要少于使用PC客户端，原因主要在于它们的实现复杂度：PC客户端软件的大小动辄几十MB，可以提供比容量很小的Web浏览器插件或移动应用更为复杂的高级功能，自然地，更复杂的功能涉及到更多的通信事

表 2.5　压缩文件创建的同步流量

云存储服务	PC客户端同步流量 (字节)				Web浏览器同步流量 (字节)			
	1	1 K	1 M	10 M	1	1 K	1 M	10 M
Google Drive	9 K	10 K	1.13 M	11.2 M	6 K	7 K	1.06 M	10.6 M
OneDrive	19 K	20 K	1.14 M	11.4 M	28 K	31 K	1.11 M	11.7 M
Dropbox	38 K	40 K	1.28 M	12.5 M	31 K	37 K	1.09 M	10.6 M
Box	55 K	47 K	1.10 M	10.6 M	55 K	58 K	1.10 M	10.5 M
Ubuntu One	2 K	3 K	1.11 M	11.2 M	37 K	39 K	1.20 M	11.3 M
SugarSync	9 K	19 K	1.17 M	11.4 M	31 K	32 K	1.10 M	10.7 M

表 2.6　压缩文件创建的同步流量（续上表）

云存储服务	移动应用同步流量 (字节)			
	1	1 K	1 M	10 M
Google Drive	32 K	71 K	1.27 M	11.0 M
OneDrive	29 K	44 K	1.23 M	10.7 M
Dropbox	18 K	32 K	1.08 M	10.9 M
Box	16 K	34 K	1.29 M	10.8 M
Ubuntu One	20 K	24 K	1.08 M	10.9 M
SugarSync	31 K	47 K	1.22 M	10.9 M

件和信令交换，所以会产生更多的同步流量。

文件删除通常产生极少的、可以忽视的同步流量，不论哪个云存储服务、删除多大的文件，基本上低于100 KB。原因很直接：因为云端从来不真正删除文件内容，它最多是改变下用户文件的属性。这个“伪删除”的做法方便了用户未来的数据恢复，比如常见的文件版本回滚功能。

（b）文件修改和数据同步粒度

在一个现存的Z字节的压缩文件中随机修改一个字节，以搞清楚数据同步粒度：近似为$C \approx$ *总的同步流量* $-$ *附加同步流量*，为了获得更准确的C，这个模式可以反复执行取平均值。我们发现当前的诸多云存储服务实际上只有两种数据同步粒度：*全文同步*和*增量同步*（也称为*差分同步*）。

- **全文同步**。Google Drive是其代表，只要在Z字节的压缩文件中修改一个随机字节，所产生的同步流量就和新创建一个Z字节的压缩文件一样多，换句话说，Google Drive 实际上将每个文件修改操作当成直接上传新文件加上删除旧文件，整个过程和我们使用FTP协议更新文件

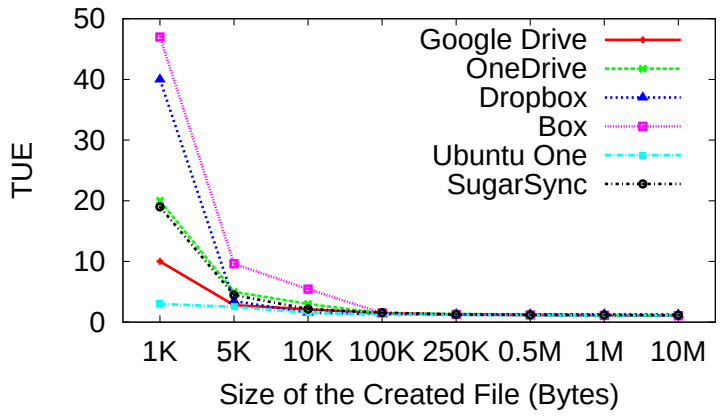


图 2.29 *TUE*和文件大小的关系

一样。因此，Google Drive更适合存储媒体文件，因为它们的内容极少会被用户修改；但不适合存储用户可能经常修改的文档或源代码。此外，OneDrive和Ubuntu One也使用全文同步。

- **增量同步**。Dropbox是其代表，在压缩文件中修改一个随机字节所产生的同步流量总是保持在50 KB左右，和被修改文件大小无关。由于创建一个极小文件的附加同步流量约为40 KB，Dropbox的数据同步粒度$C \approx 50-40=10$ KB。上述数值刚好介于差分同步原始算法中 [45,75]推荐的默认同步粒度范围：从700 B到16 KB。很明显，Dropbox适合处理任何大小的文件修改，而SugarSync也这样。

总结上文，关于数据同步粒度我们有如下发现：

- 文件修改的*TUE*主要受数据同步粒度的影响，而数据同步粒度在不同的云存储服务间差别很大：大多数服务采用粗粒度的全文同步，也有一些服务（如Dropbox 和SugarSync）使用了细粒度的增量同步来为其PC客户端节省流量，但增量同步从未见诸于网页浏览器或者移动应用。
- 不幸的是，大多数现存云存储服务构建于RESTful（自表义、无状态的）基础设施之上，比如亚马逊S3、微软Azure以及OpenStack Swift，它们仅支持全文级的数据操作。要想显著提升文件修改的*TUE*，就必须实现IDS机制，而这一般需要云存储服务提供者搭建一个额外的中间层（仿效Dropbox），将文件的MODIFY 操作转换成GET + PUT + DELETE操作。通过对真实用户数据集的分析，我们发现高达84%的文件曾被用户修改过，因此，对于云存储服务提供者来说，实现一个额外的中间层来支持增

表 2.7　创建一个10 MB的文本文件的同步流量

云存储服务	同步流量 (MB)		
	PC客户端	Web浏览器	移动应用
Google Drive	11.3	10.6	11.8
OneDrive	11.4	11.0	12.2
Dropbox	**6.1**	10.6	**8.1**
Box	10.6	10.5	10.4
Ubuntu One	**5.6**	10.9	**8.6**
SugarSync	11.3	10.4	11.6

量同步是值得的。

（c）数据压缩

减少*TUE*的一个最直接办法就是在同步之前压缩数据。为了检查不同云存储服务在不同访问方式下的数据压缩情况，我们在同步文件夹中创建一个10 MB的文本文件，它是一部纯文本的网络小说，使用高强度的7z压缩工具可以压缩到4.5 MB。表 2.7记录了相应的同步流量，我们从而得到如下发现：

- 真实用户数据集显示，52%的文件可以被有效压缩，能够减少24%的同步流量。然而现状是：Google Drive、OneDrive、Box和SugarSync从不压缩数据。Dropbox是唯一的针对所有访问方式都做压缩的云存储服务。
- 对用户来说，基于Web浏览器的文件同步过程一般不压缩数据，而移动应用的压缩程度一般不及对应的PC客户端。

（d）数据消重

跨越文件和用户的文件/数据块级别重复在今天的互联网中普适存在，如果加以利用能够有效降低同步流量开销。令人遗憾的是，大多数云存储服务并不支持数据块级别的或者跨用户的数据消重，如表 2.8所示。其中“Full file”表示数据消重只发生在整个文件级别，而“4 MB”则表示消重粒度为$B = 4$ MB，“No”表示不进行数据消重。

结合对真实用户数据集分析的结果，我们得到下面的启发：

- 虽然我们观察到18%的文件可以被消重，但令人惊奇的是，大多数云存储服务并不支持数据消重，尤其是对于基于Web浏览器的访问方式。
- 对云存储服务提供者而言，要想同时实现数据压缩和数据块级别的消重十分复杂。我们建议同时实现数据压缩和全文级别的消重，因为这两者可以无缝结合。

表 2.8　数据消重粒度

云存储服务	同一个用户的消重粒度 （PC客户端和移动应用）	跨越用户的消重粒度 （PC客户端和移动应用）
Google Drive	No	No
OneDrive	No	No
Dropbox	4 MB	No
Box	No	No
Ubuntu One	Full file	Full file
SugarSync	No	No

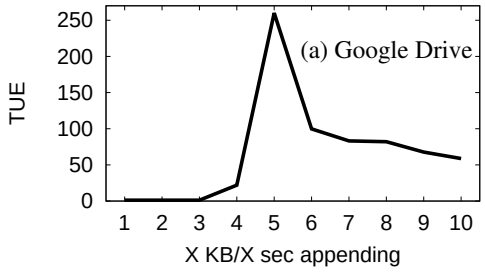


图 2.30　Google Drive处理频繁修改时的同步流量

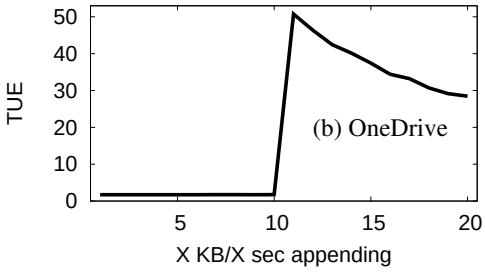


图 2.31　OneDrive处理频繁修改时的同步流量

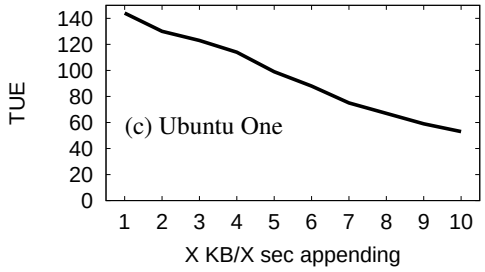


图 2.32　Ubuntu One处理频繁修改时的同步流量

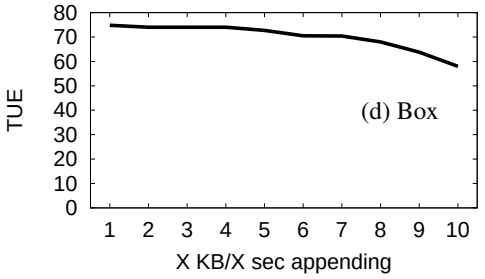


图 2.33　Box处理频繁修改时的同步流量

（e）频繁修改

随着云存储服务愈发流行，它们被用户用来做更为复杂的事情，比如协同文档编辑、团队项目开发、搭建数据库等等，而这些复杂的事情经常涉及到一类特殊的文件操作：“*频繁修改*”，它指的是一个文件被增量式地频繁修改、而不是简单地一次性修改。为理解频繁修改的同步流量情况，我们使用了一个简单的频繁修改模式：每隔X秒添加X K个随机字节到同步文件夹中的一个现

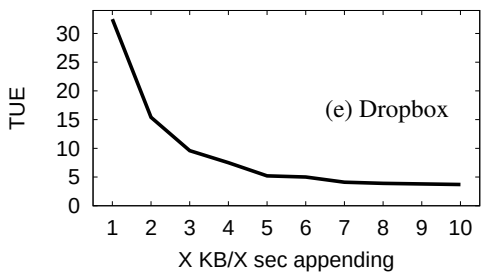


图 2.34 Dropbox处理频繁修改时的同步流量

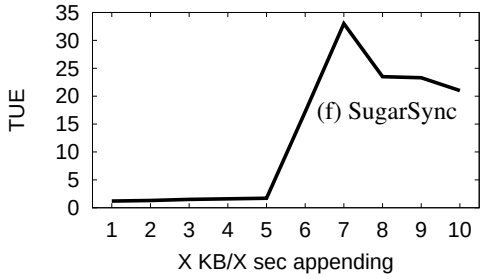


图 2.35 SugarSync处理频繁修改时的同步流量

存空文件，直到文件大小达到某个值1 MB，这样对于每个“X KB/X s”添加模式，总的数据更新更小都是1 MB，总的实验时间都是1024秒，从而比对它们的同步流量情况非常方便。实验配置如图 2.27(a)所示。

对于五个最主流的云存储服务的实验结果如图 2.30– 2.35所示，请注意每幅图的Y轴都不同。从中我们得到如下发现：

- 对一个文件的频繁修改经常导致很大的*TUE*。对于所考察的六个主流云存储服务来说，最大*TUE*可以分别达到260、51、144、75、32和33。那么为什么Google Drive、OneDrive、Ubuntu One和Box的最大*TUE*比Dropbox和SugarSync大很多呢？原因就在于Dropbox和SugarSync的数据同步粒度（差分同步）远小于Google Drive、OneDrive、Ubuntu One和Box（全文同步）。
- 对于OneDrive来说，它在修改周期低于10秒时的*TUE*很小并不是因为同步粒度，而是下一节要研究的*同步推迟*。同理，在Google Drive和SugarSync的曲线图上所存在的*TUE*接近1.0的情况，也归因于*同步推迟*。
- 我们还观察到*TUE*随着修改频率（$= \frac{1024}{X}$）的降低而降低，原因在于较低的修改频率意味着较少的数据同步次数，从而附加同步流量的比例减少了。极端情况下，如果我们令$X = 1024$，频繁修改模式就退化成简单的一次修改，从而*TUE*接近1.0。

（f）同步推迟

针对上一节所发现的“对一个文件的频繁修改经常导致很大的*TUE*”的问题，一些云存储服务已经意识到这个问题，并通过故意设置同

步推迟（如T秒）的方法来解决它。从图 2.30、图 2.31和图 2.35容易发现$T_{GoogleDrive} \in (3,5)$秒，$T_{OneDrive} \in (10,11)$秒，而$T_{SugarSync} \in (4,6)$秒。为了获得更准确的$T$，我们进一步将$X$从整数调整到实数，对于$T_{GoogleDrive} \in (3,5)$形如$X = 3.1, 3.2, \cdots, 4.9$，重复“$X$ KB/X s”数据添加的实验，从而发现$T_{GoogleDrive} \approx 4.2$秒，$T_{OneDrive} \approx 10.5$秒，而$T_{SugarSync} \approx 6$秒。

虽然如此，我们发现现存的同步推迟机制还是有两个缺点：

- 现存的同步推迟机制经常被误用，从而导致甚至是最简单的（一次性）文件创建和修改操作也被推迟同步、损害了用户体验，比如图 2.31所示。
- 由于采用固定的同步推迟时间，现存的同步推迟方法仅适用于狭窄的应用场景，可以从图 2.30、 2.31和 2.35中明显看出来。
- 对云存储服务提供者而言，我们提出一种自适应同步推迟（adaptive sync defer、简写为ASD）的方法，可以很好地克服前述固定推迟方法的缺点 [26]。

（g）网络环境和硬件配置

为检查网络环境对云存储服务的节流效率的影响，我们进行了两批实验。第一批是在北京做的，也就是使用图 2.27(b)所示的“Remote setup”，它代表一个典型的相对差的网络环境——带宽低、时延高。第二批是在明尼苏达做的，使用图 2.27(c)所示的“Network controllable setup”，它可以细粒度地调节带宽和时延。从两批实验的结果中（如图 2.36、图 2.37和图 2.38所示）我们得到下面的发现：

- 简单文件操作（比如一次性的文件创建、删除和修改）的*TUE*不受网络状态影响。
- 对频繁修改，网络环境较差的用户反而产生更少的同步流量，但并不明显损害用户体验。

为检查硬件配置对云存储服务的节流效率的影响，我们使用3台硬件配置差别极大的客户端设备M1（典型配置）、M2（过时配置）和M3（强大配置）在同一个地点（明尼苏达）进行实验。从图 2.39可以看出：

- 硬件配置较差的用户设备在面临频繁修改时，所消耗的数据同步流量反而更少。

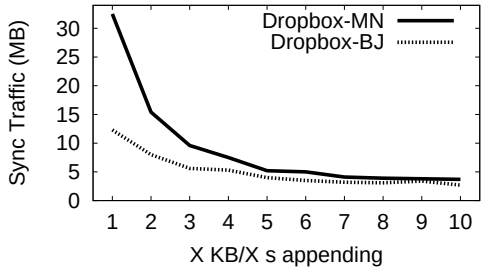


图 2.36　处理频繁修改时Dropbox在明尼苏达和北京的同步流量

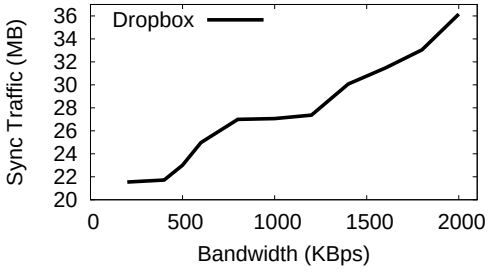


图 2.37　不同带宽环境下处理“1 KB/1 s”添加模式时Dropbox的同步流量

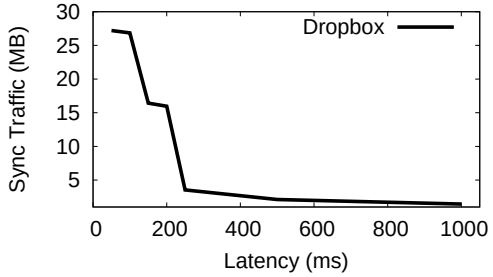


图 2.38　不同时延环境下处理“1 KB/1 s”添加模式时Dropbox的同步流量

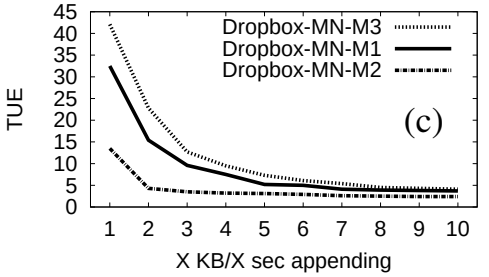


图 2.39　不同硬件配置下处理“X KB/X s”添加模式时Dropbox的同步流量

2.2.6 小结和进一步工作

本文提出一个新的指标“*TUE*”来量化云存储服务的（数据同步）节流效率，并基于真实用户数据集、在多个应用场景、采用多种访问方式测量了当前最主流的6个云存储服务：Google Drive、OneDrive、Dropbox、Box、Ubuntu One和SugarSync的节流效率，从而得到了一系列有趣的发现和有意义的启发，可以帮助云存储服务提供者开发更为经济、节流的数据同步机制，还可以指导用户（特别是那些流量或带宽受限的用户）如何选择一家适合他们主要需求和预算的服务。

第三章　云辅助的内容分发

3.1 云跟踪系统的挑战设计与性能

3.1.1 背景、动机及工作简介

互联网内容分发的现存技术大致可分为3类：直接内容分发（C/S）、P2P（Peer-to-Peer network、对等网络）和混合式内容分发。本文第1章的1.1节已经介绍过这3种内容分发技术（此外的CDN 技术可以认为是C/S的一种扩展和优化）并比较过各自的优缺点，这里不再累述。作为混合式内容分发的一种前沿技术，近几年出现的**P2SP**技术（全称**Peer-to-Server&Peer**）兼容C/S 的稳定与可靠、P2P的廉价与规模，表现出良好的系统扩展性和操控性，工业界的典型代表如Spotify、优酷的“i酷加速器”、土豆的“飞速土豆”、搜狐视频的客户端、PPTV [17]、PPStream、UUSee [18]、风行、FS2You [76]、LiveSky [1]和Akamai NetSession等。如图 3.1所示，P2SP技术通常使用一个**私有云（Private owned cloud）**或私有服务器集群加上一系列的**用户集群（Peer swarm）**进行内容分发。一个用户集群首先需要从云端获取一个“内容种子”，然后才能开始集群内部的数据交换（一般使用一种私有的P2P协议），P2SP的显著优点是能够以稳定可控的云端资源选择性地帮助那些用户较少、资源匮乏的用户集群。

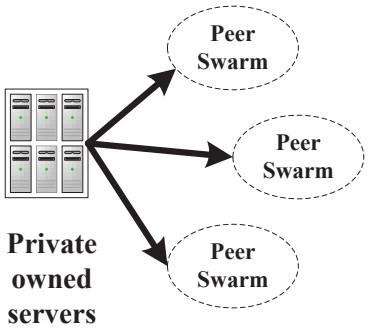


图 3.1　P2SP

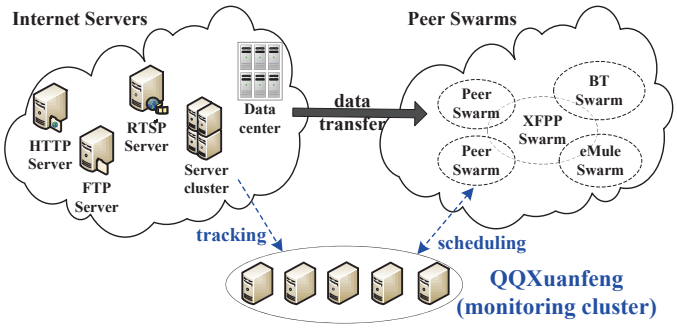


图 3.2　Open-P2SP（“云跟踪”）

虽然具有诸多优点，但P2SP仍然是个“封闭”系统：使用自己的私有云加速自己的私有用户集群，所以在内容丰富程度和服务器带宽上依然受限。首先，出于版权和存储的考虑，一个P2SP系统只能提供互联网内容的很少部分，因此用户在使用一个P2SP系统的同时往往还要求助于其它C/S、P2P和P2SP系统以获得别的内容，使用起来很不方便。其次，由于用户集群的高动态性，很可能一个P2SP系统的服务器带宽不能满足用户需求、尤其是面临用户规模突发膨胀时。因此出现了更新、更开放的“**Open-P2SP**”内容分发技术 [19]，它比P2SP强大的地方在于－－用户可以跨越协议、跨越系统，从完全不同的P2P数据集群（比如一个是BitTorrent集群、另一个是电骡集群）和内容服务器（比如一个是HTTP服务器、另一个是RTSP服务器）中并行获取数据，如图3.2所示。

Open-P2SP技术的显著特色是使用一个较小规模的云（也称监控集群、Monitoring cluster）来跟踪极大规模的互联网服务器及其内容，因此Open-P2SP技术也被形象地称为“**云跟踪**”（**Cloud Tracking**），下面的章节我们主要使用“云跟踪”这个更为简短和亲和的名称。云跟踪系统的典型代表为迅雷 [20]、QQ旋风、Flashget、Orbit和QVoD。云跟踪技术的优点可以总结为如下4 个方面：

(1) **开放收集**。云跟踪使用监控集群持续地跟踪和索引互联网中众多第三方服务器的海量可下载内容（主要包括音频、视频、文档、软件等），云跟踪使得用户能够快速、方便地找到所需内容，同时还能引导那些对相同内容感兴趣的用户形成（更大、更开放的）数据集群。

(2) **动态调度**。云跟踪动态地引导那些用户较少、资源匮乏的用户集群到内容匹配的第三方服务器中寻求下载加速。由于被跟踪的服务器数量巨大（往往在百万甚至千万量级），小规模或局部性的用户需求突发膨胀不会给云跟踪系统带来多大问题。

(3) **负载均衡**。通过智能调度用户集群到第三方服务器中获取数据（和带宽），云跟踪能够在一定程度上平衡第三方服务器的负载、同时避免任何一台服务器的过载。

(4) **轻量构建**。云跟踪系统索引的第三方服务器数量巨大，但它本身所需构建的监控集群却很轻量：既不需要大量的数据存储，也不需要大量的上传带宽（这同私有P2SP系统形成鲜明对比）。一个实证的例子就是我们本文将要介绍的“QQ旋风云跟踪系统”，其监控集群仅有53台商品化服务器，

每天却能跟踪约100万第三方服务器和服务约500万用户。

尽管拥有上述四大优点，云跟踪相比一般的P2SP技术涉及到更多更难的问题。设计一个P2SP系统时主要考虑的问题是处理用户的动态性和合理分配私有云带宽，然而设计一个云跟踪（Open-P2SP）系统（以QQ旋风为例）还要考虑更多的挑战性问题：

(1) **处理服务器和内容的动态性**。对于一般的P2SP系统来说，服务器及其所存内容通常是稳定、可控的，所以设计者可以专注于处理（不可避免的）用户动态性。但云跟踪系统涉及到各种各样的第三方服务器和内容，且两者都具有完全不可预测的动态性——第三方服务器从来都不会告知QQ旋风它们的加入、离开或内容变化。
(2) **服务器带宽的受限使用**。一般的P2SP系统可以100%使用服务器带宽，因为服务器是系统私有的，但云跟踪系统不能——它必须将其带给第三方服务器的额外带宽负担限制在一定范围内，否则可能干扰到第三方服务器本身的服务能力。更困难的地方在于，云跟踪系统根本就不知道第三方服务器“本身”的带宽负担。
(3) **用户集群的差异化加速**。由于服务器带宽的受限使用，云跟踪系统不大可能加速所有的用户集群到令人满意的程度（即拥有高下载速度）。因此，QQ旋风系统的对策是根据不同用户集群的性能需求来设计合理的差异化加速方案。
(4) **给第三方带来额外收益**。即使云跟踪系统将其带给第三方服务器的额外带宽负担严格限制在合理范围内，第三方（服务器提供者）依然可能不愿意支持云跟踪技术，除非他们的支持能获得额外的收益。

对于一个真实世界的大规模云跟踪系统来说，上述每个问题都具有相当的挑战性。事实上，在QQ旋风系统从上线到今天的几年运营和优化过程中，我们从来都没有为上述任何一个问题找到一个完美的答案；代替地，我们的办法是基于长期的大规模系统测量、为每个问题找到一个合理且实用的解法：

(1) 为了处理服务器和内容的动态性，我们构造了一个“内容爬虫”（服务器集群）和一个“内容验证器”（服务器集群）。前者在第三方服务器上不断爬取内容链接（URL），同时也接受用户端汇报的新链接。后者接受用户汇报的“错链”和“死链”并亲自到互联网上验证它们。

(2) 取样测量结果显示：第三方服务器的**“原始带宽利用率”（Original bandwidth utilization、简写为OBU）**通常都低于60%，因此QQ旋风引导到一台第三方服务器上的**“额外带宽利用率”（Extra bandwidth utilization、简写为*EBU*）**最好控制在40%以下（40% = 1 − 60%）。具体来说，QQ旋风通过周期性地收集用户汇报信息来计算第三方服务器的*EBU*，一旦发现某台服务器的*EBU*高于40%、就通知一部分对应的用户停止从该服务器下载数据。

(3) 虽然QQ旋风系统的每个用户都希望获得最佳用户体验，但我们观察到他们的“基本期望”是不一样的，从而根据每个用户集群的实时下载速率和数据供需状况将他们分成3类以区分加速：（a）饥饿集群，（b）潜在饥饿集群和（c）高需求集群，目标是保证每个用户集群的下载速率至少高于其基本期望。

(4) 对一个第三方（服务器提供者）而言，通常有两个收益指标是最重要的：***PV*（全称*Page view*）** [77]和***PTC*（全称*Paid to click*）** [78]。QQ旋风一直致力于在这两个指标上同第三方合作共赢，以鼓励更多的第三方支持QQ旋风的云跟踪技术。

在过去的5年里（从2007年发布第一个软件版本至2012年的3.9版），QQ旋风已累积超过1.2亿的注册用户，并支持几乎所有的互联网主流内容分发协议：HTTP、FTP、RTSP、BitTorrent、eDonkey/eMule等等。目前，QQ旋风日均调度约500万用户从约100万台第三方服务器获取PB量级的数据，单用户的平均下载速率从57 KBps加速到158 KBps，其中45%的流量来自第三方服务器。同时，第三方服务器的额外带宽利用率基本被控制在40%以下，因此它们本身支持的服务基本不会被干扰。就我们所知，本文是对于云跟踪（Open-P2SP）内容分发技术的第一份“白盒”研究，为领域同行提供了坚实的经验和有价值的启发。

3.1.2 相关工作综述

最近几年有很多P2SP方面的研究，比如P2SP流媒体 [18]、P2SP存储 [76]、混合式CDN-P2P内容分发 [1, 79]等等。

P2SP流媒体。吴川等人 [18]研究了P2P视频流媒体分发过程中服务器的重要性。通过对一个流行的P2P在线电视系统（即UUSee）的测量，他们发现：虽

然P2P内容分发具有用户越多、（上传带宽）贡献越多的优点，UUSee系统所提供的150 台专用流媒体服务器总的可用带宽不能满足数百个电视频道日益增长的对下载带宽的需求，这里一个电视频道可以看成一个P2P用户集群。从而，他们提出了一个名为Ration的服务器带宽分配算法，从每个电视频道的历史信息中前动地预测其对服务器带宽的最小需求，再根据这些需求给每个频道分配适量的服务器带宽。

P2SP存储。基于Rayfile这样一个流行的P2SP文件存储系统，孙等人 [76]探讨了P2P协同存储这一应用场景下文件可用性、下载性能和服务器资源开销（包括带宽和存储开销）之间的权衡。为节省服务器带宽开销，他们总结了允许一个用户向服务器获取数据的3个条件：（1）无其他在线用户拥有此用户所请求的文件；（2）无其他在线用户拥有此用户所请求文件的一个特定数据块；（3）此用户所在P2P用户集群的平均下载速度低于10 KBps。为高效利用服务器存储容量（保存更多的文件），当一个用户上传一个文件到Rayfile系统时，系统服务器保证此文件在服务器上只有一个副本。

混合式CDN-P2P内容分发。CDN一般用来加速基于服务器的内容分发，但尹浩等人 [1]在ChinaCache（中国最大的CDN提供商）上开发了一个混合式的CDN-P2P视频流媒体系统，以聚合双方面的优势：CDN的内容分发质量可控性和可靠性，以及P2P的可扩展性。他们提出一种动态扩展资源的机制以保证给予端用户合适的服务质量，从而克服了普通P2P流媒体的典型缺点、比如高缓冲需求和低服务质量。此外，基于Akamai NetSession接口 [79]，Haeberlen等人 [80]设计了一种方法以提供混合式CDN-P2P内容分发系统中可靠的用户互动审计。

迅雷系统。迅雷是中国最大的（并且很可能是全球最大的）云跟踪系统，然而到目前为止还没见到有关其系统设计或性能的官方资料。已经有了一些对于迅雷系统的初步研究 [20,81,82]，它们都使用了“黑盒”测量方法。在数据调度策略方面，迅雷一直比较激进，它经常会“吸收”全部的第三方服务器带宽来最大限度地提升用户体验，使得诸多第三方服务器的提供者抱怨迅雷损害了他们本身的服务能力；反过来，QQ旋风在数据调度策略方面则较为保守，它一直在追求用户体验和服务器带宽负担方面的合理平衡。

3.1.3 云跟踪系统概览

（a）系统架构和索引结构

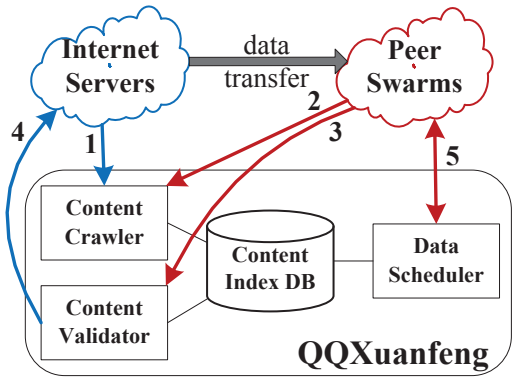


图 3.3　QQ旋风系统架构

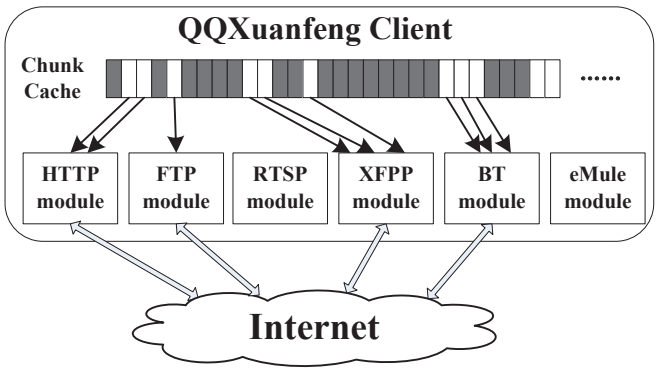


图 3.4　QQ旋风客户端软件。在**数据块缓存（Chunk Cache）**中，数据块为灰色表示已经获得，为白色则表示尚未获得。

如图3.3所示，QQ旋风系统架构由4个基本模块构成：（1）**内容索引数据库（*Content Index DB*）**、（2）**内容爬虫（*Content Crawler*）**、（3）**内容验证器（*Content Validator*）**和（4）**数据调度器（*Data Scheduler*）**，总共使用了53台商品化服务器，且所有服务器都安装了Enterprise Suse Linux 10.1版和gcc 4.1版。QQ旋风的用户需要安装其客户端软件（大约12.5 MB，可以在http://xf.qq.com 下载），此软件同时支持HTTP、FTP、RTSP、BitTorrent、eMule和XFPP内容分发协议（如图3.4所示），其中XFPP协议是QQ旋风专门设计的内容分发协议。安装客户端软件时，用户可以选择安装一个插件到其Web浏览器，从而QQ旋风客户端可以自动接管来自用户Web浏览器的下载任务。

内容索引数据库在QQ旋风系统内处于核心位置，它存储“可下载内容”的索引信息，而这些可下载内容是由内容爬虫探测到的。内容索引数据库使

用MySQL 5.1版软件管理其索引信息，包括每个文件的内容链接、服务器IP地址、内容名称、内容大小、数据块大小、内容类型、MD4散列码、SHA1散列码、“**三段散列码**”和“**分块散列码列表**”等。其中分块大小应该是2的整数次幂个字节——典型地介于32 KB和16 MB之间。对于文件f来说，128位的MD4散列码用来辅助eMule数据传输，因为eMule协议使用MD4散列码来标识内容；类似地，160位的SHA1散列码用来辅助BitTorrent 数据传输；这两者都是由内容的发布者根据文件f的完整内容计算出来的。不同的是，160位的“三段散列码”仅根据文件f的3个数据块计算出来：第一个数据块C_1、中间的数据块$C_{\lceil n/2 \rceil}$以及最后一个数据块C_n，这里n是f包含的数据块总数。具体说来，所谓的“三段散列码”= $SHA1(\ MD4(C_1) \mid MD4(C_{\lceil n/2 \rceil}) \mid MD4(C_n)\ )$，这里‘|’表示异或操作。最后，“分块散列码列表”包含着文件f的每个数据块的MD4散列码。

为什么云跟踪系统要采用很不常见的“三段散列码”？比如说，当内容爬虫发现一个新的eMule链接时，它可以从链接中直接提取对应内容的MD4散列码作为内容标识；而当内容爬虫发现一个新的BitTorrent链接时，它首先从此BitTorrent链接下载数KB大小的.torrent元文件，然后从.torrent元文件中直接提取对应内容的SHA1散列码作为内容标识。然而，当内容爬虫发现一个新的HTTP/FTP/RTSP链接时，在链接中并不包含对应内容的散列码，而让内容爬虫从链接下载整个的源文件明显是不现实的（因为源文件往往很大）。代替地，内容爬虫仅仅下载源文件的3个数据块（第一个、中间一个和最后一个）来计算“三段散列码”作为内容表示，这样所需的下载流量不大、而不同文件的三段散列码碰巧冲突的概率低到可以忽略。综上所述，只要向内容索引数据库提供任意一种类型的内容标识，它都可以找到对应的其它类型的散列码以及所有指向此内容的链接——这正是云跟踪系统独特的内容“映射”功能。

内容爬虫通过在第三方服务器上环游来爬取内容链接（如图3.3中箭头1所示），同时它还接收用户汇报的新链接（如图3.3中箭头2所示）。内容验证器则接收用户汇报的失效链接（如图3.3中箭头3 所示），然后在互联网上检查它们（如图3.3中箭头4所示）。有了内容爬虫和内容验证器，QQ旋风就可以不断地发现新链接、去除失效链接。

数据调度有3个方面的职责：（1）为每个用户集群维护一个用户列表，类似于P2P系统中Tracker的功能；（2）告诉用户他感兴趣的内容可以在哪里获取（服务器或其他用户）；（3）周期性地收集用户汇报以分析他们的工作状态、

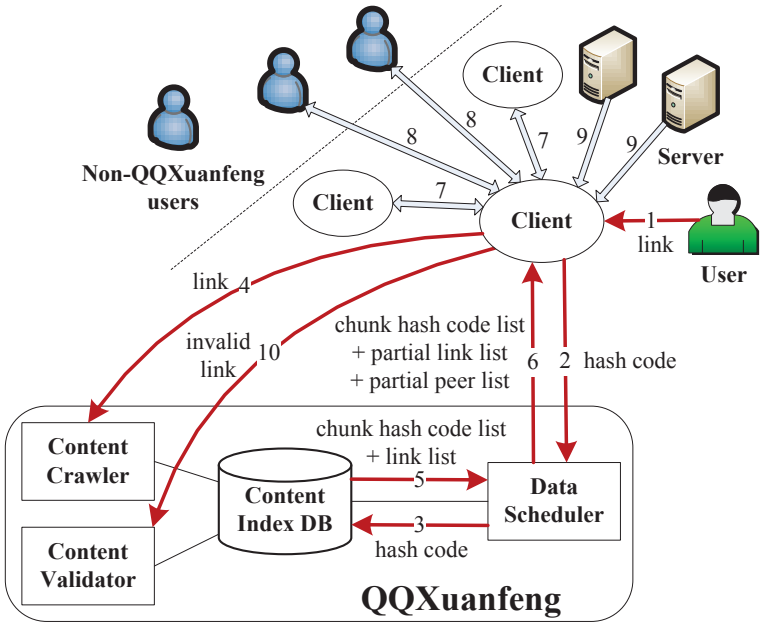


图 3.5　一次典型的用户请求处理过程

计算每台第三方服务器的额外带宽利用率（如图3.3中箭头5所示）。可见数据调度器实际上解决了两方面问题：（1）服务器带宽的受限使用；（2）用户集群的差异化加速。

（b）一次典型的用户请求处理过程

为更深入地理解云跟踪系统架构，图3.5描画了一次典型的用户请求处理过程。首先，用户从客户端（或者通过QQ旋风的Web浏览器插件）输入一个连接（如图3.5中箭头1 所示），当然他也可以输入一组关键词、进而从QQ旋风获取与关键词相关的一组链接。如果输入链接是一个HTTP/FTP/RTSP链接，客户端就首先下载源文件的3个特殊数据块来计算其三段散列码，然后再发送三段散列码给数据调度器（如图3.5中箭头2所示）。类似地，如果输入链接是一个BitTorrent链接，客户端就首先下载对应的.torrent元文件以提取SHA1散列码，然后再发送SHA1散列码给数据调度器。此外，如果输入链接是一个eMule链接，客户端就直接从eMule链接中提取MD4散列码，然后再发送MD4散列码给数据调度器。

数据调度器将客户端提交的散列码进一步提交给内容索引数据库，以找到对应的其它种类散列码和内容链接（如图3.5中箭头3所示）。如果提交的散列

码对内容索引数据库来说是新的，数据调度器将通知客户端把对应的链接提交给内容爬虫进一步处理（如图3.5中箭头4所示）；否则，内容索引数据库就把客户端提交的散列码映射成一个分块散列码列表以及一个（服务器）链接列表（如图3.5中箭头5所示）。

获得分块散列码列表以及（服务器）链接列表后，数据调度器再结合自身维护的信息返回给客户端分块散列码列表、部分链接列表以及对应的用户列表（如图3.5中箭头6所示）。之后，客户端首先尝试同其他用户建立数据连接（如图3.5中箭头7所示），请注意客户端还可以同非QQ旋风用户（这些用户不被QQ旋风监控）交换数据（如图3.5中箭头8 所示）。客户端维护一张“数据块下载图”，其中标明文件有多少个分块、已经下载好了哪些分块，每当发生数据块更新时就把数据块下载图多播给所连接的用户。如果总的P2P下载数据率低于用户的基本期望，客户端将进一步同链接列表中的服务器建立数据连接（如图3.5中箭头9所示）。

QQ旋风客户端采用了一种非常简单的数据块调度策略：任何时刻它都只给每个数据连接分配（去下载）一个数据块，如果这个数据块下载好了，对应的数据连接再被分配另一个数据块。另一方面，如果所分配的数据块（比如说C_i）不能在规定时间获得，相应的数据连接就被终止。此外，如果一个服务器型数据连接是因为上述原因被终止的，那么客户端将向内容验证器汇报此（可能的）“死链”[1]（如图3.5中箭头10所示）。一旦从某个链接获得了数据块C_i，客户端就使用分块散列码列表验证它。如果客户端发现了一个“不一致链接”，它就将此链接汇报给内容验证器（如图3.5中箭头10所示）。数据块的获取优先权取决于用户的工作模式：如果用户选择的是“边下边播”模式，数据块就按播放顺序获取；否则，数据块就按“最少者优先”顺序获取。

3.1.4 挑战性问题和解决方案

（a）处理服务器和内容的动态性

为处理服务器和内容的动态性，我们构建了内容爬虫和内容验证器。内容爬虫通过在第三方服务器上进行深度优先的环游不断地爬取内容链接，同时它还接收用户汇报的新链接。深度优先的环游行为需要使用一个“链接队列”作为其数据结构，内容爬虫收集各种各样的“文件链接”（这里“文件”指的是

[1]“失效链接”分为两类：“死链”和“不一致链接”。“死链”意味着此链接不可访问，而“不一致链接”意味着数据源已经发生改变。

视频、音频、文档、软件等）和非文件链接（比如指向Web网页的链接）放进这个队列。为避免爬取循环链接，最近访问的链接会被缓存用作后来检查，一旦发现有链接重复出现就直接去掉（重复的链接）。（可能的）新文件链接被直接提交给内容索引数据库做进一步处理，比如文件链接过滤和合并。新发现的非文件链接被插进队列尾部进行后续处理。

服务器可能因各种各样的原因加入或离开互联网，同时它们所保存的内容也一直在出现、变化和失效，然而这些内容对应的链接却经常被置之不理（维持原样），从而就变成了失效链接（包括死链和不一致链接）散布在互联网上。这些失效链接会产生至少三个方面的负面效应：（1）给内容索引数据库带来不必要的存储负担；（2）给试图连接数据源的用户带来不必要的网络开销；（3）降低数据调度器的调度性能。为此，内容验证器需要在互联网上不断地验证用户汇报的失效链接：

- 当内容验证器收到一条（针对数据块C_i的）死链l时，它首先尝试从l下载C_i。如果不能下载C_i，l就被认定为死链并从内容索引数据库删除；否则，内容验证器直接告知用户他所汇报的“死链”并非真的死链。
- 当内容验证器收到一条（针对数据块C_i的）不一致链接l时，它首先尝试从l下载C_i并计算C_i最新的MD4散列码。如果最新的MD4散列码和分块散列码列表中保存的现存C_i散列码一样，内容验证器就直接告诉用户他所汇报的“不一致链接”并非真的不一致；否则，l就被认定为不一致链接，而内容验证器继续下载该链接所指向的整个文件，在内容索引服务器中更新或添加对应的索引信息，并告知用户重启他的下载任务。

为检查内容爬虫和内容验证器的实际性能，我们测量了QQ旋风系统（2011年12月）20天内的总链接数、新链接数、死链数和不一致链接数，如图3.6所示。我们观察到：新链接、死链和不一致链接占总链接的比率非常稳定，分别在27%、3.3%和0.6%左右。虽然不一致链接的比率（0.6%）比死链的比率（3.3%）小很多，但它们所带来的负面效应却是极大的——对于一条死链，QQ旋风只需要在内容索引数据库中删掉它即可，但对于一条不一致链接，QQ旋风需要下载整个源文件并告知相关用户重启他的下载任务。

（b）服务器带宽的受限使用

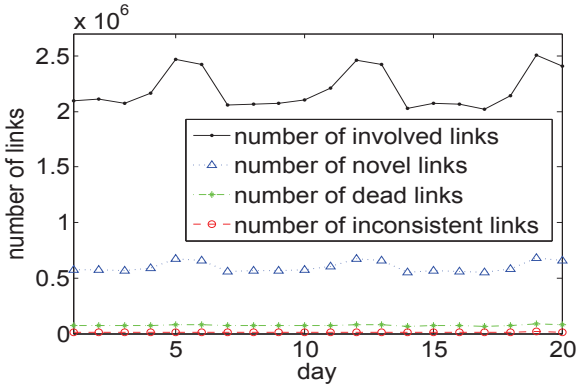


图 3.6　20天内的总链接数、新链接数、死链数和不一致链接数

QQ旋风系统日均跟踪超过100万台第三方服务器，通过智能地引导用户集群从合适的服务器获取数据，实现了对服务器带宽的受限和均衡使用，这样一来，第三方服务器本身支持的功能（也称“原始功能”）就不会受云跟踪的影响。对任意一台服务器S，通过收集和分析相关的用户汇报，QQ旋风可以计算出它的“近似最大带宽”以及被云跟踪系统使用的“额外流量”。鉴于互联网本身的高度动态性，想要获得一台服务器的精确最大带宽极其困难；代替地，我们采用了一种简单的“近似评估方法”，每当内容爬虫发现一台新的服务器S时，在随后的24小时里数据调度器对S的带宽使用不加限制，也就是说它会引导尽可能多的用户从S获取尽可能多的数据，在此过程中S被使用的峰值带宽就被认为是它的“近似最大带宽”。实际上，甚至S的实际最大带宽也可能被其管理员修改，所以上述“近似评估方法”需要周期性地施行，目前的周期设置为1个月左右。

最困难的事情是QQ旋风不能获得服务器的“原始流量”信息，所谓“原始流量”就是服务器用来支撑起原始服务的数据通信流量。为了对服务器原始流量有一个基本的理解，我们通过人为方法从相关管理员处拿到了一组服务器的原始流量信息，这组服务器被称为“采样服务器”，共619台，既包括文件下载服务器，也包括视频流媒体服务器。图3.7画出了这组采样服务器24小时内的平均“原始带宽利用率”（简写为***OBU*，*Original bandwidth utilization***），起始时刻为2011年12月15日的0:00（GTM+8），图中曲线反映出明显的“日差”模式：夜里少、白天多、晚上最多，这和人们平常访问互联网的模式一致。这些服务器的原始带宽利用率通常保持在**60%**以下，而总的平均利用率仅有22%。此外，Heller等人对292台电子商务服务器和一个Google数据中心的测量 [83]也

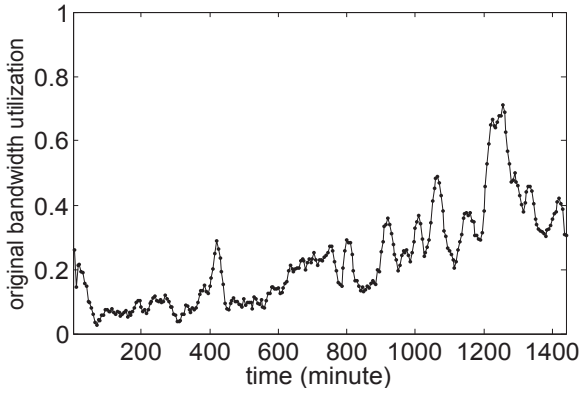


图 3.7　采样服务器24小时内的原始带宽利用率

显示出商用服务器的原始带宽利用率其实并不高（低于25%），这同我们上述观察基本一致。

由上述测量可以看出，对每台第三方服务器，云跟踪的“额外带宽利用率”（简写为***EBU***，即***Extra bandwidth utilization***）最好控制在低于**40%**（=1−**60%**）。具体来说，数据调度器周期性地（周期为5分钟）收集用户汇报来计算每台第三方服务器的额外带宽利用率，如果超过40%，数据调度器就通知相关的部分用户停止从这台服务器下载数据。假设服务器S当前的额外带宽利用率为50%，而S正在给100个用户上传数据，那么数据调度器就通知100个用户中当前下载数据率最高的那20个用户（$20 = 100 \times \frac{50\%-40\%}{50\%}$）停止从$S$下载数据。

此外，数据调度器还通过“混洗”服务器链接列表来平衡服务器负载，对文件f来说，数据调度器维护f的链接列表l_f，此列表是从内容索引数据库获得的。每当数据调度器决定分配一个部分链接列表（其中包含n条服务器链接）给一个用户时，数据调度器首先“混洗”l_f、然后再把l_f中最前面n条服务器链接分配给该用户。当然，如果某个用户的下载速率低于其基本期待，他可以要求数据调度器再次分配他一个部分链接列表。

（c）用户集群的差异化加速

QQ旋风的每个用户当然都希望能获得最佳的体验，也就是说每个用户都希望自己的下载带宽尽可能地高，但通过广泛的测量和分析我们发现：绝大多数用户对其下载带宽其实有一个“基本期待”（d_{basic} = 30 KBps），从而我们获得一个重要的设计启发，就是为用户集群提供差异化的加速。具体来说，用户集

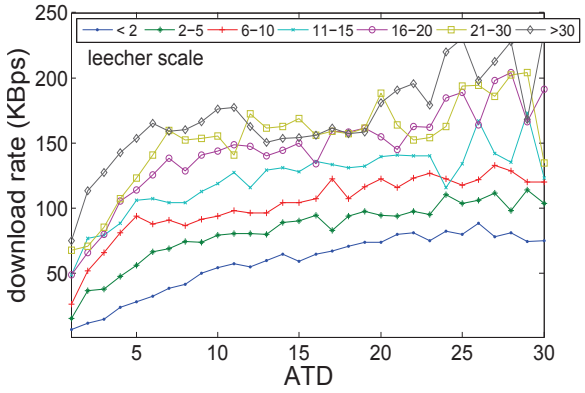


图 3.8　对应不同的下载用户规模的*ATD*和下载速率之间的统计关系

群根据其实时下载速率和数据供需情况被分为3类：(a)**“饥饿集群”**，(b)**“潜在饥饿集群”**以及(c)**“高需求集群”**。不同种类的用户集群对应差异化的加速策略，从而每个用户都能获得高于其基本期待的下载速率。

- 如果一个用户集群的平均下载速率低于基本期待（d_{basic} = 30 KBps），它就被认为是一个“饥饿集群”。
- 如果一个用户集群的平均下载速率高于基本期待，但其“*ATD*”值很低，就被认为是一个“潜在饥饿集群”。***ATD*（*Availability-to-Demand*）**是一个经验性的指标，它表示一个用户集群统计性的数据供需情况，低*ATD*经常（但并不总是）意味着用户集群有潜在的饥饿风险。图3.8画出了对应不同的下载用户规模的*ATD*和下载速率之间的统计关系，其数据来自于对100万个QQ旋风用户集群的测量。相应地，*ATD*的设置也随下载用户规模而变化，如表3.1所示，从而我们使用“ATD of 30 KBps”来识别潜在饥饿集群。
- “高需求集群”分享的文件必须是视频，因为下载视频的用户往往需求高下载速率（d_{high} = 100 KBps）以保持视频的连续播放；并且“高需求集群”还必须包含至少10个用户，因为只有其中包含的用户足够多（一般要大于10）所投入的服务器带宽才值得[1]。

一个饥饿集群会被数据调度器分配一个部分链接列表，其中包含n条服务器链接，以提升该集群的*ATD*到“ATD of 30 KBps”。比如说，如果一个分享文件f的饥饿集群W_h包含8个下载用户和6个种子用户，而数据调度器维护f的链

[1]用户集群中的数据交换能够“放大”所投入的服务器带宽，*放大效应* = $\frac{\text{用户的下载带宽增量}}{\text{所投入的服务器带宽}}$ > 1。一般说来，一个集群的用户规模越大、放大效应也越大。

表 3.1　对应于图3.8的*ATD*设置

Leecher scale	<2	2 - 5	6 - 10	11 - 15	16 - 20	21 - 30	>30
ATD of 100 KBps (d_{high})	26	19	11	5	4	4	2
ATD of 30 KBps (d_{basic})	6	2	2	1	1	1	1

接列表，那么数据调度器将分配$n = 2 \times 8 - 6 = 10$条服务器链接给W_h中的每个用户以加速他们的下载，这里的“2”来自表3.1的第3行第4列。需要注意的是，W_h中不同的用户所分配到的部分链接列表也不同，因为数据调度器每次分配部分链接列表前都要做一次“混洗”。此外，由于*ATD*是一个统计指标，所以很可能某些用户在使用了分配到的部分链接列表后下载速率仍然低于30 KBps，这样的用户可以要求数据调度器再次分配他一个部分链接列表。

对于一个潜在饥饿集群W_p，服务器链接的分配策略同上面类似，不同的是所分配的服务器链接不能被真的用来下载数据，而是当做候选以备不时之需——一旦W_p中的某个用户p的下载速率低于30 KBps，p就可以立即使用它被分配的服务器链接来真正下载数据。一个高需求集群则被分配足够多的服务器链接以提升其*ATD*到“ATD of 100 KBps”。

（d）给第三方带来额外收益

QQ旋风致力于给参与云跟踪系统的各个方面带来利益：服务器提供方、用户集群和QQ旋风自身，因为任何一方的不满都会损害甚至破坏整个云跟踪系统。用户集群从云跟踪系统获得下载带宽的提升，但服务器提供方却不一定愿意支持云跟踪系统、除非他们可以从额外带宽支出中获得额外收益。对服务器提供方来说，最重要的两个收益指标是：***page view* (*PV*)** [77]和***paid-to-click* (*PTC*)** [78]。一般来说，如果一个互联网用户想从一个网站（服务器）获取文件f，他首先需要点击分布在多个页面上的多个链接以获得最终的文件下载链接，在此过程中网站就获得了相应的*PV*值增长；同时网站的*PTC*值也可能有所增长，如果用户在此过程中也点击了广告链接的话。然而，云跟踪将文件f的下载链接直接分配给了用户，这当然给用户带来了便利，但服务器提供者没有获得任何*PV*或*PTC*的增长，也就损失了一定的经济效益。

目前，QQ旋风正在*PV*和*PTC*两个方面加强和服务器提供者的利益协作，以鼓励更多的服务器提供者支持云跟踪。创建QQ旋风和服务器提供者之间的利益协作方案涉及到复杂的技术和经济因素，事实上，并不存在商业上成熟的先

驱供我们效仿，因此我们目前的方案可能是不完备的和过渡性的。一方面，如果一个服务器提供者在QQ旋风的引导下上传文件f给用户集群，那么服务器提供者对应的*PV*值在腾讯“搜搜”搜索引擎中自动增长——“搜搜”目前是中国排名第4的Web搜索引擎，仅次于百度、谷歌和“搜狗”。如果某个服务器提供者对QQ旋风贡献很大，与它相关的网页或者文件链接也会在“搜搜”中排名靠前，这样该服务器提供者对云跟踪系统的带宽贡献就得到了回报。另一方面，QQ旋风也和服务器提供者分享其*PTC*收入。QQ旋风通过在其客户端软件中嵌入广告来获得*PTC*收入，如果某个服务器提供者对QQ旋风贡献很大，它将获得QQ旋风的*PTC*收入的一部分。

3.1.5 性能评估

这一小节评估QQ旋风云跟踪系统的性能，基于大规模的系统测量以及特殊操作的局部性和采用测量。主要的性能指标列举如下：

(1) *对用户集群的加速效果*：代表用户的下载速率被云跟踪系统提升的程度，这是云跟踪的核心性能指标，所以被进一步细化成三个指标：（a）*用户下载数据率分布*,（b）*云跟踪加速效果*以及（c）*加速效果随系统规模的变化*。
(2) *服务器的带宽贡献*：表示用户的总下载带宽中从第三方服务器获得的加速带宽比率。
(3) *服务器的额外带宽利用率（EBU）*：反映云跟踪系统给第三方服务器带来的额外带宽负担。

（a）对用户集群的加速效果

用户下载数据率分布。图3.9描画了QQ旋风系统2011年12月15日的用户下载数据率分布，图中横坐标的“50”表示[40, 50)之一区间，而“500+”则表示[400, $+\infty$) 这一区间。很明显，[100K, 150K)、[150K, 200K)和[200K, 250K)这3个区间拥有最多的用户：16% + 15% + 7% = 38%，超过一半（51.4%）的用户下载速率超过100 KBps (d_{high})。另一方面，仍然存在17.5%的用户下载速率低于30 KBps (d_{basic})，也就是说，超过1/6的用户集群仍然“饥饿”，主要因为QQ旋风不能找到足够多的服务器提供对应的内容——对于这部分用户，我们另外实现了一个新的“云下载”系统 [31]可以为他们提供高质量的内容分发服务。总的来说，大多数用户的下载速率高于他们的基本期待，超过一半的用户

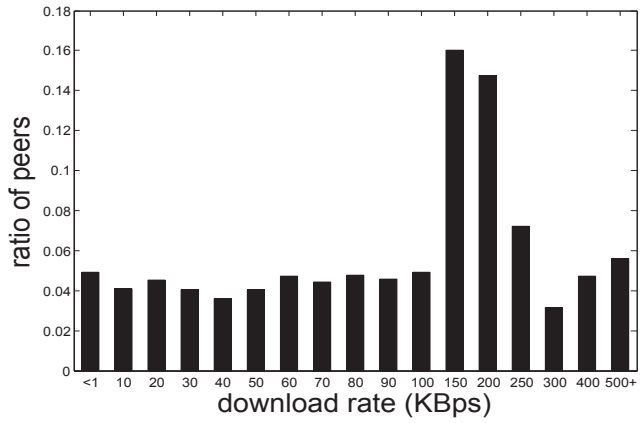


图 3.9　用户下载数据率分布

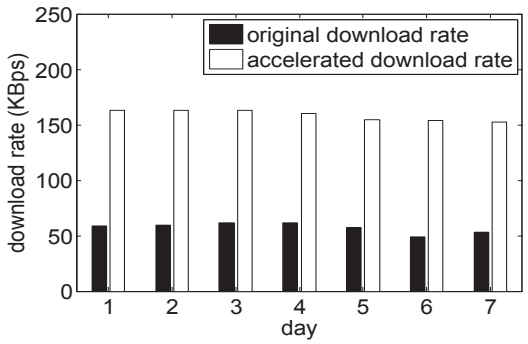


图 3.10　对用户下载速率的加速效果

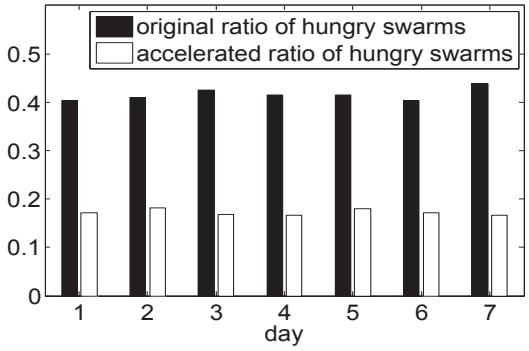


图 3.11　对饥饿集群比例的加速效果

下载速率足够高、能支持连续的视频播放。

云跟踪加速效果。为了评估云跟踪对用户集群W的下载数据率加速效果，我们必须首先停止云跟踪对W的加速、才能测量到W的原始下载数据率，然而这意味着W的工作性能将明显下降，所以我们只能选择停止对一小部分用户集群的加速，且停止时间不能太长，以避免对集群用户体验造成严重损害。出于上述考虑，我们从QQ旋风系统所有的用户集群中随机选择了约1000 个集群进行评估，停止对这些集群加速，一个星期后再恢复加速。所测量到的云跟踪加速效果如图3.10所示：用户平均下载数据率从57 KBps提升到158 KBps (增幅177%)；此外，图3.11记录了对饥饿集群比例的加速效果：该比例从41.6%下降到17.2% (降幅58%)。

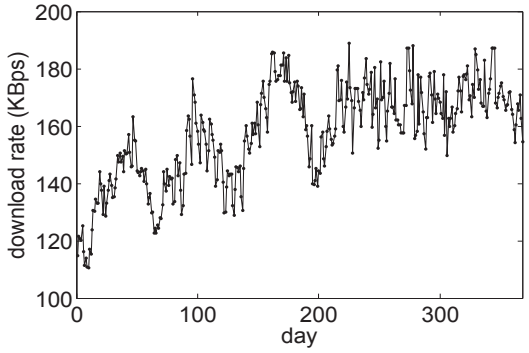


图 3.12　一年中所有的用户日均下载速率

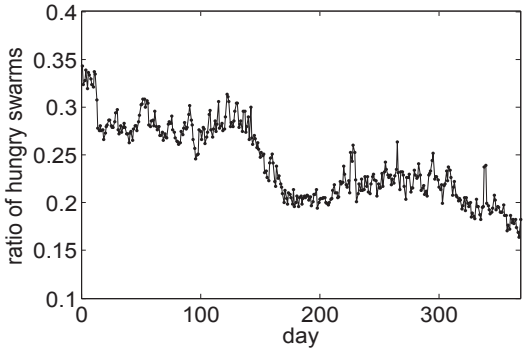


图 3.13　一年中逐天饥饿集群比例变化

加速效果随系统规模的变化。随着时间的推移，QQ旋风从越来越多的第三方服务器收集到越来越多的内容链接、同时服务越来越多的用户集群。比如说，仅在2010年QQ旋风每天就获得大约5千个新用户、索引大约60万条新链接。为研究云跟踪加速效果随系统规模的变化，我们将2010全年中所有的用户日均下载速率记录在图3.12中，可以看到用户日均下载速率从115 KBps提升到165 KBps (增幅43.5%)。此外，图3.13反映出一年中逐天饥饿集群比例从34%下降到17% (降幅50%)。在这一年里，系统硬件架构并未扩容或升级，但基本上工作的很好，可见QQ旋风的加速效果是随系统规模扩展而趋于上升的。

（b）服务器的带宽贡献

QQ旋风的一个重要设计目标是让第三方服务器贡献它们的可用带宽给那些需要的用户集群。这一小节我们测量服务器（对用户集群）的带宽贡献，结果列举在图3.14和图3.15中。一般来说，当用户下载一个小文件（$<$ 100 MB）或处在一个小规模集群（$<$ 20）中时，他主要依靠服务器带宽；反过来，当用户下载一个大文件（$>$ 1 GB）或处在一个大规模集群（$>$ 30）中时，他更多依靠邻居用户。

服务器带宽贡献和用户集群规模之间的关系是直观的，所以这里不再累述，下面我们重点讨论文件大小的影响。从服务器直接获取一个小文件是方便、快捷的，但从用户集群中获取一个小文件则比较麻烦，因为它涉及到诸多P2P操作（比如用户间数据连接的建立和维护）、而这些操作往往需要可观的网络通信开销。此外，大多数用户一旦完成其文件下载就会立刻离开用户集

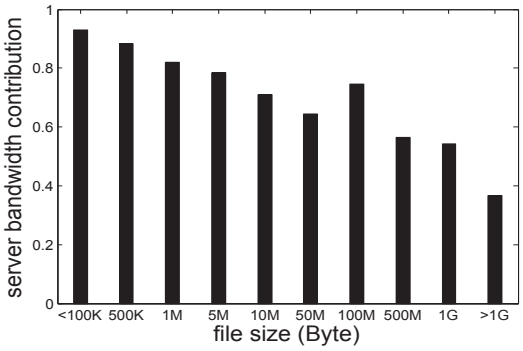


图 3.14　对应不同文件大小的服务器带宽贡献

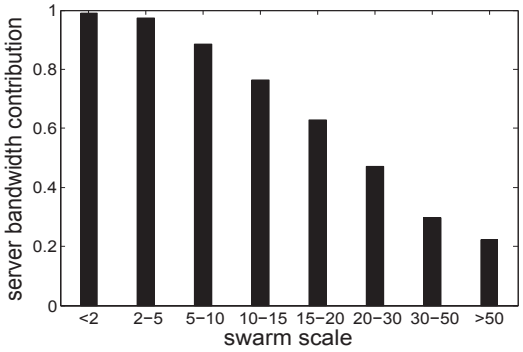


图 3.15　对应不同集群规模的服务器带宽贡献

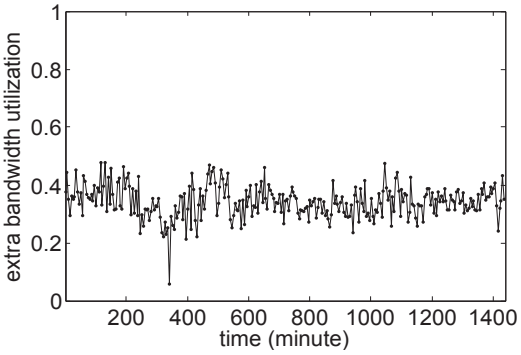


图 3.16　24小时内第三方服务器平均额外带宽利用率

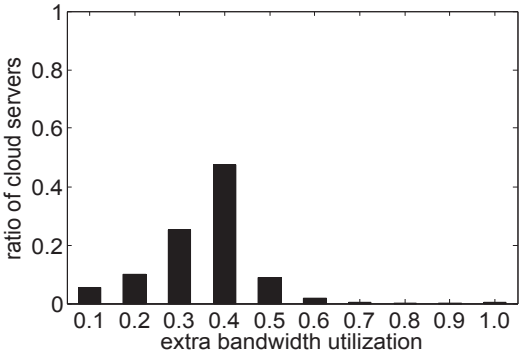


图 3.17　5分钟内第三方服务器额外带宽利用率分布

群，因为对小文件来说更难找到分享的用户。总计起来，45%的用户下载数据率来自服务器的贡献。

（c）服务器的额外带宽利用率

QQ旋风的另一个重要设计目标是平衡第三方服务器的带宽负载，这样它们本身支持的服务就不会被干扰。图3.16记录了2011年12月15日全天所有服务器平均的额外带宽利用率（*EBU*），每隔5分钟记录一次，从GTM+8时区的0点开始。可见在每个5分钟间隔里，平均*EBU*几乎总是低于40%。此外，图3.17中还描画了服务器*EBU*分布在一个5分钟间隔里的“快照”，这里“0.4”表示[0.3, 0.4)区间。在这个5分钟间隔里，88%的服务器其*EBU*低于40%。由于QQ旋风每5分钟收集一次用户汇报信息来计算服务器的*EBU*、然后采取措施限制每台

服务器的*EBU*，因此有少部分服务器暂时过载是正常的。总而言之，图3.16和图3.17确证了QQ旋风系统限制和平衡了第三方服务器的带宽负载。

3.1.6 小结和进一步工作

本文首先回顾互联网内容分发的三种主要模式：直接内容分发、P2P和P2SP。虽然P2SP聚合了前两者的优点、能提供高效的混合式内容分发，但作为一个封闭系统，它仅使用自己的私有云加速自己的私有用户集群，所以在内容丰富程度和服务器带宽上依然受限。因此出现了第四种内容分发模式（也可称为泛化的P2SP模式）即“Open-P2SP”、也称为“云跟踪”，它将互联网上各种各样的第三方服务器、数字内容和数据传输协议聚合到一起，构造成一个大规模、开放和联盟的内容分发体系。基于QQ旋风这样一个大规模商业云跟踪系统，本文研究了云跟踪的挑战性问题、实际系统设计和真实世界性能。

下面一项重要的工作是研究云跟踪系统的参数设置。本文通过广泛测量发掘出诸多有价值的系统参数，比如用户对下载速率的基本期待$d_{basic} = 30$ KBps、服务器额外带宽利用率的限制$EBU = 40\%$、等等。使用这些参数，目前QQ旋风系统基本上工作的很好，但我们并不能断言这些参数是最好的、也不敢说它们能适应未来可能出现的巨大变化（比如底层网络变动、用户行为变化等）。因此，探索如何设计一些机制来自动化地收集和分析云跟踪系统数据、从而动态调整系统参数以匹配新的场景，将是有趣且有用的。

此外，最近我们发现一个潜在的趋势，就是将云跟踪服务聚合到Web浏览器中以“透明地”加速普通Web下载方式（如HTTP/FTP/RTSP），且已确证至少有3个流行的Web浏览器已经实现或者部分实现了上述聚合。毫无疑问，云跟踪和Web浏览器的聚合将给Web用户带来福利，而相关的开发者可以从QQ旋风系统的设计中获得有用的启发。

3.2 最大化带宽放大效应的云调度算法

3.2.1 背景、动机及工作简介

近十年来，网络大规模内容分发越来越普遍，占据了很大比例的互联网流量。今天的大规模内容提供商（如Youtube和Netflix）典型地采用一种基于云的（Cloud-based）内容分发方法，依赖巨大的数据中心进行计算、存储和上传，同时利用地理上广泛分布的CDN（如Akamai和ChinaCache）来提升内容分发性能。这一方法需要重量级的、昂贵的基础设施投入，并非所有公司都有能力买单。反过来，基于P2P的内容分发方法（如BitTorrent和Skype）几乎不需要什么基础设施投入、还能随着用户规模增大而性能自扩展（所谓“用户越多，下载越快”），但缺点是系统成员具有不可避免的高动态性和高异构性、以及对于很多内容找不到种子节点或分享节点，所以P2P的性能时好时坏、很不稳定、也难以预测。

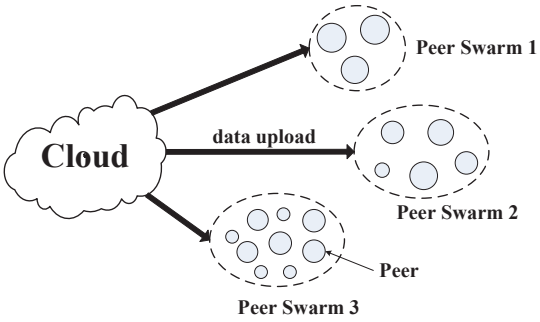
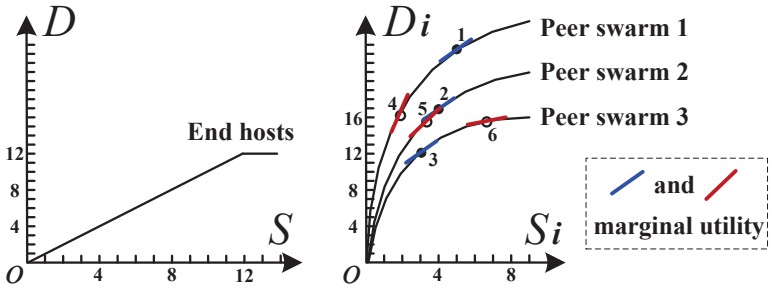


图 3.18　混合式CloudP2P内容分发的基本架构

鉴于上述两种方法各有长短，混合云和P2P的内容分发技术（**CloudP2P**）应运而生，它克服了两者各自的局限性、又继承了双方的优点，表现出良好的系统扩展性和操控性，典型系统如优酷的“i 酷加速器”、土豆的“飞速土豆”、搜狐视频的客户端、PPTV [17]、PPStream、UUSee [84]、迅雷看看、QQ旋风、LiveSky [1]、Novasky [85]和Antfarm [86] 等。如图3.18 所示，CloudP2P 方法通常由一个云（平台）加上一系列的**用户集群（Peer swarm）**构成，云不仅提供内容“种子”、还帮助那些对相同内容感兴趣的用户互相发现以形成数据集群，一个用户集群首先需要从云获取一个“内容种子”，然后才能开始集群内部的数据交换。以优酷为例，在它运营前期带宽



(a) Cloud　　(b) CloudP2P

图 3.19　带宽放大效应（$\frac{D}{S}$）示例图，图(a)对应基于云的内容分发，图(b)对应CloudP2P内容分发。S表示投入的云带宽，D表示用户的聚集下载带宽。对CloudP2P内容分发来说，$S = \sum_i S_i$，$D = \sum_i D_i$，这里i表示第i个用户集群。对每个用户集群，它在特定点的“边际效应”是一个偏导数：$\frac{\partial D_i}{\partial S_i}$，这里我们使用偏导数$\frac{\partial D_i}{\partial S_i}$、而不是直接导数$\frac{dD_i}{dS_i}$，因为$D_i$并不完全依赖$S_i$。

开销占公司总收入的一半以上，但最近两年通过鼓励用户安装“i酷加速器”客户端，优酷的系统架构实际上已经转变为CloudP2P混合结构。

CloudP2P的核心优势在于其**“带宽放大效应”（Bandwidth multiplier effect）**：通过合理分配一部分服务器带宽（S_i）给一个用户集群i，激发集群内部用户（高效地）互相交换数据，CloudP2P可以获得更大的聚集下载带宽（D_i），借用经济学术语，我们将比例$\frac{D_i}{S_i}$定义为用户集群i的带宽放大效应。因此，设计一个CloudP2P内容分发系统的主要问题就是如何最好地分配有限的云带宽、从而最大化系统总体的带宽放大效应——即所谓的**“最优带宽分配问题”（Optimal bandwidth allocation problem、简写为OBAP）**。

下面我们用一个简单的例子形象化地阐述CloudP2P的“带宽放大效应”，同时说明在解决OBAP问题时、为什么云带宽分配的**“边际效应”（Marginal utility**，也是经济学术语）是非常关键的。如图 3.19(a)所示，对于传统的基于云平台的内容分发技术，带宽放大效应始终为1.0，因为用户仅从云端下载数据。而对于CloudP2P来说，带宽放大效应可以远大于1.0，因为用户间的数据交换可以“放大”云端上传带宽的效果。尽管如此，必须注意到CloudP2P的带宽放大效应非常依赖于云端带宽的分配以及每个用户集群的具体情况——分配同样多的云端带宽、不同的用户集群可以达到的带宽放大效应可能很不相同。图3.19(b)描画了3条“带宽放大效应曲线”，每一条对应一个用户集群，假定可以分配的云端带宽总计为$S = 12$、分配方案对应于点1、2和3，那么总的带宽放大

效应为：$\frac{D}{S}=\frac{\sum_i D_i}{\sum_i S_i}=\frac{23+17+12}{3+4+5}=4.33$。

鉴于“带宽放大效应曲线”的非线性特征，在解决上文提出的“最优带宽分配问题”（OBAP）时，我们必须考虑云带宽分配的“边际效应”（$\frac{\partial D_i}{\partial S_i}$）。然而，CloudP2P工业系统中常用的云带宽分配算法几乎没有考虑到边际效应，因此其带宽放大效应并非最优、实际常常很差。比如说，常用的**“自由竞争”（Free-competitive）**分配算法让所有用户自由地竞争云带宽，而**“比例分配”（Proportional-allocate）**算法则根据用户集群规模来正比地分配云带宽。不管采用上面哪种分配算法，都有可能出现“过饱和”的用户集群被分配过多的云带宽从而导致很低的边际效应、而同时“饥饿”的用户集群被分配过少的云带宽从而导致很高的边际效应。比如图 3.19(b)所描画的对应于4、5和6的分配方案，云带宽S 被“比例分配”给3个用户集群，总的带宽放大效应是$\frac{16+15+15}{2+3.3+6.7}=3.83<4.33$（对应点1、2和3的分配方案的带宽放大效应）。从直觉上讲，我们发现更大的带宽放大效应往往对应于更“均衡”的边际效应，即每个用户集群的边际效应趋于一致——下文将形式化证明这个发现。

通过对真实世界大规模测量数据的分析，我们找到了影响用户集群带宽放大效应的关键因素，从而构建了一个细粒度的性能模型来解决CloudP2P内容分发的“最优带宽分配问题”（OBAP）。该模型同时考虑了用户集群外部和内部的带宽“供给”，并且能够和真实测量数据很好地吻合。我们进一步形式化证明了带宽放大效应确实和云带宽分配的边际效应紧密相关。为解决OBAP问题，我们设计了一个快速收敛的迭代算法，在每一步迭代中选择最优迭代方向、自适应设置迭代步长；同常用的迭代算法相比，我们设计的迭代算法具有可以证明的收敛性、更快的收敛速度和更易用性——这一点对于大规模、高动态的CloudP2P系统来说十分重要。上述整个方案（性能模型+迭代算法）合起来称为**FIFA**，即**细粒度的性能模型（Fine-grained performance model）+ 快速收敛的迭代算法（Fast-convergent iterative algorithm）**。

FIFA既在QQ旋风系统数据集（包含了约100万用户数天内的工作情况）上进行了模拟实验，又在CoolFish系统上进行了原型部署实验。模拟实验的结果表明：FIFA的带宽放大效应比现存的带宽分配算法（自由竞争、比例分配和Ration算法 [18]）分别高出20%、17%和8%；而FIFA所带来的额外控制带宽一直保持在15 KBps之下，这实际上比一个普通用户的下载带宽还要低。此外，在CoolFish系统部署的原型实验也证实了FIFA的有效性。

3.2.2 相关工作综述

CloudP2P内容分发常用的带宽分配算法基本上都使用了一个粗粒度的性能模型、其中包含诸多理想化或简化的假设。比如说，大部分商业系统（如PPLive）使用了“自由竞争”算法或其变形，这个算法非常简单：所有用户自由竞争云带宽、争到多少算多少，很明显这种分配算法对于那些贪心的、自私的用户有利，因为这样的用户会同云端建立大量的TCP/UDP连接以争取尽可能多的云带宽。为克服环节“自由竞争”分配算法的缺点，某些系统（如UUSee）使用了“比例分配”算法，它将云带宽按照每个用户集群的规模成比例分配，这就意味着比例分配算法假设了用户集群的云带宽需求只取决于用户规模，但这一假设其实是不争取的。

和FIFA类似，Antfarm [86]使用集中式调度方案来动态分配种子用户的带宽到用户集群中，但Antfarm没有考虑稳定的云（服务器）带宽。Antfarm的实际应用可能会比较困难：首先，种子用户的带宽不像云带宽那样稳定、可靠，也不容易细粒度地分配，所以到目前为止很少见到商业P2P系统这么干的；其次，Antfarm的调度器使用一种集中式的令牌协议来严格控制每个用户的行为（比如邻居的选择和数据的交换），这同P2P的分布式工作原理并不兼容。

和FIFA类似的另一份工作称为Ration [18]，它是专注于P2P在线电视的云带宽分配算法。Ration使用两个影响因子：S_i和l_i来构建其CloudP2P内容分发性能模型，其中S_i表示分配给用户集群i的云带宽，l_i表示用户集群i中的在线下载用户数。由于Ration专注于在线电视环境，而这种环境下大多数用户（观看者）都把带宽主要用在下载上、且一旦他们不看了就会离开用户集群（也就是说$s_i \ll l_i$，其中s_i是用户集群i中的在线种子用户数），所以Ration没有考虑（也不需要考虑）种子用户的影响——即用户集群内的带宽供给。我们提出的算法FIFA则同时考虑了种子用户的影响，并且我们发现它对于构建大规模CloudP2P文件分享系统的性能模型有不可忽视的作用，因为P2P文件分享系统通常比在线电视系统具有更高的用户动态性和异构性，这也正是我们提出的性能模型比Ration更细粒度、更普适的地方。

为解决最优带宽分配问题，Antfarm使用了**“爬山”迭代算法（简称“HC”、即Hill Climbing）**，而Ration使用了**“水流”迭代算法（简称“WF”、即Water Falling）** [87]。这两个迭代算法在解决优化问题方面非常经典，但我们发现当处理大规模、高动态的用户集群时则可能迭代收敛过

慢，主要因为它们的迭代步长过短且恒定。Ration其实已经意识到这个问题，所以它提出一种增量“水流”算法来提升经典“水流”算法的收敛速度，但这种增量方法治标不治本、只起到缓解作用。通过在每一步迭代中寻找最优迭代方向和自适应设置步长，我们提出的快速收敛算法（简称“FA”、即Fast-convergent iterative algorithm）具有更快的收敛速度、且易于使用。

3.2.3 细粒度的性能模型

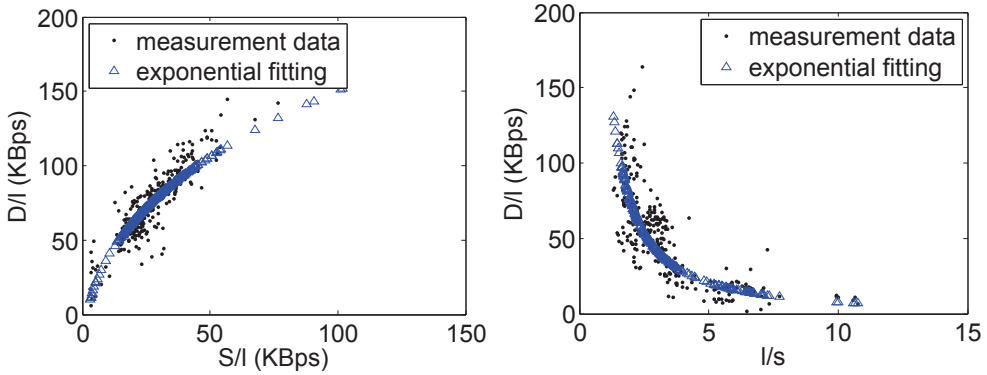
（a）关键影响因素

一个用户集群的“带宽放大效应”可能受很多因素影响，比如所分配的云带宽、下载用户数、种子用户数、每个用户的实际可用带宽、用户间的拓扑连接、相应数据块的分布情况等等。很明显，要建立用户集群的“带宽放大效应”的性能模型，我们不可能考虑所有的影响因素，并且考虑太多细节琐碎的因素也会导致过高的通信和计算开销。代替地，我们的方法是去寻找那些关键影响因素，为此我们使用了QQ旋风系统的测量数据－－QQ旋风是一个大规模CloudP2P文件分享系统 [19, 28]、符合我们所研究的场景。具体来说，我们跟踪了QQ 旋风系统在一天内的1457 个用户集群、涉及大约100万用户，每隔5分钟记录每个集群的工作参数：D_i、S_i、s_i和l_i，然后分析这些参数间的关系。这些参数的含义如下：

- D_i：用户集群i内所有用户的聚集下载带宽。
- S_i：分配给用户集群i的云带宽，它表示集群外部的带宽供给。
- s_i：用户集群i内的在线种子用户数，它表示集群内部的带宽供给。
- l_i：用户集群i内的在线下载用户数，它表示有多少用户是带宽消费者（请注意他们同时也上传数据）。

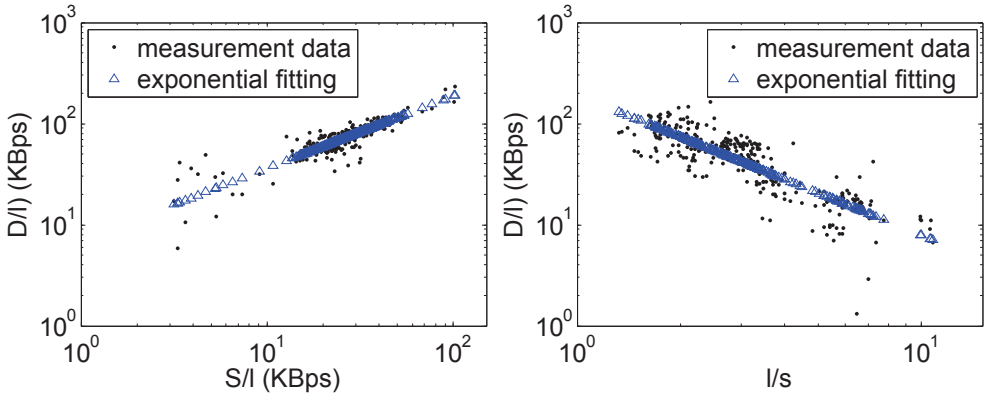
如图3.20(a)所示，我们发现$\frac{D_i}{l_i}$和$\frac{S_i}{l_i}$之间存在近似指数关系，而图3.21(a)中的*log-log*曲线更进一步证实了我们的发现。因此，对于一个典型的用户集群i：

$$\frac{D_i}{l_i} \propto (\frac{S_i}{l_i})^{\alpha_i}, \quad 0 < \alpha_i < 1;$$



(a) $\frac{S_i}{l_i}$ vs $\frac{D_i}{l_i}$. (b) $\frac{l_i}{s_i}$ vs $\frac{D_i}{l_i}$.

图 3.20 对一个典型的用户集群i：(a) $\frac{D_i}{l_i}$和$\frac{S_i}{l_i}$的关系，(b) $\frac{D_i}{l_i}$和$\frac{l_i}{s_i}$的关系。



(a) $log(\frac{S_i}{l_i})$ vs $log(\frac{D_i}{l_i})$. (b) $log(\frac{l_i}{s_i})$ vs $log(\frac{D_i}{l_i})$.

图 3.21 对一个典型的用户集群i：(a) $log(\frac{D_i}{l_i})$和$log(\frac{S_i}{l_i})$的关系，(b) $log(\frac{D_i}{l_i})$和$log(\frac{l_i}{s_i})$的关系。

另一方面，图3.20(b)和图3.21(b)显示：$\frac{D_i}{l_i}$和$\frac{l_i}{s_i}$之间也存在近似指数关系，唯一的区别是这里的“指数”是负的：

$$\frac{D_i}{l_i} \propto (\frac{l_i}{s_i})^{-\beta_i}, \quad \beta_i > 0.$$

上述指数关系基本上同我们在图3.19中的直观经验一致：当一个用户集群被分配太多的云带宽或者含有太多种子用户时，提升其聚集下载带宽的边际效应将非常微弱。然而，由于上述指数关系仍然是“近似”的，就留给我们这样

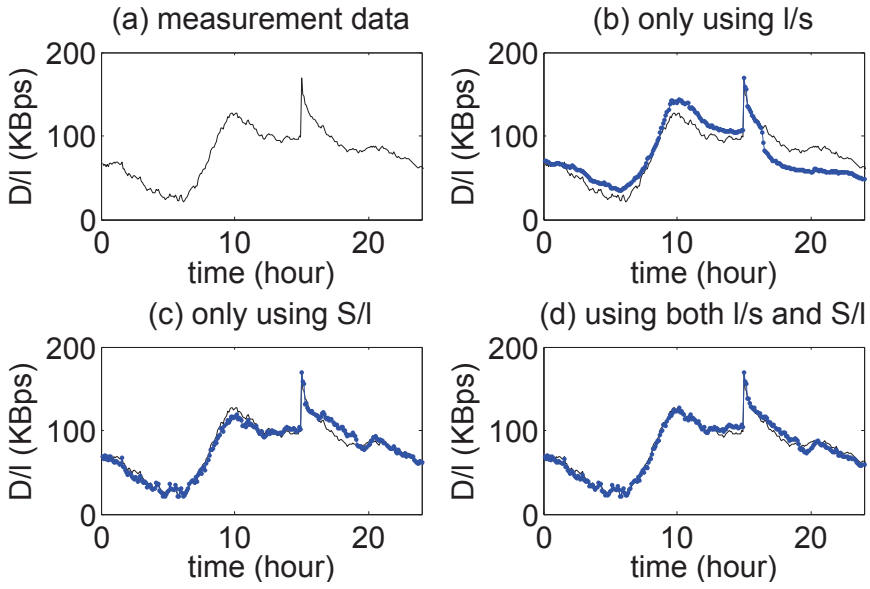


图 3.22　为构建$\frac{D_i}{l_i}$的细粒度模型，使用(b) $\frac{l_i}{s_i}$（即$\frac{D_i}{l_i} = (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i$），(c) $\frac{S_i}{l_i}$（即$\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot f_i$）以及(d) 两者一起用（即$\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i$）。很明显，关键影响因素必须同时包括$\frac{S_i}{l_i}$和$\frac{l_i}{s_i}$，这样模型才能很好地匹配(a)中的测量数据。

表 3.2　把三个模型应用到1457个用户集群数据的相对误差。

模型	平均相对误差	最小相对误差	最大相对误差
(b) 仅使用$\frac{l_i}{s_i}$	0.1391	0.006	0.9036
(c) 仅使用$\frac{S_i}{l_i}$	0.0738	0	0.2366
(d) 同时使用$\frac{S_i}{l_i}$和$\frac{l_i}{s_i}$	0.0308	0	0.0972

一个问题：为了给$\frac{D_i}{l_i}$建立一个细粒度的模型，我们是应该使用$\frac{S_i}{l_i}$、$\frac{l_i}{s_i}$还是两者一起用？为此，我们对每种方案都进行了尝试，而相关的常系数（α_i和f_i）、（β_i和f_i）和（α_i、β_i和f_i）则基于对用户集群i一天的测量数据计算得到。从图3.22和表3.2我们确认：影响“带宽放大效应”的关键因素应该同时包含$\frac{S_i}{l_i}$和$\frac{l_i}{s_i}$，从而得到如下公式：

$$\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i,^{1} \tag{3.1}$$

[1]If $s_i = 0$, we just let $\frac{l_i}{s_i} = 1$ so that $(\frac{l_i}{s_i})^{-\beta_i}$ is ignored.

上式中$0 < \alpha_i < 1$，$\beta_i > 0$，$f_i > 0$。那么用户集群i的聚集下载带宽可表示为：

$$D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i. \tag{3.2}$$

由于S_i是我们唯一可以调度的“决策变量”，所以我们也把D_i写成$D_i(S_i)$，那么用户集群i的“带宽放大效应”可表示为：

$$\frac{D_i}{S_i} = S_i^{\alpha_i-1} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i. \tag{3.3}$$

为计算上面公式中的常系数α_i、β_i和f_i，我们首先将公式(3.1)变形为指数形式：

$$\log \frac{D_i}{l_i} = \log \frac{S_i}{l_i} \cdot \alpha_i - \log \frac{l_i}{s_i} \cdot \beta_i + \log f_i, \tag{3.4}$$

这样α_i、β_i和f_i就可以通过测量D_i、l_i、S_i和s_i、并利用经典的线性回归方法算出来。需要注意的是：这3个“常系数”只是在一段时间里（一般是一天或数小时）可以被“认为”是恒定的，超过时限就需要再次计算。

（b）OBAP问题和它的最优解

基于上一小节的模型，CloudP2P内容分发的**“最优带宽分配问题”（OBAP）**可以形式化为：

OBAP

最大化 系统总的带宽放大效应$\frac{D}{S}$

限制条件:

$$\begin{cases} D = \sum_{i=1}^{m} D_i, \text{其中} m \text{是用户集群数}; \\ S = \sum_{i=1}^{m} S_i, \text{其中} S \text{在一个分配周期可以认为是常数}; \\ S_i \geq 0, \quad \forall i \in \{1, 2, \cdots, m\}; \\ D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i, \quad \forall i \in \{1, 2, \cdots, m\}; \end{cases}$$

决策变量：$S_1, S_2, \cdots, S_m$。

容易看出OBAP是一个受限非线性优化问题 [88]。由于S在一个分配周期可以认为是常数，最大化$\frac{D}{S}$的目标其实就相当于最大化D。当OBAP问题存在最优解时，假定最优解是$\mathbf{S}^* = (S_1^*, S_2^*, \cdots, S_m^*)$[1]，而对应的每个用户集群的聚集下载带宽是$(D_1^*, D_2^*, \cdots, D_m^*)$，那么根据受限非线性优化问题的最优性条件 [88]有：

$$\sum_{i=1}^{m} \frac{\partial D_i(S_i^*)}{\partial S_i}(S_i - S_i^*) \leq 0, \quad \forall S_i \geq 0 \text{ with} \sum_{i=1}^{m} S_i = S. \tag{3.5}$$

然后，固定任意一个i、令j表示其它任意一个下标，可以构造一个在限制集内的合适解$\mathbf{S}'$：

$$S_i' = 0, S_j' = S_i^* + S_j^*, S_k' = S_k^*, \quad \forall k \neq i, j.$$

将$\mathbf{S}'$代入公式(3.5)得到：

$$(\frac{\partial D_j(S_j^*)}{\partial S_j} - \frac{\partial D_i(S_i^*)}{\partial S_i}) \cdot S_i^* \leq 0, \quad \forall i, j \quad (i \neq j).$$

如果$S_i^* = 0$，那么用户集群i得不到云带宽，从而我们也不需要考虑它，这样就得到$\forall i \in \{1, 2, \cdots, m\}, S_i^* > 0$，那么

$$\frac{\partial D_j(S_j^*)}{\partial S_j} \leq \frac{\partial D_i(S_i^*)}{\partial S_i}, \quad \forall i, j \quad (i \neq j). \tag{3.6}$$

所以最优解$\mathbf{S}^*$有如下的形式：

$$\frac{\partial D_j(S_1^*)}{\partial S_1} = \frac{\partial D_2(S_2^*)}{\partial S_2} = \cdots = \frac{\partial D_m(S_m^*)}{\partial S_m}, \tag{3.7}$$

这意味着如果最优解存在，最优解出现的条件是：分配到每个用户集群的云带宽的边际效应恰好相等。在实际的CloudP2P系统中，存在一种例外情况，也就是当一个用户集群i无法被调整到“理想状态”时（所谓“理想状态”，就是说该用户集群的云带宽分配边际效应等同于其它集群的），OBAP问题不存在形如公式(3.7)的最优解。如图3.23所示，由于某些原因用户集群i的聚集下载带宽（图中“Maximum D_i”）有一个上界，它阻止了用户集群i的云带宽分配边际效应被调整到理想状态。这种情况下，我们只能只能分配一定的带宽、使得

[1] **黑色字体**表示一个向量。请注意OBAP问题在其限制集内可能并不存在最优解。

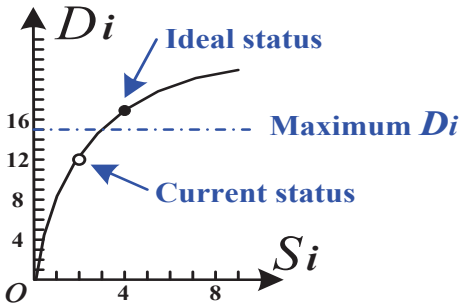


图 3.23　当一个用户集群i无法被调整到其“理想状态”的例外情况

集群i的聚集下载带宽达到“Maximum D_i”，这样所有用户集群的边际效应的“相对偏差”可以尽量小。总结起来我们得到如下定理：

Theorem 1 *对于CloudP2P内容分发，最大带宽放大效应意味着分配到每个用户集群的云带宽的边际效应恰好相等。在实际系统中，我们希望所有用户集群的边际效应的“相对偏差”可以尽量小，也就是说，更大的带宽放大效应等价于用户集群的边际效应更均衡。*

3.2.4 快速收敛的迭代算法

上一节我们将最优带宽分配问题（OBAP）形式化成一个受限非线性优化问题。一般来说，这类问题的最优解是通过多步迭代计算、直到最终计算收敛而获得 [88]，因此，迭代算法的收敛性对于解决OBAP就很关键。

一个迭代算法的收敛性主要取决于两个方面：*迭代方向*和*迭代步长*。对于一个d维受限非线性优化问题[1]，它所有的合适解构成一个d维*迭代空间*$\mathbb{S}^d$，假设迭代算法从一个任一点$\mathbf{P^{(0)}} = (P_1^{(0)}, P_2^{(0)}, \cdots, P_d^{(0)}) \in \mathbb{S}^d$开始迭代，那么在随后的每一步迭代中，算法必须确定一个*迭代方向*和一个*迭代步长*以行进到一个新的迭代点$\mathbf{P^{(k)}} = (P_1^{(k)}, P_2^{(k)}, \cdots, P_d^{(k)}) \in \mathbb{S}^d$，从而$\mathbf{P^{(k)}}$能够离最优点$\mathbf{P}^*$比$\mathbf{P^{(k-1)}}$更近，如图3.24所示。具体来说，整个迭代过程可以形式化为：

$$\begin{aligned} &\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} + t^{(k)}(\overline{\mathbf{P}}^{(k)} - \mathbf{P}^{(k)}), \\ &\text{until} \quad |f(\mathbf{P}^{(k+1)}) - f(\mathbf{P}^{(k)})| < \epsilon. \end{aligned} \tag{3.8}$$

[1] “d维”意味着此优化问题总共处理d个决策变量。对于OBAP 问题，d就是CloudP2P系统中的用户集群数。

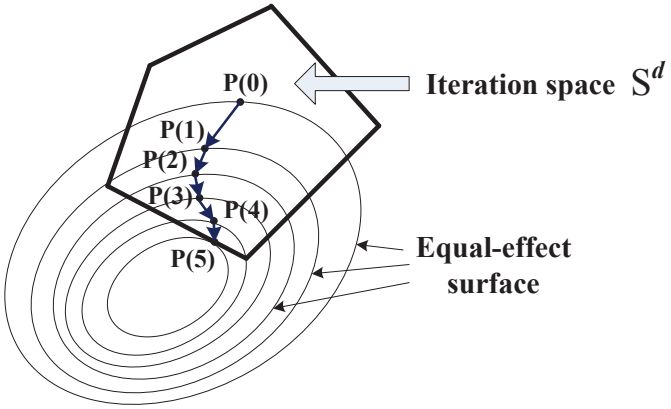


图 3.24　迭代过程示例，图中**“Equal-effect surface”（等效面）**是所有性能函数值$f(\mathbf{P})$相同的点的集合

这里$f(.)$是“性能函数”，ϵ是一个极小的常数（迭代收敛条件），$(\overline{\mathbf{P}}^{(k)} - \mathbf{P}^{(k)})$是迭代方向，而$t^{(k)}$是第$k$步的迭代步长。我们要设计的快速迭代算法（FA）的任务就是要确定第k步迭代中合适的$\overline{\mathbf{P}}^{(k)}$和$t^{(k)}$、让迭代过程能尽量快地收敛。

迭代方向：对于一个非线性优化问题，在第k步直接找到最终迭代方向$\mathbf{P}^* - \mathbf{P}^{(0)}$或$\mathbf{P}^* - \mathbf{P}^{(k)}$通常是不可能的，所以我们的快速迭代算法（FA）使用*条件梯度法* [88]来确定每一步的迭代方向。具体地说，对函数$f(\mathbf{P})$，$f(\mathbf{P}^{(k+1)})$可以通过“泰勒展式”近似为：

$$\begin{aligned} f(\mathbf{P}^{(k+1)}) = f(\mathbf{P}^{(k)}) + \nabla f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T + \\ \frac{1}{2}(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})\nabla^2 f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T + \cdots . \end{aligned} \tag{3.9}$$

其中$\nabla f(\mathbf{X}) = (\frac{\partial f(\mathbf{X})}{\partial X_1}, \frac{\partial f(\mathbf{X})}{\partial X_2}, \cdots, \frac{\partial f(\mathbf{X})}{\partial X_d})$。条件梯度法仅使用一阶泰勒展式来近似$f(\mathbf{P}^{(k+1)})$：

$$f(\mathbf{P}^{(k+1)}) \approx f(\mathbf{P}^{(k)}) + \nabla f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T. \tag{3.10}$$

对于OBAP问题，其维度d就是用户集群数m，因此$\mathbf{P}^{(k)} = \mathbf{S}^{(k)}$, $f(\mathbf{P}^{(k)}) =$

$f(\mathbf{S}^{(k)}) = \frac{D^{(k)}}{S} = \frac{\sum_{i=1}^{m} D_i(S_i)}{S}$并且$D_i(S_i) = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i$。从而有下式：

$$f(\mathbf{S}^{(k+1)}) \approx f(\mathbf{S}^{(k)}) + \nabla f(\mathbf{S}^{(k)})(\mathbf{S}^{(k+1)} - \mathbf{S}^{(k)})^T. \tag{3.11}$$

由于我们的目标是在限制条件$\sum_{i=1}^{m} S_i = S$和$S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$下最大化$f(\mathbf{S})$，所以我们需要在第$k$步迭代中贪心地最大化公式(3.11)中的$f(\mathbf{S}^{(k+1)})$。因此，我们必须找到满足如下问题的特定$\mathbf{S}$：

最大化 $\nabla f(\mathbf{S}^{(k)})(\mathbf{S} - \mathbf{S}^{(k)})^T$

限制条件：
$\sum_{i=1}^{m} S_i = S$和$S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$.

通过扩展$\mathbf{S}$、$\mathbf{S}^{(k)}$和$\nabla f(\mathbf{S}^{(k)})$，我们将上述问题变形为

最大化 $\sum_{i=1}^{m} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}(S_i - S_i^{(k)})$

限制条件：
$\sum_{i=1}^{m} S_i = S$和$S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$.

不难发现上述问题是一个线性优化问题，所以其最优解$\overline{\mathbf{S}}^{(k)}$即为：

$$\begin{aligned} \overline{S}_j^{(k)} = S, \text{对于} j = \arg\max_{i \in \{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}; \\ \text{并且} \quad \overline{S}_i^{(k)} = 0, \forall i \in \{1, 2, \cdots, j-1, j+1, \cdots, m\}. \end{aligned} \tag{3.12}$$

这样我们就得到了第k步迭代的最优方向：

$$\mathbf{d}^{(k)} = \overline{\mathbf{S}}^{(k)} - \mathbf{S}^{(k)}. \tag{3.13}$$

迭代步长：到目前为止，我们已经知道快速迭代算法（FA）的第k步为：

$$\mathbf{S}^{(k+1)} = \mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)}$$

其中$\mathbf{d}^{(k)}$由公式(3.12)和(3.13)来确定。理想情况下，第k步的迭代步长$t^{(k)}$应满足下列条件：

最大化 $f(\mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)})$

限制条件： $\mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)}$是一个合适解。

不幸的是，上述问题依然是一个非线性优化问题，因此不可能直接获得其最优解。代替地，我们使用*Armijo rule* [89]来自适应地设置第k步的迭代步长$t^{(k)}$、以保证$f(\mathbf{S}^{(k+1)})$比$f(\mathbf{S}^{(k)})$至少多出一定限度：

$$f(\mathbf{S}^{(k)} + \tau^j \mathbf{d}^{(k)}) - f(\mathbf{S}^{(k)}) \geq |\sigma\tau^j \nabla f(\mathbf{S}^{(k)}) \mathbf{d}^{(k)^T}| \tag{3.14}$$

其中两个常系数$\tau, \sigma \in (0, 1)$，而变量j按照0, 1, 2, ...的顺序挨个尝试、直到上面的不等式成立，此时的j即为$j^{(k)}$。这样，我们就得到了第k步的自适应迭代步长：

$$t^{(k)} = \tau^{j^{(k)}} \tag{3.15}$$

快速迭代算法总结：快速迭代算法（FA）通过在迭代的每一步中寻找最优方向和自适应设置步长、高效地解决了OBAP问题。首先，由于组合使用了*条件梯度法*和*Armijo rule*，FA的收敛性是可以证明的（参见文献 [88]的Proposition 2.2.1）。其次，FA易于使用，因为所有相关系数、即τ和σ很容易配置。比如说，我们只需要配置$\tau = 0.5$、$\sigma = 0.01$，就适用于3.2.5节和3.2.6节的所有模拟和原型实验。最后，虽然FA算法的精确收敛速度很难理论证明，但FA在3.2.5节的性能评估中展现出近似线性的收敛速度，也就是说，如果一个CloudP2P系统包含m个用户集群，那么FA大概在$\Theta(m)$步中收敛。

水流、爬山和快速迭代算法的对比：由于其简单性和直观性，“水流”算法（WF）是解决受限非线性优化问题的经典迭代算法（被文献 [18]使用）。在每一步迭代中，水流算法仅寻找$\mathbf{S^{(k)}}$的*两个*分量，即$S_h^{(k)}$和$S_l^{(k)}$，满足条件$h = \arg\max_{i\in\{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$和$l = \arg\min_{i\in\{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$。然后水流算法移动固定的长度$\delta$，即从点$S_l^{(k)}$移动到$S_h^{(k)}$：$S_h^{(k)} \leftarrow S_h^{(k)} + \delta$，并且$S_l^{(k)} \leftarrow S_l^{(k)} - \delta$，上述移动看起来就像从一个杯子往另一个杯子灌水。换句话说，水流算法的迭代方向和步长是这样设置的：$\mathbf{d}^{(k)} = (d_1^{(k)}, d_2^{(k)}, \cdots, d_m^{(k)})$, 其中$d_h = 1, d_l = -1, d_i = 0, \forall i \neq h, l$; 并且$t^{(k)} = \delta$。

很明显，水流算法使用总是在两个维度上移动，且其迭代步长（δ）固定。

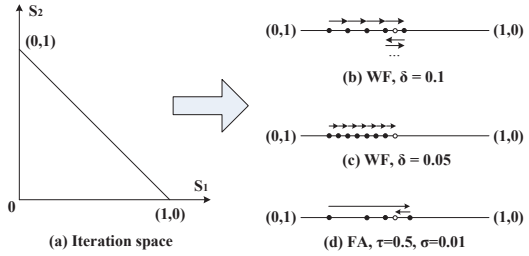


图 3.25　水流算法和快速迭代算法比较：仅有两个用户集群的示例。(a) 迭代空间是直线。(b) 步长设置太大时水流算法不收敛。(c) 步长设置足够小时水流算法迭代7步收敛。(d) 快速迭代算法仅需2步即收敛。

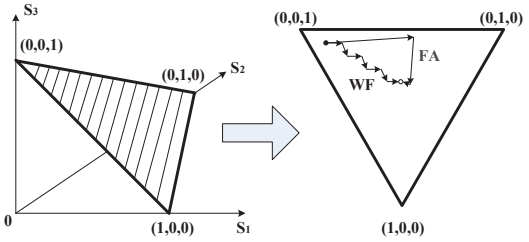


图 3.26　水流算法和快速迭代算法比较：有三个用户集群的示例。水流算法总是在两个维度上移动，而快速迭代算法可以在所有维度上移动。

关键问题在于δ的设置：如果设置太大，水流算法很容易不收敛；反之太小，则收敛太慢。更糟糕的是，当处理大量的高动态的用户集群时，设置一个合理的（即不大不小的）迭代步长会极为困难。因此，对水流算法来说唯一实用的选择是设置一个很小的δ，从而算法迭代步数很多、收敛速度很慢。反过来，快速迭代算法的迭代步长是自适应设置的，所以其迭代步数取决于用户集群数。图3.25是仅有两个用户集群时水流算法和快速迭代算法的比较，这里$S_1 = 0.15$而$S_2 = 0.85$，总的云带宽标准化为$S = 1$。此外，水流算法极为受限的迭代方向进一步降低了其迭代速度，而快速迭代算法可以在所有维度上移动，正如图3.26所展现的。

“爬山”算法（HC）一开始将$\mathbf{S}^{(0)}$的所有分量设为0、而把总的云带宽S存放在一个“仓库”（R）里。在每一步迭代中，爬山算法仅选择$\mathbf{S}^{(k)}$的一个分量，即满足$h = \arg\max_{i\in\{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$的$S_h^{(k)}$，然后从仓库中移动固定的部分$\delta$：$S_h^{(k)}$: $S_h^{(k)} \leftarrow S_h^{(k)} + \delta$, $R \leftarrow R - \delta$，上述移动看起来就像在爬山（每一步都在最陡峭的维度上移动）。容易看出爬山算法其实是水流算法的一个特例：前者每次只在一个维度上移动、而后者是两个维度。因此，在相同步长的前提下，爬山算法的迭代步数通常是水流算法的数倍。

3.2.5 基于真实数据集的模拟实验

（a）数据集

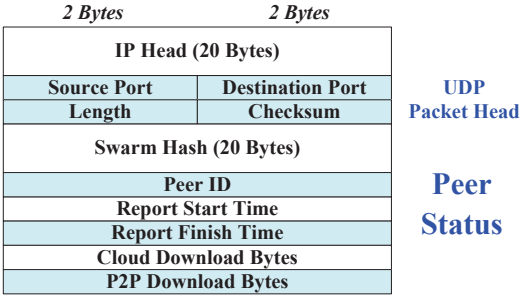


图 3.27 用户状态汇报的数据结构。种子用户的状态汇报中不包含“Cloud Download Bytes”和“P2P Download Bytes”这两个域。

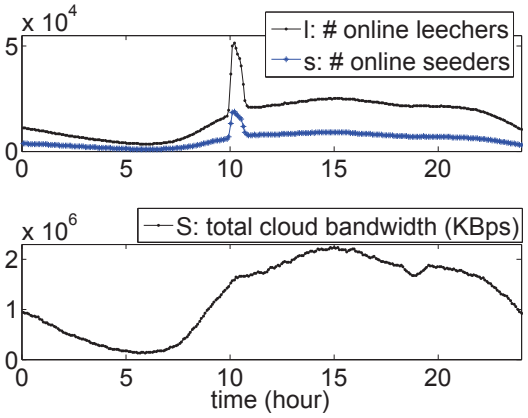


图 3.28 l、s和S随时间的变化

模拟用的数据集来自“QQ旋风”系统，它是一个大规模的CloudP2P文件分享系统，其中每个用户（的客户端）每隔5分钟以UDP方式向系统的“带宽调度器”（一台集中式服务器）汇报其用户状态，所以FIFA的云带宽调度周期也设置为5分钟，用户状态汇报的数据结构见图3.27。在每个调度周期里，带宽调度器需要将用户状态聚合成用户集群的状态（包括S_i、s_i、l_i和D_i）。

模拟实验使用了QQ旋风系统一天内（2011年8月17日）1457个用户集群、包含约100万用户的状态数据集。如图3.28所示，同时在线的下载用户数（l）介于4千和5万之间，而总的云带宽（S）介于0.2到2.25 GBps。对一个用户集群i，为建模其性能（$D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot s_i^{\beta_i} \cdot f_i$）所需要的常系数（$\alpha_i$、$\beta_i$和$f_i$）是用其一天内的所有状态信息计算出来的，包括288条集群状态信息、其中$288 = \frac{24\text{小时}}{5\text{分钟}}$)。获得每个用户集群的性能模型之后，我们模拟了自由竞争算法（*free-competitive*）、比例分配算法（*proportional-allocate*）、*Ration*算法和我们提出的*FIFA*算法来重新分配系统云带宽，同时观察它们的*带宽放大效应*、*边际效应*等等。

（b）性能指标

- *带宽放大效应*定义为$\frac{D}{S}$，其中D表示端用户剧集下载带宽，S表示所投入的云带宽。带宽放大效应越大，意味着云带宽越被高效利用（来加速用户间的P2P 数据交换）。
- *边际效应*定义为$\mu_i = \frac{\partial D_i}{\partial S_i}$（对用户集群$i$）。在**Theorem 1**中已经证明：对

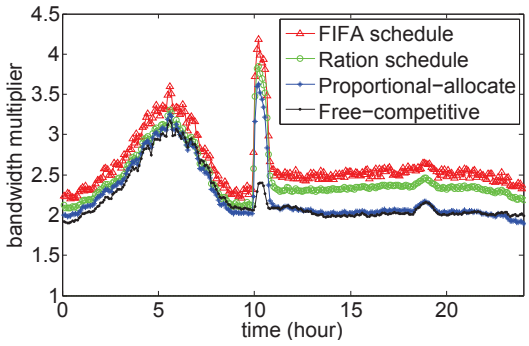


图 3.29 带宽放大效应随时间的变化

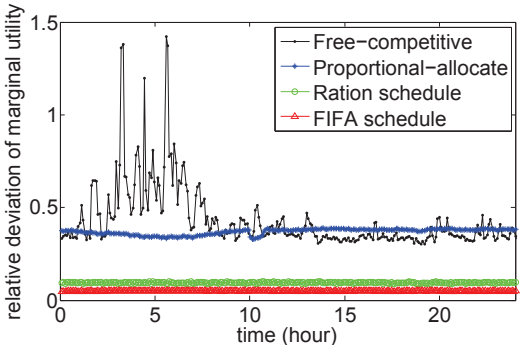


图 3.30 边际效应的相对偏差随时间的变化

于CloudP2P内容分发，最大带宽放大效应意味着分配到每个用户集群的云带宽的边际效应恰好相等。在实际系统中，我们希望所有用户集群的边际效应的“相对偏差”（$dev_\mu = \frac{\sum_{i=1}^{m}|\mu_i-\bar{\mu}|}{m\cdot\bar{\mu}}$）可以尽量小。

- *收敛速度*表示迭代算法解决OBAP问题的迭代步数，此外我们还关心迭代算法的易用性。
- *控制开销*表示带宽分配算法给系统带来的额外通信开销，因为带宽分配算法需要收集用户汇报的状态信息。

（c）模拟实验结果

带宽放大效应。 图3.29描画了带宽放大效应随时间的变化，可以看到比例分配算法的带宽放大效应和自由竞争算法很接近，而*FIFA*和*Ration*则明显优于自由竞争算法，*FIFA*的提升为：$2.20 \rightarrow 2.64 = 20\%$，*Ration*的提升为：$2.20 \rightarrow 2.45 = 11\%$，也就是说，*FIFA*的带宽放大效应比自由竞争算法、比例分配算法和*Ration*分别高20%、17%及8%。

边际效应。 边际效应提供了对带宽放大效应的微观解释——较大的带宽放大效应意味着较小的边际效应“相对偏差”（dev_u）。图3.30描绘了所有用户集群的边际效应相对偏差，可以明显看出边际效应相对偏差和和带宽放大效应之间有紧密的“负相关”关系——*FIFA*的dev_u最小，因此它的带宽放大最大。

收敛速度。 我们在3.2.4节理论分析了爬山算法、水流算法和快速迭代算法的收敛速度，这里则检验它们的实际收敛速度。我们使用了不同数量的用户集

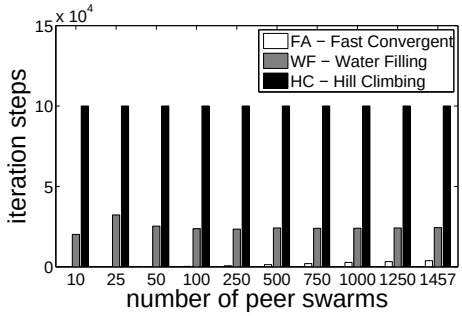


图 3.31 快速迭代算法（τ=0.5, σ=0.01）、水流和爬山算法（δ=0.00001）的收敛速度

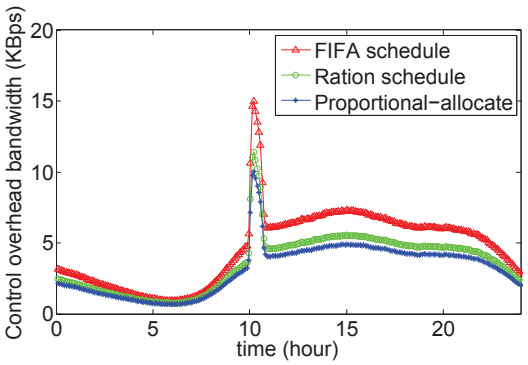


图 3.32 控制带宽开销随时间的变化

群的状态数据：10, 25, . . . , 1250, 1457。对于快速迭代算法，我们仅使用简单的参数$\tau = 0.5$和$\sigma = 0.01$，就适用于所有的实验规模。然而，对于爬山算法和水流算法，我们需要多次尝试才能找到一个合理的步长δ、以使得爬山算法和水流算法能够对所有实验规模都收敛。最终，我们发现$\delta = 0.00001$是一个合适的参数，将对应的收敛速度描画在图3.31中。这里我们主要有两个发现：（1）随着集群规模增加，快速迭代算法展现出近似线性的收敛速度（$\Theta(m)$），且其收敛速度明显快于爬山算法和水流算法。（2）爬山算法和水流算法展现出近似常数（$\Theta(\frac{1}{\delta})$）的收敛速度，且δ越大收敛越快，但不收敛的风险也会增加；反过来，快速迭代算法的收敛性对参数并不敏感，所以更易于使用。

控制开销。自由竞争算法的控制开销为0，因为它根本不需要收集用户状态信息。图3.28记录了一天内每隔5分钟的在线下载用户数和在线种子用户数，这样就很容易计算出*比例分配*、*Ration*和*FIFA*分配算法的控制开销。对于*比例分配*算法来说，它的用户汇报不需要“Cloud Download Bytes”和“P2P Download Bytes” 这两个域（如图3.27所示），并且它只收集下载用户的状态汇报；而*FIFA*则同时收集下载用户和种子用户的状态汇报；由于*Ration*不考虑种子用户，所以它也只收集下载用户的状态汇报。从图3.28我们算出*比例分配*算法收集了452万条下载用户状态汇报（不包含“Cloud Download Bytes”和“P2P Download Bytes” 这两个域），总计$4.52M \times 60\text{B} = 271$ MB；*Ration*收集了452万条下载用户状态汇报，总计$4.52M \times 68\text{B} = 307$ MB，而*FIFA*收集了6.05万条用户状态汇报，总计$4.52M \times 68$ Bytes $+ 1.53M \times 60\text{B} = 399$ MB。将总的状态汇

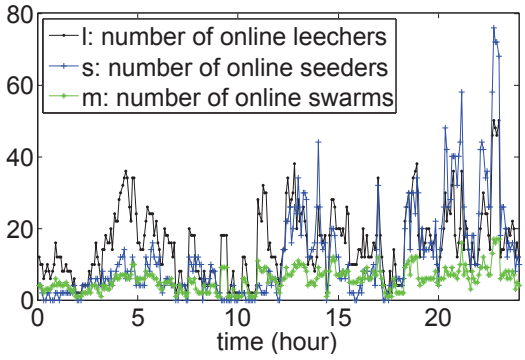


图 3.33 CoolFish系统的l、s和m随时间的变化

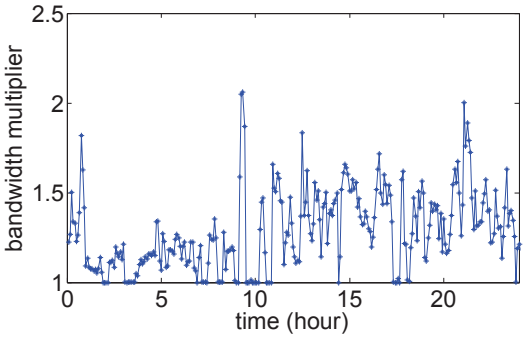


图 3.34 CoolFish的带宽放大效应随时间的变化

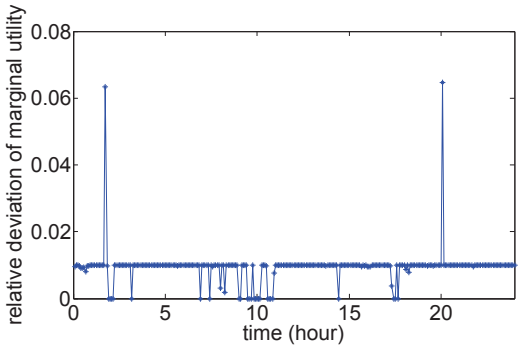


图 3.35 CoolFish系统的边际效应相对偏差随时间的变化

报开销平摊到每一分钟，我们就可以画出*FIFA*、*Ration*和*比例分配*算法的*控制带宽开销*，如图3.32所示。很明显，*FIFA*的控制带宽开销始终低于15 KBps——这比一个普通用户的下载带宽还要低。

3.2.6 原型系统实现

除了上一节的模拟实验，我们还在“CoolFish” [90]系统之上实现了FIFA算法的一个小规模原型。CoolFish是一个CloudP2P视频点播系统，它主要部署在“中国科研网”（CSTNet） [91]内，服务中科院和其他一些研究所的师生员工。CoolFish由两部分组成：一个由4台服务器构成的“微型云”，和较多用户构成的P2P用户集群。CoolFish能够支持超过700 Kbps的平均视频播放码率，这比一般的商业系统高大约50%。

图3.33描画了CoolFish系统一天内的在线下载用户数(l)、在线种子用户数(s)

和在线用户集群数(m) 随时间的变化。很明显，CoolFish的用户规模比QQ旋风小得多，尤其是单个集群内的平均用户数($\overline{p}$)，因此CoolFish系统总体的带宽放大效应也比QQ旋风更低且更不稳定，如图3.34所示。应用FIFA算法后，QQ旋风的带宽放大效应介于2.25到4.2之间，而CoolFish的带宽放大效应介于1.0到2.1之间。虽然后者的带宽放大效应看起来不够高，但FIFA的效果仍然可以从边际效应相对偏差（dev_u一直保持在1% 左右，如图3.35所示）得到证实，因为我们已经证明了边际效应相对偏差很低时、带宽放大效应接近最大。

3.2.7 小结和进一步工作

作为一种混合内容分发模式，CloudP2P继承了基于云的和P2P的内容分发的有点，从而为未来大规模互联网内容分发提供了一个值得期待的前景选择。本文研究了CloudP2P内容分发的最优带宽分配问题（OBAP），其目标是最大化带宽放大效应。基于大规模真实世界测量，我们为解决OBAP问题构建了一个细粒度的性能模型，并证明了带宽放大效应同云带宽分配的边际效应密切相关，还提出了一种快速迭代算法来解决OBAP问题。模拟实验和原型实验的寄过都证实了我们提出的方案的有效性。

对于CloudP2P内容分发，本文关注其带宽放大效应和对应的微观方面、即边际效应。事实上，对于某些特殊场景的CloudP2P内容分发来说，我们还需要考虑*用户满意度*或者*用户集群优先级*。比如说，下载速率30 KBps对文件分享用户来说可能没问题，但高达300 KBps的下载速率对高清视频播放用户来说却可能很不够。因此，虽然我们提出的FIFA算法已经为整体的CloudP2P系统实现了最大带宽放大效应，但这并不代表最大的用户满意度。未来工作的困难在于我们需要同时考虑多个性能指标：带宽放大效应、用户满意度等等，而多个性能指标间也会有冲突，所以如何权衡也值得思考。

第四章　完全依赖云的内容分发

4.1 服务冷门视频分发的云下载系统

4.1.1 背景、动机及工作简介

视频内容占据了互联网数据流量的主要部分，思科公司的一份报告 [92]中提到：2012年接近90%的消费者IP数据流量来源于视频内容分发，包括Web视频（例如YouTube）、P2P视频（例如BitTorrent和PPLive）等等。因此，实现“**高质量**”的互联网视频内容分发对于学术界和工业界都有重要意义。这里“高质量”的含义包括两个方面：“**高数据健康度**”和**高数据传输率**。“数据健康度”这一性能指标原本使用在BitTorrent协议中，代表一个BitTorrent用户集群中所分享的文件的完整副本数目，比如说数据健康度1.0意味着有用户集群中存在一个完整的副本，而当数据健康度低于1.0时该用户集群则被称为“不健康的”，因为集群中没有用户可以获得一个完整的副本。本文我们沿用数据健康度这一概念来表示一个视频文件的数据冗余程度：高数据健康度意味着用户能够获取一个完整的视频副本。此外，高数据传输率则使得在线视频流媒体（包括实时视频流媒体和视频点播流媒体）的功能成为可能。

当前主流的视频内容分发技术主要是CDN和P2P。CDN是最经典的技术，它通过在多个地点（通常跨越多个ISP）策略性地部署边缘服务器来优化互联网内容分发的性能。这些边缘服务器互相协作、根据数据流行度和服务器负载来复制和移动数据，从而端用户可以从其邻近的边缘服务器获取数据副本，这样数据传输率就有很大提升、并且原始数据源的负载也极大减轻。虽然如此，由于CDN有限的存储和带宽容量，让CDN来复制冷门视频到边缘服务器 [93]的性价比很低：我们知道互联网上存在的冷门视频比流行视频的数量要多得多，因此只能把CDN有限的存储和带宽容量用来服务流行视频。此外，CDN是一种付费设施，它只服务那些付费的互联网内容提供商，而不是一个公共的互联网设施。综上所述，指望CDN来高效地分发冷门视频是不切实际的。

不同于CDN，P2P内容分发主要依靠不稳定但是数量众多的端用户，这些端用户互相形成数据集群，在一个集群中邻居用户间直接交换数据。P2P真正的威力体现在流行视频的分发上，因为流行视频被众多用户分享、而用户多通常

意味着较高的数据健康度和并行下载度，它们进一步带来了高数据传输率。对于一个冷门视频，通常很难找到一个对应的用户集群；即使找到了，集群中稀少的用户也不太可能具有高数据健康度和高数据传输率，因此集群中每个用户都必须保持长时间在线才能等到视频完整下载——这是一个冗长而低能效的过程。总而言之，虽然CDN和P2P一般可以很好地分发流行视频，但却因为低数据健康度和低数据传输率而不能为冷门视频提供满意的内容分发服务。

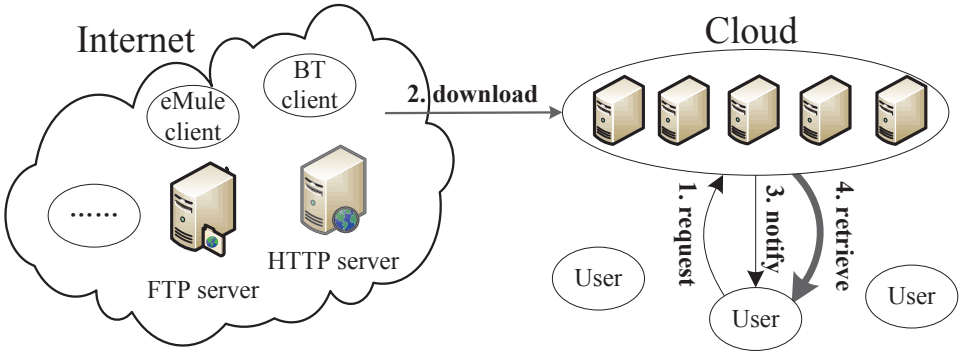


图 4.1　云下载的基本原理

最近几年来，云设施在世界范围内的广泛部署为我们考虑上述问题提供了一条新的思路：在本文中，我们提出和实现了**云下载**方案，通过使用云设施来保证数据健康度和提升数据传输率，从而提供高质量的互联网视频内容分发服务。云下载的基本原理如图 4.1所示：用户首先发送其视频请求到云端（见图 4.1中箭头1），视频请求中包含一个文件链接和其它信息，文件链接可以是一个HTTP/FTP链接、一个BitTorrent/eMule链接、等等。随后，云端在互联网上从此文件链接下载用户请求的视频并将视频存储在云缓存中（见图 4.1中箭头2）以保证数据健康度不低1.0（每个视频在云缓存中保存有一个副本以作为冗余）。然后云端通知用户他请求的视频已经下载完成（见图 4.1中箭头3），用户通常可以在任何时间任何地点以高数据率从云端获取其请求的视频（不管该视频是否流行）（见图 4.1中箭头4），这是依靠云端内部的数据传输加速技术实现的。在实际系统构建中，当云端下载好视频时并不一定非要通知用户，也可以让用户自己在上线时主动检查视频下载进度以采取相应措施；也就是说，图 4.1中的箭头3也可以替换成另外一个箭头“3. 检查”。

云下载带来了一个重要的衍生优势：用户端的**“能效”（Energy efficiency）**。互联网用户经常需要让其电脑（包括网卡）保持长时间在线（假设在线时长为t_1）、仅仅为了下载一个冷门文件 [94]。在此过程中，大量

的能量被浪费了，因为其电脑的主要构件（包括处理器、内存、磁盘、等等）也一直处于工作状态却未被高效利用。云下载使用运服务器代替用户上网下载所请求的视频，因此用户不需要在线；实际上，他们根本就不需要打开电脑——这正是我们也称云下载为“离线下载”的原因。当云端下载完成所请求的视频时，用户通常可以在短时间内（假设时长为t_2）获取视频；从而，云下载极大地减少了用户端的能耗（节能比例为：$\frac{t_1-t_2}{t_1}$）。

云下载唯一的缺点在于：对于某些视频，其用户必须等待云端下载完成视频、因而他不能立刻播放它，这段等待的时间就称为**播放启动时延**。应当注意到：即使是CDN和P2P也有它们的播放启动时延以缓冲一定量的数据来平滑视频播放的启动阶段（比如为一个100分钟的电影缓冲几分钟的数据）。我们通过“隐式和安全的用户间数据复用”有效地缓解了这一缺点：对每个视频，仅在它第一被用户请求时云端才会从互联网中下载它，此后对于该视频的所有请求都使用缓存的副本直接满足、从而播放启动时延极低（除非该视频的缓存副本后来被替换掉了）。上述隐式的数据复用比P2P用户间显式的数据复用（容易遭受Sybil攻击 [95]、Eclipse攻击 [96]、DHT攻击 [97]等等）要安全得多，因为它完全发生在云端、对用户不可见。根据真实部署系统的性能数据，用户间数据复用率达到了87%，这意味着大多数视频请求会立刻得到满足（称为“下载秒杀”）、其播放启动时延几乎为零。

从2010年6月起，我们开始在“QQ旋风”平台上部署一个大规模的商业云下载系统（称为“VideoCloud”），它使用了一个由426台商品化服务器构成的“微型”数据中心。截止2011年6月，VideoCloud 已经吸引了超过6百万注册用户，并支持大多数主流的互联网内容分发协议，比如HTTP、FTP、BitTorrent、eMule、磁性链（Magnet link）等等，每天接收到来自5万用户的超过20万视频请求，其中大多数请求的是冷门视频。VideoCloud的数据中心部署在在中国的4个主要ISP中，并计划覆盖更多的ISP。用户为了获取云下载服务所需支付的费用不同于Amazon S3和Microsoft Azure，而更接近于Dropbox：VideoCloud的用户是根据其云端存储容量来付费、不考虑带宽开销，每个注册用户分配5 GB的免费空间，额外空间则以5 GB为单位来付费。

VideoCloud系统的实际运行日志确证了云下载方案的有效性。比如说，用户的冷门视频平均数据传输率超过1.6 Mbps，并且超过80%的数据传输率高于在线视频的基本播放码率300 Kbps。相比之下，当使用“一般下载方案”时，

用户的平均数据传输率仅有0.57 Mbps，并且70%的数据传输率低于300 Kbps，这里“一般下载方案”指的是用户从互联网下载视频内容的一般方式，比如使用Web 浏览器或P2P下载软件。

最后，本文的贡献可大致总结如下：

(1) 分析了当前视频内容分发的主流方案（即CDN和P2P）并发现两者皆不能为冷门视频提供满意的内容分发服务。受此问题驱动，我们提出和实现了云下载方案，通过使用云设施来保证数据健康度和提升数据传输率，从而提供高质量的互联网视频内容分发服务。云下载唯一的缺点在于播放启动时延，我们通过隐式和安全的用户间数据复用有效地缓解了这一缺点。
(2) 我们部署了VideoCloud，一个大规模商业化云下载系统。我们还讲述了其系统架构和设计技术，以及我们的观察、分析和启发。我们的研究为利用云设施来实现高效的互联网服务提供了实践性的经验和有价值的启发。
(3) 通过系统运行日志，我们依据三个主要指标：数据传输率、播放启动时延和能效评估了VideoCloud的性能，评估结果确证了云下载方案的有效性。

4.1.2 相关工作综述

随着互联网骨干带宽和端用户接入带宽的持续提升，网民对视频内容分发提出的要求越来越高。视频像素解析度从CIF (common intermediate format, 352*288)进化到VGA (video graphics array, 640*480)、HD (high-definition, 1280*720)，同时视频播放模式从“先下后看”进化到“边下边看”。在过去的15年里，高质量的视频内容分发在工业界和学术界一直是一个非常热门的研究课题 [98]。前面我们已经分析过CDN、P2P和云下载各自的优缺点，而在最近几年里，很多研究者已经认识到CDN和P2P都有他们各自的局限性、从而提出了组合和优化CDN和P2P的新型内容分发方案。

基于中国最大的商业CDN服务平台ChinaCache，尹浩等人开发了一个混合式的CDN-P2P实时流媒体系统（名为LiveSky [1]）以整合双方的优势：P2P的可扩展性和CDN的分发质量控制能力及灵活性。虽然混合式的CDN-P2P架构不可避免地带来了额外的部署复杂性和维护开销，但LiveSky减少了CDN网络的开销、并使得流行视频的P2P流媒体工作效率升高。虽然如此，对于冷门文件来说，其用户往往太少而不足以形成具备规模的高效率的用户集群，从而LiveSky的分发性能可能会类似于普通CDN。

吴川等人重新聚焦于P2P流媒体系统中服务器的重要性，提出了服务器辅助的P2P流媒体方案 [18]。通过分析一个大规模商业P2P流媒体系统（名为UUSee [84]）7个月的运行数据，她们发现系统所提供的150台流媒体服务器的总上传带宽不能满足几百个电视直播频道的不断上升的下载带宽需求（一个电视直播频道可以看成是一个P2P用户集群），尽管用户的总上传带宽也伴随下载需求的上升而增加。因此，她们提出了一种称为“Ration”的服务器带宽分配算法，该算法能够基于历史信息前动预测每个电视直播频道的最低服务器带宽需求，这样就能够保证每个电视直播频道具有比较理想的流媒体质量。Ration和LiveSky有相似的缺点，因为UUSee的每个电视直播频道都可以看成是一个非常流行的视频，但对于冷门视频来说，Ration的执行效果将类似传统的客户/服务器（C/S）视频流媒体方案。

最近几年，我们可以看到IT业界对极端高清视频内容分发（比如在线高清电视和影院质量的视频点播）的需求越发强烈。毫无疑问，极端高质量的视频可以提供互联网用户优秀的观看体验；然而，在高动态性的互联网环境下进行极端高质量视频的内容分发（尤其是那些没有CDN支持的冷门视频）一直是一个挑战性问题。最新的实际解决方案可能是Novasky [85]，一个实用的服务于高带宽网络（实际上是清华大学校园网）的影院质量视频点播流媒体的“P2P存储云”。Novasky的局限性非常明显：它工作于一个高带宽的校园网络，工作环境比真实的互联网要稳定和同构得多。云下载使用云设施来保证数据健康度，同时提升数据传输率到相当高和稳定的水平，因此云下载可以看成未来互联网极端高清视频内容分发的一个充满希望的可能性方案。

4.1.3 云下载系统概览

如图 4.2所示，VideoCloud系统架构主要由5个模块构成：（1）**ISP代理（*ISP Proxies*）**，（2）**任务管理器（*Task Manager*）**，（3）**任务分发器（*Task Dispatcher*）**，（4）**下载机集群（*Downloaders*）**和（5）**云缓存（*Cloud Cache*）**。每个模块的硬件配置如表 4.1所示，需要注意的是所有的内存、存储和带宽信息都指的是一台服务器。我们没有列出CPU信息，因为CPU性能对VideoCloud这样一个数据（即带宽和存储）密集型系统来说并不重要。整个系统使用了426台商品化服务器，包括300台数据块服务器（构成一个600 TB的云缓存）、80台下载服务器（共使用26 Gbps互联网带宽）和33台上传服务器（共使用20 Gbps互联网带宽）。每台服务器运行Suse Linux v10.1 操作

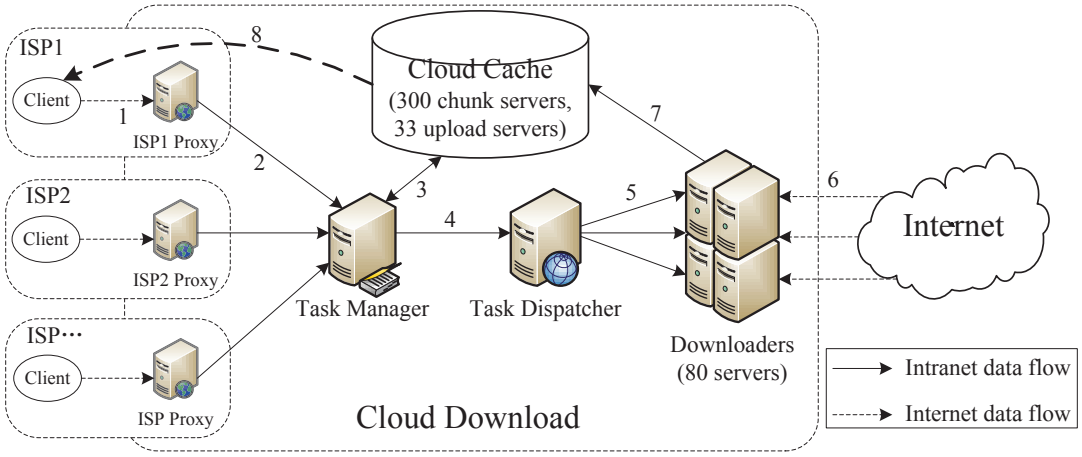


图 4.2　云下载系统架构

表 4.1　云下载系统硬件配置

硬件模块	服务器数	内存	存储	带宽
ISP代理	4	8 GB	250 GB	1 Gbps (内联网), 0.3 Gbps (互联网)
任务管理器	4	8 GB	250 GB	1 Gbps (内联网)
任务分发器	3	8 GB	460 GB	1 Gbps (内联网)
下载机集群	80	8 GB	460 GB	1 Gbps (内联网) 0.325 Gbps (互联网)
云缓存	335	8 GB	4 TB (数据块服务器) 250 GB (上传服务器)	1 Gbps (内联网) 0.6 Gbps (互联网)

系统。下面我们通过跟踪一次典型的云下载任务的消息和数据流来描述系统的组织和工作过程。

云下载用户首先必须安装客户端软件（下载链接： xf.qq.com），它能识别出用户位于哪个ISP，这样用户发出的视频（下载）请求就可以送达对应的ISP代理（图4.2箭头1）。每个ISP代理维护一个*请求队列*来控制用户发送给任务管理器的任务数（箭头2），以防止来自某个ISP的任务不正常剧增（比如遭受黑客攻击）。当时（*ACM-MM'11*论文发表时）系统维护了4个ISP代理，分别对应中国最大的4个ISP运营商：电信、联通、移动和教育网。

每接收到一条视频请求，任务管理器首先检查所请求的视频是否已包含在云缓存中（箭头3）。如果用户请求的视频链接是一个P2P链接，任务管理

器就检查云缓存中是否存在一个视频、它具有和P2P链接中包含的散列码相同的散列码，因为P2P（BitTorrent/eMule/Magnet）链接中会包含其文件散列码，但HTTP/FTP/RTSP链接不包含。如果不是P2P链接，任务管理器直接检查云缓存是否保存有相同的视频链接。如果所请求视频确实已存在于云缓存中，用户就可以立刻从云缓存中取回他所要的视频（箭头8）；否则，任务管理器就将视频请求转发给任务分发器（箭头4）。

每当从任务管理器那里收到一个视频下载请求，任务分发器就分配此任务给一台下载机（箭头5）。比如说，如果下载任务中的文件链接是一个P2P链接，那么对应的下载机就会像一个普通P2P 用户那样参与到对应的P2P用户集群中去。任务分发器主要负责平衡80台下载机的下载带宽压力，每台下载机并发执行多个下载任务（箭头6），而任务分发器总是分配新任务给当前下载带宽压力最小的那台下载机。

下载机一旦完成一个下载任务，就计算所下载视频的散列码，并试图把视频存入云缓存（箭头7）。但存入之前必须（用计算出的散列码）检查所下载视频是否已经包含在云缓存中。如果是，下载机直接删除该视频；否则，下载机需要检查下云缓存是否有足够的空间足以存放新视频。如果空间不足，云缓存需要删除部分视频来腾空间。具体的缓存容量规划和缓存替换策略在下面一章会详细研究。

最终，用户所要的视频就在云缓存中了，用户可以随时、随地来取回（箭头8）。由于用户的ISP信息可以在其提交的请求中得知，所以云缓存可以利用系统具备的**“ISP加速上传技术”**将数据高速传送给用户。

4.1.4 云下载系统设计

（a）数据传输加速

云下载必须尽一切可能保证用户能够高速取回数据，否则，用户可能还不如直接从互联网下载。考虑到视频传输的主要瓶颈就是跨越ISP的人为障碍 [99, 100]，我们通过**“ISP加速上传技术”**来达到目的，具体来说，就是尽量保证上传服务器和用户位于同一个ISP内。具体来说，如图4.3所示，云缓存由300台数据块服务器、33台上传服务器和2台索引服务器构成，这些服务器通过一个3层树状结构的数据中心网络连接。

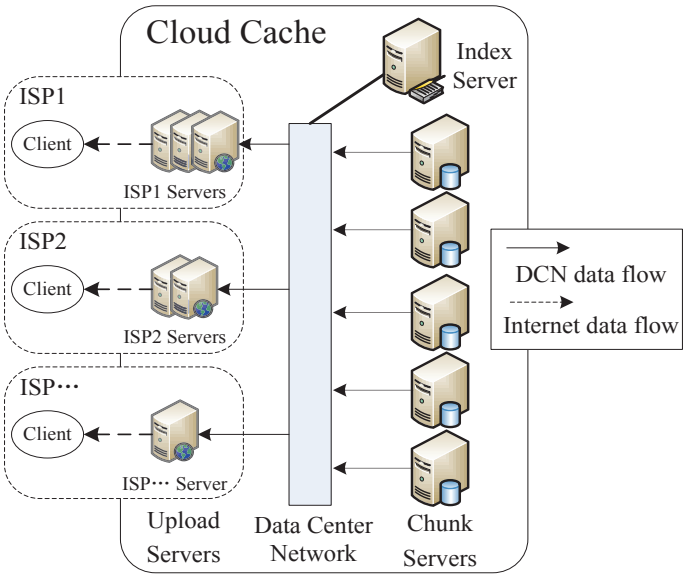


图 4.3　云缓存体系结构

每个文件都被切分成等大的数据块存在放数据块服务器中，且每个数据块都有一个冗余块以防止意外丢失，从而300台数据块服务器总共可以容纳$\frac{300 \times 4\,\mathrm{TB}}{2} = 600\,\mathrm{TB}$的独特数据。索引服务器维护数据块的元数据，以便于数据块查找和验证，元数据一个n维属性列表，其形如：$<$ *文件散列码*, *文件链接*, *数据块数目*, *第一个数据块的物理位置*, *第一个冗余块的物理位置* $>$。假设有一个用户A位于ISP1，他想要获取数据块服务器S中的视频f，而云缓存在ISP1中放置了10台上传服务器：U_1、U_2、$\cdots$、U_{10}，那么f的数据块首先从S随机传送到U_1、U_2、$\cdots$、U_{10}中的一台，比方说U_4，然后再从U_4传到用户A。

（b）下载成功率提升模型

在文件可以成功下载的情况下，云下载保证用户所请求视频的数据健康度不低于1.0。然而，云下载不能保证*一定可以成功下载某个文件*（实际上任何方案都不能保证）。本节我们构建一个简单的模型来分析云下载和普通下载的下载成功率。所谓普通下载，指的是用户通过普通方式（如Web下载、P2P客户端下载）从互联网上下载文件。考虑到互联网环境的高动态性和复杂性，我们所建立模型的目的是展示下载成功率的长期趋势，而不是短期的准确预测。

不管用户使用云下载还是普通下载，HTTP/FTP文件的下载成功率仅取决于文件链接本身的可访问性。根据我们对于大量HTTP/FTP文件链接的可访问性测

量结果，HTTP/FTP 文件的平均下载成功率为：

$$R_1 = R_1' = 0.414\text{”}$$

而P2P文件的下载成功率则要复杂得多，因为每一个用户的可访问性都是一个随机变量。考虑一个由N个用户分享文件f而构成的用户集群，并假设时间限制为T小时：每个用户都必须在T 小时内作出结论——下载成功或下载失败，因为我们不能无限期地等待下去。在时间限制T小时内，假设每个用户的平均在线时间为t小时、平均数据健康度为h（$h \leq 1.0$），那么对于一个普通用户P来说，在他上线的这t小时内，P可以期望碰到$\frac{n \cdot t}{T}$个其他用户（n是用户集群中的平均用户数）。假设用户之间的上线行为是互相独立的，那么用户P下载文件f的成功率期望值为：

$$R_2 = 1 - (1-h)^{\frac{n \cdot t}{T}}\text{”}$$

云下载使用一台稳定的下载服务器P'加入P2P用户集群，因此P'的在线时间t为$t = T$，从而P'的下载成功率期望为：

$$R_2' = 1 - (1-h)^n.$$

对于VideoCloud系统收到的所有用户请求，令α代表HTTP/FTP请求的比例，令β代表P2P请求的比例，那么普通下载和云下载的下载成功率分别为：

$$R = \alpha \cdot R_1 + \beta \cdot R_2, \text{ and } R' = \alpha \cdot R_1' + \beta \cdot R_2'.$$

我们对于VideoCloud系统17天的测量数据表明：h = 0.4，n = 5.4，$\frac{t}{T}$ = 0.12，α = 27.4%，β = 72.6%。从而

$$R = 31.8\%, \text{ and } R' = 79.3\%.$$

从模型中分析出的上述期望结果（R和R）被描绘在图4.4中，以同真实测量结果进行比较，可见真实系统的日均下载成功率为81.4%。

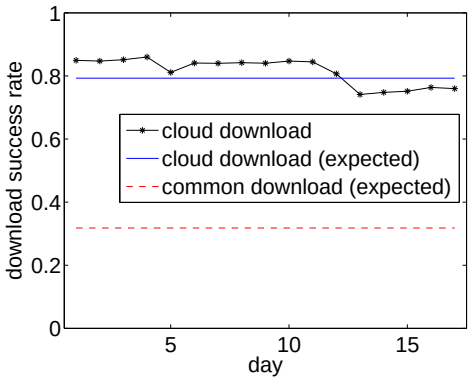


图 4.4　逐天下载成功率

（c）云缓存容量规划

VideoCloud系统拥有超过600万注册用户，不可能给每个用户分配无限制的缓存空间，代替地，每个用户只分配5 GB的免费缓存空间，因为5 GB基本上可以容纳一部完整的电视剧。从而超过600 万的注册用户在*最差情况*下需要超过29000 TB的缓存空间，这里所谓*最差情况*包含3个方面的意思：（1）每个用户都用满了分配给他的5 GB空间；（2）用户间不存在数据复用；（3）用户缓存数据被恒久保存、除非他已经取回。前两个方面很难预测，因为他们取决于用户心理和行为，所以我们唯一的选择是从第3个方面着手——我们把它替换成一个更实用、更经济的原则：用户缓存数据*只保存7天*，除非用户已经取回。7天的设置来自实践的观察：超过95%的用户在云端下载好视频后一周内会把它取回。当前VideoCloud系统日均接收到约22万用户请求，所请求视频的平均大小为379 MB，所以*最差情况*下总的云缓存容量应该为：

$$C = 379\ \text{MB} \times 0.22M \times 7 = 584\ \text{TB}.$$

为处理用户请求规模的波动，最终总的云缓存容量规划为$C' = 600$ TB，略高于上面计算的$C = 584$ TB。依靠这个600 TB的云缓存，用户间的数据复用率（即缓存命中率）达到了87%，说明大多数用户请求可以立刻满足，俗称为“云下载秒杀”。

（d）缓存替换策略

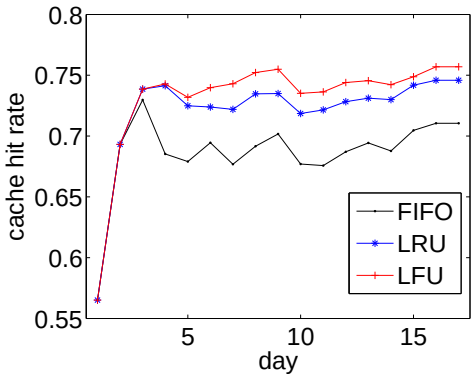


图 4.5 云缓存命中率

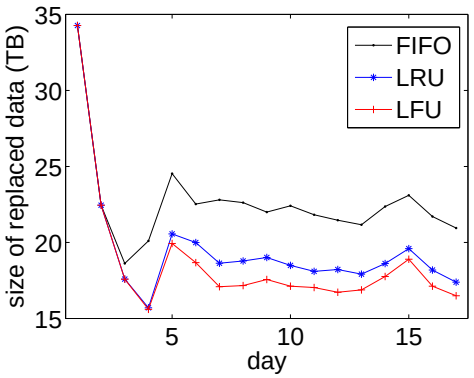


图 4.6 缓存替换数据量

我们还需要研究缓存替换策略以备不时之需。这里我们使用17天的系统日志记录来研究3个最常用的缓存替换算法：FIFO (first in first out)、LRU (least recently used)和LFU (least frequently used)，模拟缓存容量为70 TB（使用别的模拟缓存容量得到的结果类似）。主要考察两个指标：缓存命中率，和替换数据大小，前者希望越大越好，后者希望越小越好。从图4.5和图4.6我们发现：FIFO的命中率最低、LFU的命中率最高，所以VideoCloud系统采用LFU算法。实际上，已有文献 [101]说明LFU特别适合于数据对象的流行度较为稳定的系统，而VideoCloud正是这样的系统——其中大多数视频都是冷门视频、流行度十分稳定。

4.1.5 系统性能评估

（a）实验数据集

我们使用VideoCloud系统17天（2011年1月1日到17日）的完整日志数据来评估云下载的实际性能。日志数据包含大约387万条用户视频请求，涉及到100万左右的视频。大多数视频是.rmvb格式（约占40%）和.avi格式（约占20%），27.4%的请求视频是HTTP/FTP文件，剩下来的则是BitTorrent/eMule文件。系统日均统计数据见图4.7。对于每一条视频请求，我们都记录其下列信息：*用户标识*、*文件链接*、*文件散列码*、*文件类型*、*文件大小*、*视频请求时间*、云端下载机的*下载持续时间*、用户的*取回持续时间*、*缓存命中与否*、*下载成功与否*、等等。

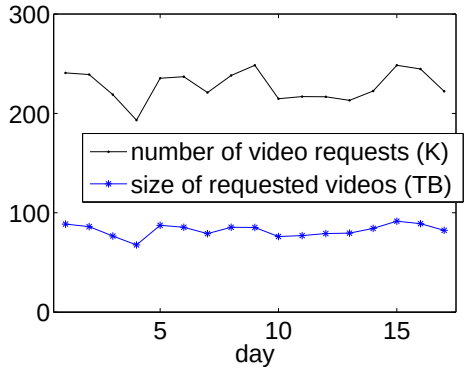


图 4.7　系统日均统计

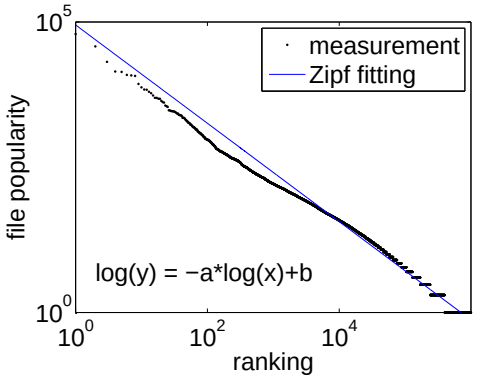


图 4.8　被请求视频的流行度分布

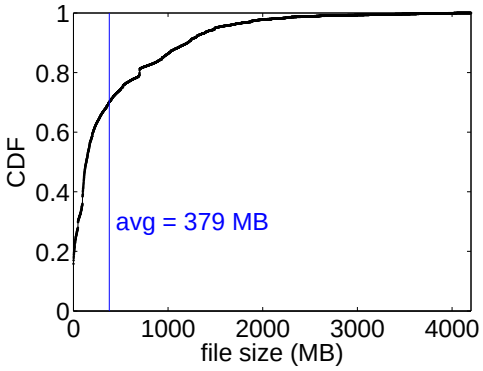


图 4.9　文件大小的CDF分布图

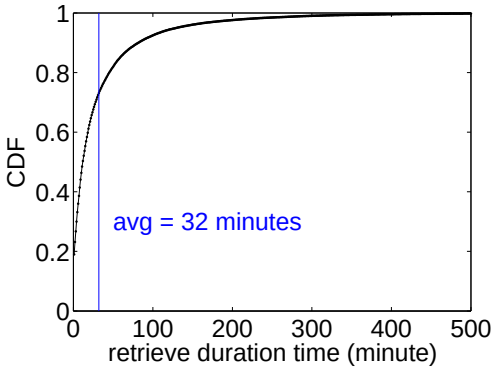


图 4.10　文件获取时间的CDF分布图

图4.8说明用户请求视频的流行度近似符合Zipf模型 [102]：令x表示一个视频文件的流行度排序，令y表示文件的流行度，那么就有下面的拟合公式

$$log(y) = -a \cdot log(x) + b,$$

它等价于

$$y = 10^b \cdot x^{-a},$$

其中$a = 0.8464$，$b = 11.3966$。更细节地看，97.1%的视频日均请求次数不到1次，只有0.09%的视频日均请求次数超过10次——这正是我们使用“日均请求超过10次”作为冷门视频和热门视频之间界限的根据。

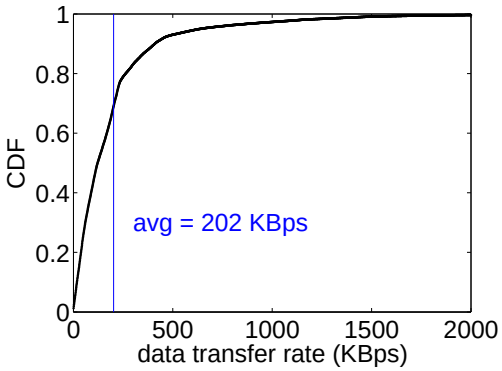


图 4.11　数据传输率的CDF分布图

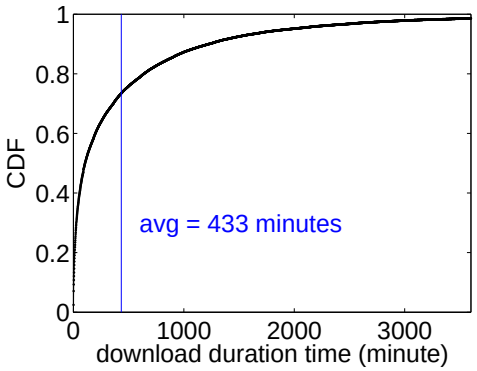


图 4.12　数据下载时间的CDF分布图

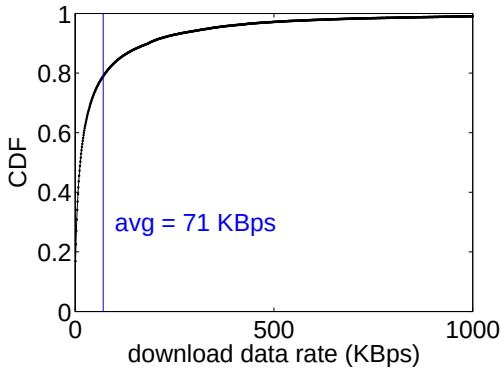


图 4.13　云端下载数据率的CDF分布图

（b）性能指标

(1) *数据传输率（Data transfer rate）*表示用户从云端取回视频时的数据率，具体来说data transfer rate $= \frac{\text{file size}}{\text{retrieve duration time}}$。

(2) *播放启动时延（View startup delay）*表示用户需要等待多久才能从云端开始获取他请求的视频。

(3) *能效（Energy efficiency）*表示用户使用云下载相比于普通下载的节能效率，这部分内容参见http://net.pku.edu.cn/ lzh/publications.html.

（c）数据传输率

从图4.9可以看出：用户请求文件的平均大小为379 MB，接近于一个普通质量100分钟视频的大小。奇怪的是有16%的文件小于8 MB，而且基本上是视频

简介、相关图像和文本，因为许多视频内容提供商喜欢在视频上“捆绑”一些简介、快照和广告。

从图4.10我们看到：用户平均取回时间为32分钟，73%的文件取回时间不到32分钟，93%的文件不到100分钟，因此，如果文件是一个普通质量100分钟视频，那么它可以顺畅地边下边播。如图4.11所示，平均数据传输率为202 KBps (> 1.6 Mbps)，超过80%的数据传输率超过37.5 KBps (= 300 Kbps)，这正是目前在线视频的普遍码率。为什么会有20%的数据传输率低于300 Kbps？原因主要是相关用户不在系统目前所支持的国内4个最大的ISP中，所以我们的ISP加速上传技术对他无效。

我们发现平均文件大小、平均数据传输率和平均取回时间这三者存在一个简单而精确的关系：

$$\frac{\text{平均文件大小(379 MB)}}{\text{平均数据传输率(202 KBps)}} = \text{平均取回时间(32分钟)}. \tag{4.1}$$

它给我们进一步优化性能提供了非常好的启发，有助于找到一个好的折中点。

（d）播放启动时延

从图4.12和图4.13可以看出，平均播放启动时延为433分钟，而云端平均下载数据率为71 KBps (= 0.57 Mbps)。联想到上一节我们发现的公式(4.1)，我们发现它对于播放启动时延和云端下载数据率完全不成立：

$$\begin{aligned}\frac{\text{平均文件大小(379 MB)}}{\text{云端平均下载数据率(71 KBps)}} &= 91 \text{ minutes} \\ &\ll \text{平均播放启动时延(433分钟)}.\end{aligned} \tag{4.2}$$

原因在于大多数的下载过程都比较漫长而低俗，比如说22%的下载数据率低于2 KBps，45%低于10 KBps，70%低于37.5 KBps (= 300 Kbps)。

4.1.6 小结和进一步工作

视频内容分发正成为互联网的“杀手级应用”，因为用户对视频质量和数量的需求不断膨胀，从而研究如何实现高质量的视频内容分发十分迫切。鉴于互联网存在数量巨大的冷门视频，却并未发现有专门针对冷门视频内容分发的

成熟系统方案，本文提出并实现了云下载方案，利用新兴的云计算/云存储平台为冷门视频提供高质量的内容分发服务。真实系统的运行日志确证了云下载方案的有效性。

进一步的工作至少包含3个方面。首先，可以扩展云下载的应用到更小的、私有的组织单位，比如一所大学、一个公司。我们观察到，许多公司的员工下班后还会继续开着电脑下载冷门内容，这是很不节能的，如果公司为员工搭建一个小规模的私有云下载系统，员工就可以在下班后集中提交下载任务，不仅节能、而且方便。

其次，虽然我们的云下载系统已经将下载成功率提升到81.4%，下载失败的情况任然无法忽视。事实上，精确判断下载失败是极为困难的，即使是最强大的搜索引擎也只能发现互联网上不到1% 的内容 [103]，所以不可能准确告知用户他请求的文件“最终”能不能下载成功。所以，我们需要在“合适的”时间告诉用户他的请求无法完成，但什么是“合适的”时间呢？目前我们采用了一个非常简单的准则：周期性检查下载进度，如果24小时内进度不变，就告诉用户他的请求失败、不要再等。下一步我们计划进一步分析系统日志数据，以发现更多的与下载失败相关的因素，从而设计一套更准确、更全面的判断准则。

最后，作为一个还处在起步阶段的新系统，VideoCloud云下载系统在构建每个模块时都倾向于采用直接和成熟的设计方案，而不是100%追求性能。所以本文的各项设计只代表我们最朴素和原始的方案，还存在很大的优化空间。

4.2 适应移动视频分发的云转码系统

4.2.1 背景、动机及工作简介

最近几年，使用iPhone、iPad、Android设备的移动用户越来越多，移动设备的销量已经远远超过传统的个人电脑销量。这虽然带来了使用的便捷，却同时带来了移动设备屏幕分辨率各不相同、电池储能较低、处理器计算能力有限等问题。现在用户使用移动设备观看视频已经非常普及，但互联网上的视频（比如美国的Youtube、Netflix、Hulu和中国的优酷、土豆、搜狐视频等）主要还是针对个人电脑、固定分辨率的，移动设备观看时被迫下载不适合其分辨率的视频且需进行转码运算，这对于移动设备的下载带宽、电池能量消耗、处理器计算能力都带来了沉重的（而并不必要的）负担。我们称上述问题为当前互联网视频和移动设备之间的格式和分辨率“鸿沟”。

解决上述“鸿沟”问题的传统做法，是让移动设备借助于Apple App或者Android App等软件在本地直接进行视频转码，但是由于视频转码是一项计算密集型的工作（视频转码的计算负载相当于同时播放多部视频）、可以轻易消耗掉移动设备本来就非常有限的电池能量和处理器计算能力，大多数用户并不会实际这么做，而是先在其个人电脑上转码、然后将转码好的视频传输到移动设备，然而这一“间接”方法又很不方便、特别是当用户出门在外的时候。

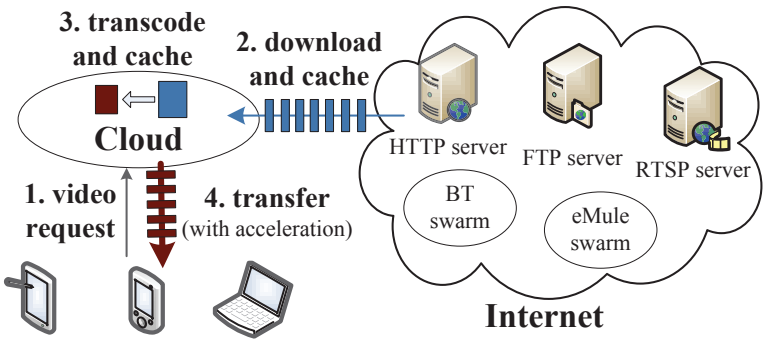


图 4.14　云转码方案工作原理

我们提出的**“云转码”**方案（如图4.14所示）使用一个中间的云系统“桥接”了互联网视频和移动设备之间的“鸿沟”，用户只需要向云系统汇报视频下载请求（一个HTTP/FTP/BitTorrent/eMule链接）和移动设备特性（所支持视

频格式和屏幕分辨率），云系统就会自动从互联网中下载（并缓存）相应视频，随后进行切合移动设备需要的转码操作（转码后的视频也被缓存），最后将转码过的视频高速传送给移动设备。在整个工作流程中，移动设备只在最后一步消耗电池能量——从云系统高速获取转码后的视频，因此“云转码”方案可以为移动设备播放视频提供良好的节能增值服务。

由于“云转码”方案将视频下载、转码的工作量都从移动设备转移到了云系统，必然导致云端较为沉重的下载带宽负担和转码计算负担。为了解决这一问题，我们使用了“隐式的”用户数据复用技术和“显式的”视频转码推荐和预测技术。对于一个视频v，云系统通常只需要在第一个用户请求时从互联网下载它并缓存到云中，之后的其他用户请求直接复用缓存的v即可。同时，v被云系统转码后的视频（注意v的转码视频可以有多个）也缓存到云中，依然可以复用。所有缓存和复用的操作都由云来执行，对用户是“隐式的”。

由于移动设备所支持的视频格式和分辨率千变万化，而云中缓存的转码视频数量有限，很可能缓存的转码视频不能精确匹配特定的用户请求。为了降低云端转码计算负担，当不能精确匹配时，云端会尝试向用户推荐一个比较“经典”的缓存转码视频作为近似匹配（当然用户完全可以不接受推荐）。另外，我们观察到云系统在夜间的计算负担要远远低于白天，出于负载平衡的考虑，我们设计了基于视频流行度的预测技术、提前转码一部分视频，以满足用户的潜在需求、将白天的计算负担部分转移到夜间。真实“云转码”系统（http://xf.qq.com）的运行日志显示，云端下载任务的缓存命中率达到87%，云端转码任务的缓存命中率达到66%，说明绝大部分的下载带宽负担和转码计算负担可以被有效避免。

4.2.2 云转码系统架构

真实“云转码”系统（如图4.15所示）总共使用了244台商品化服务器。目前它还处于开始阶段，每天大约收到8.6千用户请求，其中大约96%的用户请求是针对长视频的（超过100 MB），但目前的系统架构（尤其是云缓存部分）是为服务日均10万用户请求而设计。平均来说，一个移动设备在提交请求之后，需要大约33分钟来从云系统获取一个466 MB 的转码视频。典型地，上述过程会消耗iPhone手机9%的电量（约0.47瓦时）、iPad平板5%的电量（约1.25瓦时）。最后，用户从云端获取转码视频的平均数据传输率达到了1.9 Mbps，使得用户可以“边下边播”。

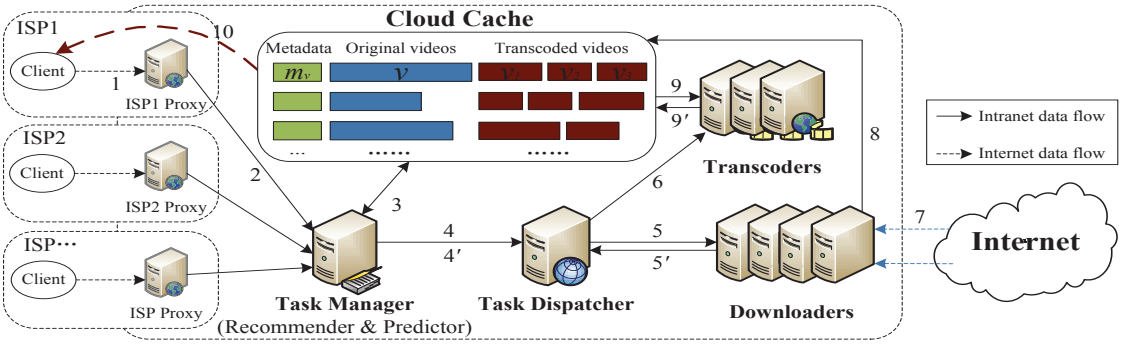


图 4.15　真实云转码系统架构和流程图

云转码系统架构由6个模块构成：（1）***ISP Proxies***（**ISP代理**）、（2）***Task Manager***（**任务管理器**）、（3）***Task Dispatcher***（**任务分发器**）、（4）***Downloaders***（**下载机集群**）、（5）***Transcoders***（**转码机集群**）和（6）***Cloud Cache***（**云缓存**），共计使用了244台商品化服务器，包括170台数据块服务器（构造了一个340-TB的云缓存）、20台下载服务器（使用6.5 Gbps的互联网带宽）、15台转码服务器（含有60个2.4 GHz频率的处理核心）以及23台上传服务器（使用6.9 Gbps的互联网带宽）。上述硬件配置（尤其是云缓存）是为服务10万日均用户请求而设计的，虽然当时（*NOSSDAV'12*论文发表时）的日均用户请求还不到1万。下面我们通过跟踪一次典型的云转码任务的消息和数据流来描述系统的组织和工作过程。

首先，用户将其视频转码请求发送给对应的ISP代理（图4.15箭头1）。每个ISP代理维护一个*任务队列*来控制用户发送给任务管理器的任务数（箭头2），以防止来自某个ISP的任务不正常剧增（比如遭受黑客攻击）。目前系统维护了10个ISP代理，分别对应中国最大的10个ISP运营商：电信、联通、移动、教育网等。每接收到一条任务请求，任务管理器就分两步检查所请求的视频是否已包含在云缓存中（箭头3）：

- *步骤1：检查视频链接。* 在云缓存中，每个原始视频v（所谓“原始视频”就是互联网上原先存在的、未经云转码系统转码的视频）都有一个独特的散列码，外加一系列指向v的视频链接、保存在对应的元数据m_v中。如果用户请求的视频链接是一个P2P链接，任务管理器就检查云缓存中是否存在一个视频、它具有和P2P链接中包含的散列码相同的散列码，因为P2P（BitTorrent/eMule/Magnet）链接中会包含其文件散列码，

但HTTP/FTP/RTSP链接不包含。如果不是P2P链接，任务管理器直接检查云缓存是否保存有相同的视频链接。如果最终没有在云缓存中找到用户请求的视频链接，任务管理器就发送一个*视频下载任务*给任务分发器（箭头4）。

- *步骤2：检查转码视频*。如果在云缓存中找到了用户请求的视频链接，任务管理器就进一步检查集合$T(v)$中是否包含一个存在的转码视频、刚好匹配用户提交的转码参数，$T(v)$指的是云缓存中保存的对应视频v的所有不同格式的转码视频的集合。如果刚好匹配，用户就可以立刻从云缓存中取回他所要的视频（箭头10）；否则，任务管理器会推荐一下$T(v)$给用户，如果用户愿意选择其中的某个转码视频，这就可以免除云端的转码计算压力——可见任务管理器同时也承担着“任务推荐器”的职责；但如果用户并不接受推荐，那么任务管理器就发送一个*视频转码任务*给任务分发器（箭头4'）。

每当从任务管理器那里收到一个视频下载任务时，任务分发器就分配此任务给一台下载机（箭头5）。比如说，如果下载任务中的视频链接是一个P2P链接，那么对应的下载机就会像一个普通P2P 用户那样参与到对应的P2P用户集群中去。如果从任务管理器那里收到一个视频转码任务，任务分发器就分配此任务给一台转码机（箭头6）。

任务分发器主要负责平衡20台下载机的下载带宽压力以及15台转码机的运算压力。每台下载机并发执行多个下载任务（箭头7），而任务分发器总是分配新任务给当前下载带宽压力最小的那台下载机。同样，新的转码任务也总是分配给当前转码计算压力最小的那台转码机。

下载机一旦完成一个下载任务，就计算所下载视频的散列码，并试图把视频存入云缓存（箭头8）。但存入之前必须（用计算出的散列码）检查所下载视频是否已经包含在云缓存中。如果是，下载机直接删除该视频；否则，下载机需要检查下云缓存是否有足够的空间足以存放新视频。如果空间不足，云缓存需要删除部分视频来腾空间。具体的缓存容量规划和缓存替换策略在**??**节中详细研究。当上述视频存储过程完成后，下载机就通知任务分发器（箭头5'）进行后续处理。

转码机接到转码任务后，首先从云缓存中读取对应的原始视频（箭头9），然后根据转码任务中所包含的用户指定参数进行转码，使用开源的FFmpeg编

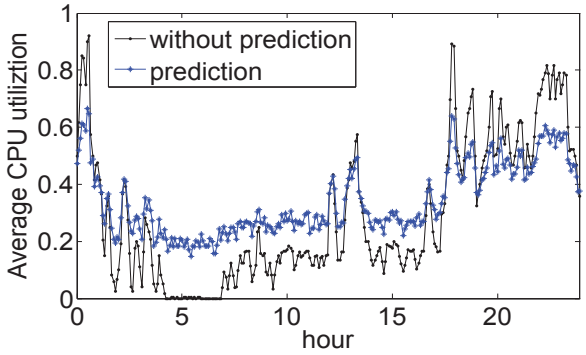


图 4.16　使用转码预测技术前、后的转码机平均CPU利用率

解码软件。一旦转码完成，转码机就检查云缓存中是否有足够空间可以保存新的转码视频（箭头9'）。

最终，用户所要的视频就在云缓存中了，用户可以随时来取回（箭头10）。由于用户的ISP信息当初在其提交请求时就已得知，所以云缓存可以利用系统具备的“ISP加速上传技术”将数据高速传送给用户。

4.2.3 云转码系统设计

（a）转码预测技术

一旦转码机集群的平均计算压力低于某个阈值（目前设置为CPU利用率50%），任务管理器就开始预测哪些视频可能需要提前转码。它首先使用最近24小时所收到的用户请求来更新视频流行度信息，包含两个方面：（1）每个视频的请求数，（2）最常见的转码参数。目前，排名前1000的视频和排名前3的转码参数被采纳，以构造“预测转码任务”。

每个预测转码任务被发送到任务分发器，这样系统的转码计算压力就从高峰期“填补”到了空闲期，如图4.16所示。此外，我们发现15台转码机的平均CPU利用率大概是34.13%，表示它们一天最多支持$\frac{8600}{34.13\%} \approx 25K$个转码任务，这里8600是一天收到的用户请求数。因此，为了支持日均10万的用户请求规模，我们至少还需要增加45台转码机。

（b）云缓存组织规划

如图4.17所示，云缓存由170台数据块服务器、23台上传服务器和3台索引服务器构成，这些服务器通过一个3层树状结构的数据中心网络连接。每个视

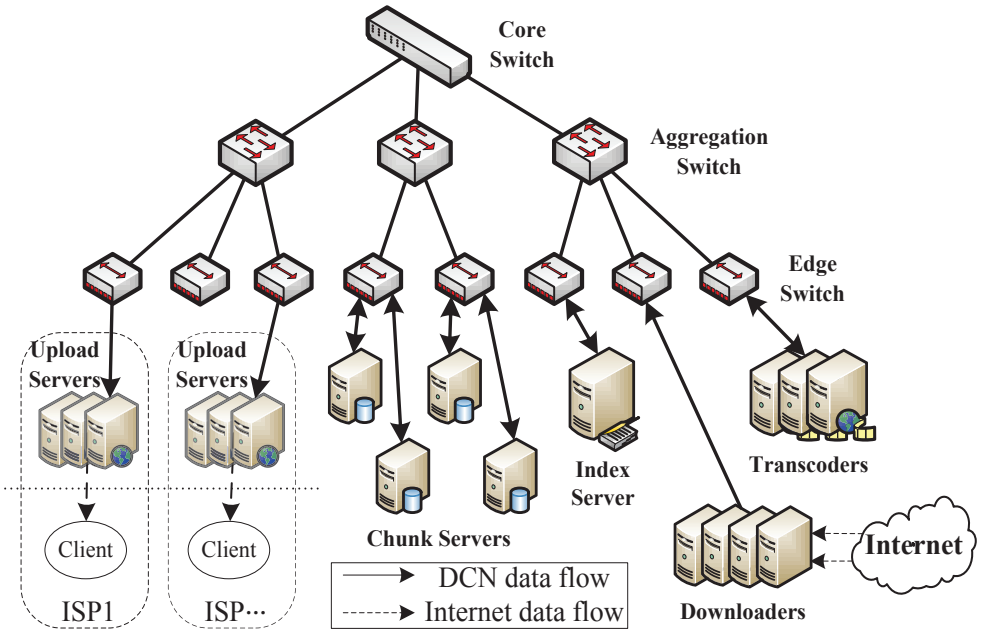
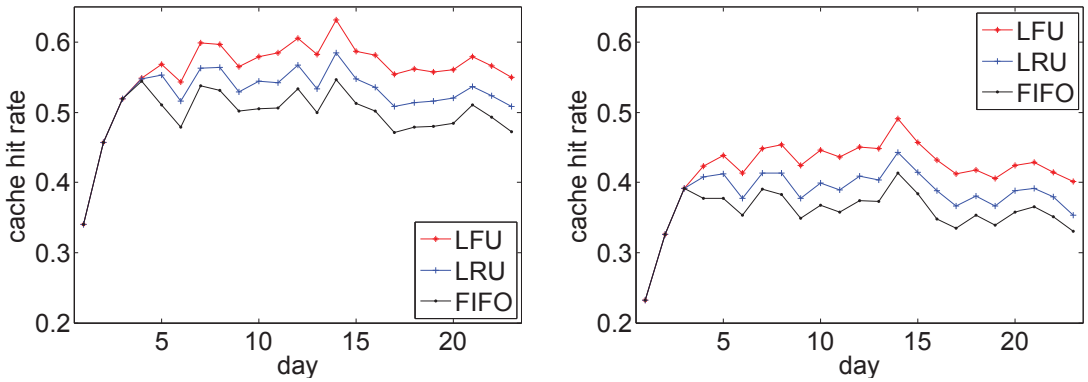


图 4.17　云缓存的硬件架构图

频都被切分成等大的数据块存在放数据块服务器中，且每个数据块都有一个冗余块以防止意外丢失，这样170台数据块服务器就可以容纳$C = \frac{170 \times 4\,\text{TB}}{2} = 340$ TB的独特数据。有一台索引服务器、外加两台备份索引服务器维护视频的元数据。

我们的云缓存系统是为服务日均10万用户请求而设计的。从图4.20中可以看出原始视频的平均大小为827 MB，而每个视频在云缓存中最多存放12天，所以如果不考虑用户间数据复用（也称缓存命中）、原始视频的总存储容量应该为$827\,\text{MB} \times 100K \times 12 = 969\,\text{TB}$；但系统日志显示，原始视频的缓存命中率大约为87%，所以原始视频的总存储容量规划为：$C_1 = 827\,\text{MB} \times 100K \times 12 \times (1 - 87\%) = 126\,\text{TB}$。每个原始视频平均关联3个转码视频，而从图4.21中可以看出转码视频的平均大小为466 MB，所以转码视频的存储容量规划为：$C_2 = 3 \times 466\,\text{MB} \times 100K \times 12 \times (1 - 87\%) = 213\,\text{TB}$。综上所述，总的云缓存存储容量应该是$C = C_1 + C_2 \approx 340\,\text{TB}$。

我们还需要研究缓存替换策略以备不时之需。这里我们使用23天的系统日志记录来研究3个最常用的缓存替换算法：FIFO (first in first out)、LRU (least recently used)和LFU (least frequently used)，模拟缓存容量为30 TB ($\approx \frac{8600}{100K} \cdot 340\,\text{TB}$。从图4.18我们发现：不管对于原始视频还是转码视



(a) 原始视频　　(b) 转码视频

图 4.18　缓存命中率

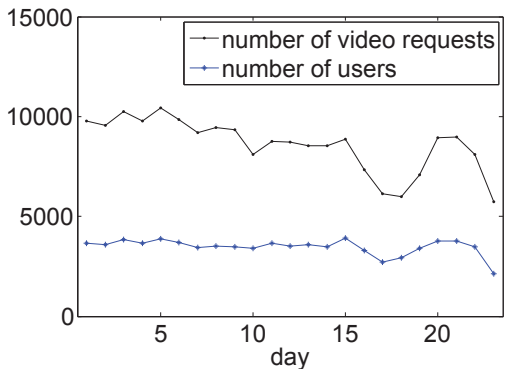


图 4.19　逐天统计数据

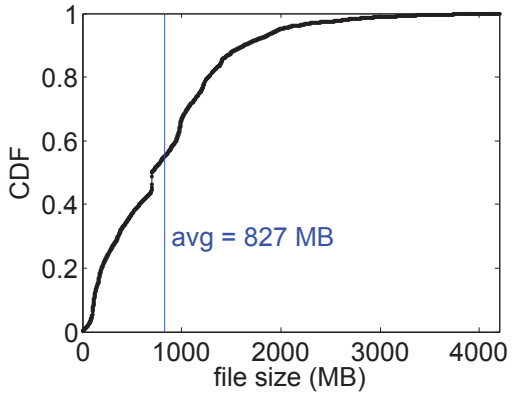


图 4.20　原始视频大小

频，FIFO的命中率都最低、LFU的命中率都最高。

（c）加速转码视频的传送

加速转码视频传送的目的在于节省（移动）用户取回视频的能耗。考虑到视频传输的主要瓶颈就是跨越ISP的人为障碍，我们通过“ISP加速上传技术”来达到目的，具体来说，就是尽量保证上传服务器和用户位于同一个ISP内。假设有一个用户A位于ISP1，他想要获取数据块服务器S中的视频v'，而云缓存在ISP1中放置了3台上传服务器：U_1、U_2和U_3，那么v'的数据块首先从S随机传送到U_1、U_2和U_3中的一台，比方说U_3，然后再从U_3传到用户A。

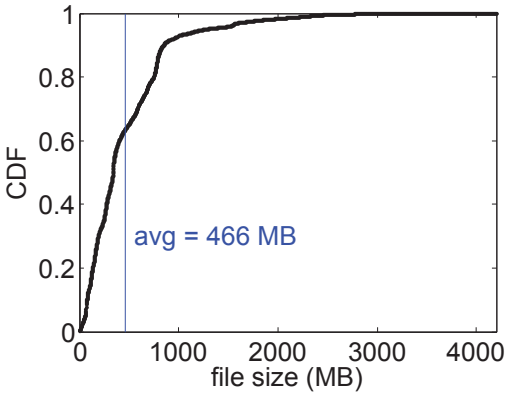


图 4.21　转码后视频大小

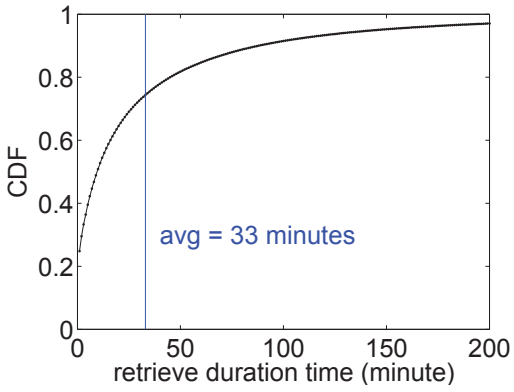


图 4.22　用户取回时间

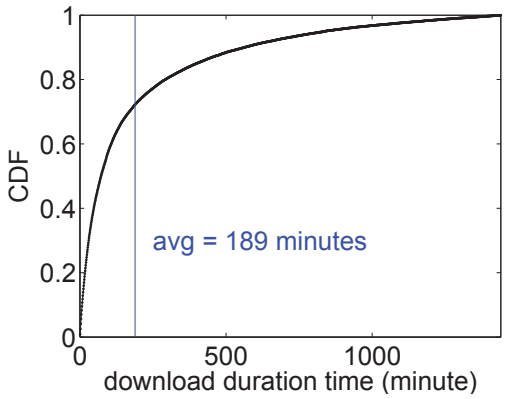


图 4.23　下载机的下载持续时间

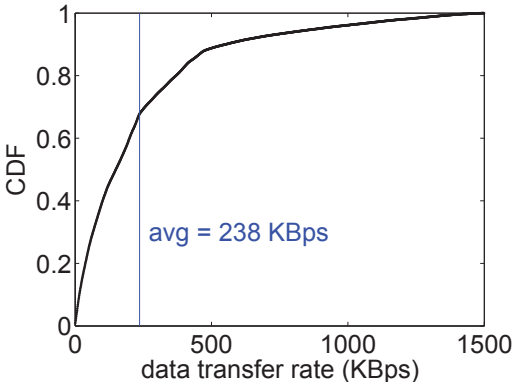


图 4.24　数据传输率

4.2.4 性能评估

我们使用云转码系统23天（2011年10月1日到23日）的完整日志记录来评估其性能，此日志包含197,400个视频转码任务的信息，共涉及76,293个视频。逐天统计数据见图4.19。对每个任务，系统都记录它的*用户设备类型*、*视频链接*、*转码参数*、*原始大小*、*转码后大小*、下载机的*下载持续时间*、*转码时间*、用户*取回时间*等等。

可以发现85%的视频链接是P2P链接，最流行的转码参数包括MP4-1024*768（10%，主要来自iPad用户）、MP4-640*480（38%，主要来自iPhone和Android手机用户）以及3GP-352*288（27%，主要来自Android手机用户）。如图4.20和图4.21所示，原始视频平均大小为827 MB，是转码视频平均大小的1.77倍。96%的原始视频是长视频（≥ 100 MB）。

如图4.22所示，用户平均需要约33分钟来取回转码后的视频，上述过程需要消耗iPhone4S大概9%（≈ 0.47 WH）的电力、iPad2大概5%（≈ 1.25 WH）的电力（如下表所示），这里iPhone4S/iPad2上的所有其它应用程序都关掉，屏幕亮度调整到25%。

数据传输率(≈KBps)	50	100	200	300
iPhone电力消耗(%)	8.7	8.9	9.0	9.2
iPad2电力消耗(%)	4.5	4.8	5.0	5.1

另一方面，图4.23描画了下载机的下载持续时间，平均为189分钟，是用户平均取回时间的5.72倍，因为下载机是从互联网上直接下载原始视频、没有ISP加速上传技术的支持。最后，图4.24反映了用户取回转码后视频的平均数据传输率达到了238 KBps (= 1.9 Mbps)，从而使得用户可以轻松地边下边播。

4.2.5 小结和进一步工作

作为一个还处在起步阶段的新系统，我们的云转码系统在构建每个模块时都倾向于采用直接和成熟的设计方案，而不是100%追求性能。所以本文的各项设计只代表我们最朴素和原始的方案，还存在很大的优化空间。

此外，我们观察到有一些Web浏览器也开始提供视频转码服务，比如中国最流行的移动Web浏览器“UC浏览器”已经使用其云计算平台将网上flash视频转码成3种解析度：144*176、176*208和240*320，以方便其移动用户。亚马逊公司最近报道其最新的Silk浏览器也具有云转码功能，特别是为其7英寸的Kindle Fire平板电脑服务。我们自己也已经开始探索将云转码功能整合进QQ浏览器。

第五章　用户构造云的内容分发

5.1 稳定性最优的端用户分组策略

5.1.1 背景、动机及工作简介

从经济的角度，可以将高动态性和高异构性的互联网用户组织起来、构造稳定可靠的“**虚拟云**”。为此我们提出将端用户分组、每个分组作为一个更为强大和稳定的虚拟云节点以承担更为重要的内容分发角色。这一方案典型的应用场景是P2P内容分发，它的本质优点在于能够容纳并利用互联网上巨大数量的边缘节点（如个人电脑、平板电脑、智能手机等），但由于边缘节点一般极不稳定，核心服务仍然主要依赖稳定的服务器节点。比如说，BitTorrent系统依靠服务器节点作为核心Trackers，KaZaa系统和eMule/eDonkey系统使用“超节点”（稳定强大的节点）来组织边缘节点和索引文件，而Skype也是通过从系统节点中挑选稳定节点作为其传输音频数据的“骨干”节点。有测量结果 [104]显示：在P2P流媒体系统中（如PPLive），大约80%的流量是由系统中不到10%的稳定节点负责传输的。因此，为了有效支持P2P系统的核心服务，我们面临一个困境：一方面，P2P系统中绝大部分节点都是不稳定的边缘节点，另一方面，P2P系统又需要足够数量的稳定节点来支持其核心服务。

虽然互联网中的端用户极不稳定，但他们数量众多、聚沙成塔的效应不可轻视——基于这样的思考，本文提出一种“稳定性最优”的P2P系统节点分组策略，其核心思想是：将P2P系统中大量的不稳定节点按照一定规则组合成“**稳定节点组**”，一个这样的节点组在功能上相当于一个稳定的服务器节点，从而构造出足够多的稳定节点来有效支持系统核心服务。图5.1是一个简单的例子，其中4个不稳定的节点构成了一个稳定的节点组。

分组策略的关键是要在系统稳定性和可扩展性之间做出一个好的平衡：一方面，将多个节点组合成更少数目的稳定节点组可以有效提高**系统稳定性**；另一方面，由于一个稳定节点组中的节点通常执行相同的功能，给系统提供的服务变少了，就影响了系统的**可扩展性**。因此，我们要求：将系统中不稳定节点组合成一定数量的稳定节点组，既能显著提高系统的稳定性，又能保证系统的可扩展性保持在一定的水平。

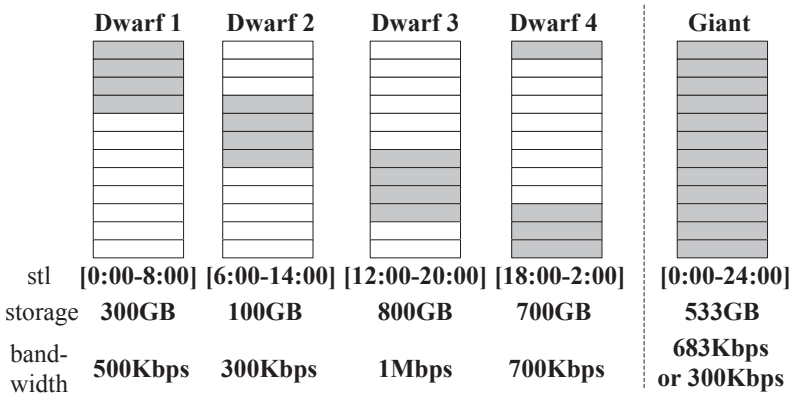


图 5.1　节点组服务能力示例，其中Dwarf表示普通节点，Gaint表示稳定节点组

鉴于上述要求，我们的方法是将P2P系统中近似同构（即稳定性相近）的节点分到相同的节点组中，组内、组间的节点连接关系不同，并且节点组的个数及节点组内的节点个数都仔细设计以在系统稳定性与系统可扩展性之间保持良好的权衡。更具体地，我们将上述分组问题形式化为一个“**同构的最大稳定性分组问题**”（Homogeneous Maximum Stability Grouping Problem、缩写为“**H-MSG**”问题），在统计模型下给出了该问题的最优理论解，并且结合实际情况设计了实现该理论解的具体策略，最后使用人工合成数据集、AmazingStore和CoolFish 两个真实系统数据集对该策略进行了模拟实验。

5.1.2 相关工作综述

上一节我们讲过研究者所面临的困境：一方面，P2P系统中绝大部分节点都是不稳定的边缘节点，另一方面，P2P系统又需要足够数量的稳定节点来支持其核心服务。试图解决这一困境的相关工作大致可分为三类：

(1) *GiantOnly*。除了在无结构P2P系统的广泛应用，*GiantOnly*策略还被用在一些基于DHT的系统中、比如OpenDHT [105]，其中只有十分稳定的节点才可以承担DHT节点的职责，不稳定节点不允许加入DHT网络，但可以作为享受服务的客户。

(2) *TotallyFlat*。虽然在覆盖网的组织上非常不同，但Gnutella [106]和Chord [107]这两个网络都为其成员构造了一个*Totally Flat*（完全水平）的世界——所有成员在功能上平等，不管它们稳定还是不稳定。

(3) *StableNeighbor*。由于稳定节点通常是匮乏的，某些研究者试图通过为每

一个节点攫取稳定的邻居节点来绕过这一问题。Godfrey等人 [108]研究如何从可用节点集中选择一个合适的子集，以作为相对稳定的邻居来替换失效的邻居节点。Yeung和Kowk [109]将邻居选择过程建模为一个协同博弈过程，这样用户就很可能形成稳定的“联合”。

5.1.3 分组模型

（a）符号和术语

考虑一个P2P系统S，它包含N个节点，每个节点在一个周期内都以一定概率在线（对实际系统周期$Period$一般取24小时）。我们要把这些节点分成m个不相交的节点组$G_1, G_2, \cdots, G_m$。对于一个节点组G_k，其周期在线时间（stl，即*session time length*）为τ_k，对应的随机变量为T，其组稳定性为$\psi_k = \frac{\tau_k}{Period}$，我们希望$\psi_k$越大越好（当然不会超过1.0）。从整个P2P系统来说，我们希望系统宏观的稳定性Ψ越大越好（下文将详述ψ和Ψ的具体定义）。

另一方面，我们还要考虑节点的“服务能力”：对于一个节点组G_k，其服务能力C_k定义为其组内在线节点的“加权”平均服务能力，这里“加权”是按照在线时间来计算的。仍然以图5.1为例，其中节点组的服务能力533 GB = 300 GB * 6/24 + 100 GB * 4/24 + 800 GB * 8/24 + 700 GB * 6/24。从整个P2P系统来说，我们要求系统宏观的服务能力C必须高于某个阈值（否则系统无力承担其必要的职责）。由此可见，P2P系统S的可扩展性取决于两个因素：分组数目m和所有分组的平均服务能力，定义如下（其中$\mathbb{E}()$表示数学期望值，C表示节点组服务能力）：

$$Scalability(S) = m \cdot \mathbb{E}(C) = m \cdot \overline{C} = \sum_{k=1}^{m} C_k. \tag{5.1}$$

很明显，当$m = N$时，$Scalability(S)$最大，这相当于节点不分组，或者说每个节点都是一个节点组。但$Scalability(S)$最大意味着系统S的稳定性最小。

P2P系统S的稳定性定义比可扩展性定义略为复杂，因为S的稳定性不可以简单地用每个节点组的稳定性的平均值或者和来衡量。举例来说，如图5.2所示，两种不同的分组方案S_1和S_2，节点组的稳定性均值接近（分别为0.615、0.6285）；但所形成的系统稳定性是不同的，很明显S_2优于S_1，因为S_2的节点组稳定性方差（0.0044）要远低于S_1的节点组稳定性方差

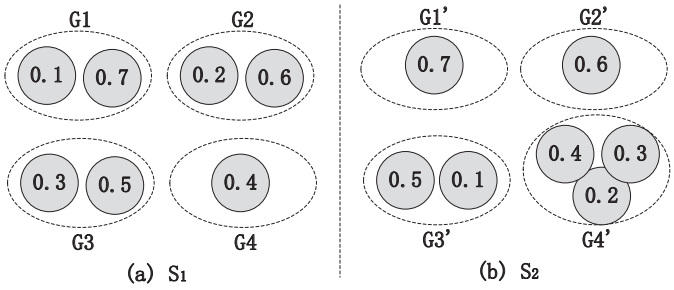


图 5.2　同一个系统的两种不同的分组方案S_1和S_2

（0.0216）。由此可见，节点组稳定性方差要比节点组稳定性均值更能表达系统稳定性，所以P2P系统S的稳定性定义如下：

$$Stability(S) = \frac{1}{\mathbb{V}ar(\Psi)} = \frac{m-1}{\sum_{k=1}^{m}(\psi_k - \overline{\Psi})^2}. \tag{5.2}$$

很明显，公式(5.1)和公式(5.2)构成了一对矛盾权衡：分组数m越多，系统可扩展性越高，但系统稳定性越低，我们的目标是保证系统可扩展性在一定水平之上、大幅优化系统稳定性。

（b）保证系统可扩展性

一个节点组G_k的服务能力C_k可以从多个方面来测度：带宽、CPU、内存、存储、查询效率等等。这里我们以查询效率为例来说明如何保证系统的可扩展性，因为查询效率经常被认为是一个P2P系统最重要的（网络）属性。具体来说，当系统中每个节点都发送一条查询请求后，分组策略下总的消息数目不能多于不分组的情形，这个要求可以形式化为公式(5.1)的特例：$Scalability(S) = \sum_{k=1}^{m} C_k = \sum_{k=1}^{m}(|G_k| \cdot \frac{1}{Avg\_search\_msg\#}) = N \cdot \frac{1}{Avg\_search\_msg\#}$。由于无结构网络和结构化（DHT）网络在运行机制上区别很大，下文将分别讨论二者的系统可扩展性保证。

无结构网络。 无结构网络的代表为Gnutella，考虑一个由N个节点构成的Gnutella网络S_1，其平均节点度为d，洪泛查询半径为TTL跳。如果我们将S_1分组成一个新的网络S_2，其中包含m个组，那么原来S_1中的大多数边将变成S_2中的“组间边”（inter-group edge），剩下的边则变成S_2中的“组内

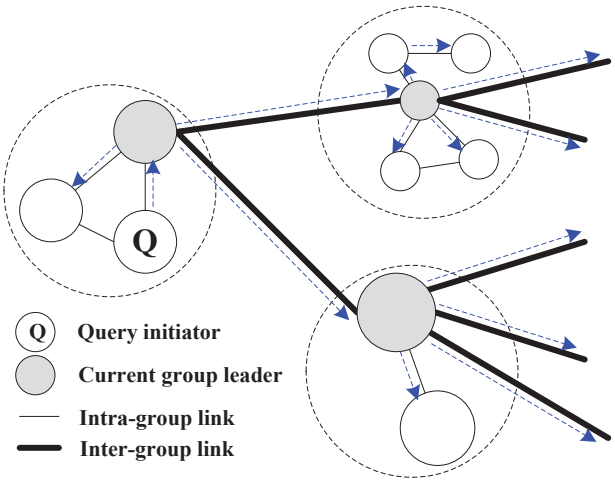


图 5.3　无结构P2P网络的分组示意图

边”（intra-group edge），从而导致两个问题：（1）平均组间边度d_G太大，导致S_2中的组被过度密集地连接；（2）组内边太少，导致很多组不连通。鉴于上述考虑，我们在保证S_2一直连通的前提下随机地修剪S_2中的组间边，直到d_G被减少到接近d，这样S_2的组间边密度就和普通的Gnutella网络差不多了。此外，对S_2的每个分组，我们随机加入组内边，直到此分组连通。最终，S_2的分组及其边连接情形将如图5.3所示。

为保证S_2的系统可扩展性，我们需要令$Scalability(S_1) \leq Scalability(S_2)$,

也就是说　　$N \cdot \frac{1}{Avg\_search\_msg\#1} \leq N \cdot \frac{1}{Avg\_search\_msg\#2}$,

$$Avg\_search\_msg\#2 \leq Avg\_search\_msg\#1. \tag{5.3}$$

假设TTL'是S_2的组间洪泛查询半径，那么在分组G_k内，查询消息数大概为$|G_k|$，因为组内洪泛通常可以到达所有成员。从而，公式(5.3)可变形为

$$d_G{}^{TTL'} \cdot \frac{N}{m} \leq d^{TTL}. \tag{5.4}$$

由于$d_G \approx d$，公式(5.4)近似为　$m \geq \frac{N}{d^{TTL-TTL'}}$,，或者　$\frac{N}{m} \leq d^{TTL-TTL'}$.　(5.5)

通常d位于3到5之间，$TTL \leq 7$，所以$TTL - TTL'$可能是1、2或3。

结构化（DHT）网络。 DHT的代表是Chord，与上文相似，一个Chord网络S_1也分组成一个新的网络S_2，包含m个组。由于DHT中同一个分组的成员共享同一个标识（*ID*），对分组G_k我们随机选择一个成员的*ID*作为整个分组的*ID*。组间边按Chord的方式组织。对S_2的每个分组，随机加入组内边直到此分组连通。公式(5.3)对DHT网络的分组依然成立，但形式有所变化：

$$O(\log m) + \frac{N}{m} \leq O(\log N). \tag{5.6}$$

公式(5.6)是一个数学超越式，因此我们只需要构建一个合适解$m \in O(\frac{N}{logN})$，由于

$$O(\log m) + \frac{N}{m} \in O(log\frac{N}{logN}) + O(logN)$$

$$\in O(logN) - O(\log\log N) + O(logN) \in O(logN).$$

事实上，$m \in O(\frac{N}{logN})$意味着平均分组数$\in O(logN)$。

（c）最大稳定性分组问题

在一段时间内加入系统的节点集合为$\mathbb{S} = \{n_1, n_2, \ldots, n_L\}$，假设每个节点$n_i$的加入时间$n_i.join$和离开时间$n_i.leave$已知，那么分组数$m$在上一小节已经确定，从而我们的目标可以形式化为如下问题：

定义 5.1 (最大稳定性分组问题（MSG）):

实例: 给定m和$\mathbb{S}$，已知每个节点n_i的加入时间$n_i.join$和离开时间$n_i.leave$。

解法: 将$\mathbb{S}$分成m个不相交的分组$G_1, G_2, \ldots, G_m$。

目标: 最小化$\mathbb{V}ar(\Psi) = \frac{1}{m-1}\sum_{k=1}^{m}(\psi_k - \overline{\Psi})^2$。

可以证明只要$m \geq 1$，MSG问题就是NP难的（不可解）。除了不可解，MSG还是一个“不合适”问题，因为它需要提前知晓每个节点的加入和离开时间，而这在真实系统中是很难做到的。因此，我们从另一个角度看待此问题，方法是限制每个分组中成员的同构性，试图将MSG简化为一个合适问题，即**“同构最大稳定性分组问题”（H-MSG）**：*在随机模型下，将稳定性相似的节点分到同一个组，不稳定的节点和稳定的节点不能在同一个组。*

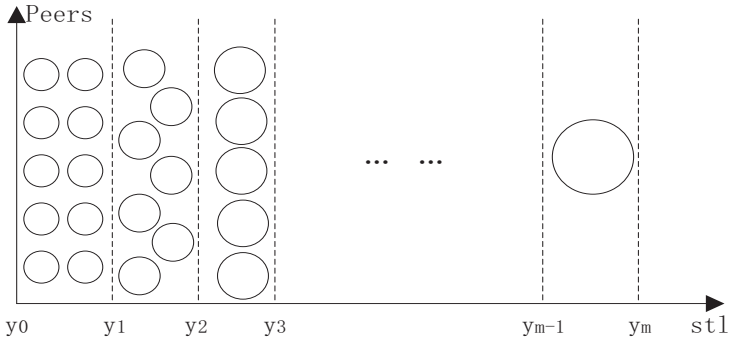


图 5.4　同构节点分组策略示意图

为解决H-MSG问题，如图 5.4所示，*stl*横轴被分成m个间隔：$[y_0, y_1)$, $[y_1, y_2), \ldots, [y_{m-1}, y_m)$，其中$y_0 = 0$，而$y_m = +\infty$，*stl*位于同一个区间的节点被分到同一个组。而我们的目标是最小化$\mathbb{V}ar(\Psi)$。

（d）H-MSG问题在统计模型下的最优解

统计模型。过去的文献中多次提及：一个网络中节点的到达$v.(.)$是一个无记忆的随机过程，且往往是经典的泊松过程 [110]。此外文献 [111]和 [112]都指出：节点*stl*的分布$D(.)$表现出一种可以预测的随机模式，且往往是类Zipf模式 [113, 114]。图5.5和图5.6描画了AmazingStore系统在线用户数和加入用户数随时间的变化，很明显两者的分布都具有周期模式。虽然图5.6的曲线很难用一行公式来简单地总结，但通过取样和差值等方法，很容易近似出$v.(.)$ 的分布模式。AmazingStore系统3个月内所有用户的会话时间模式被描画在图5.7中，这一长尾模式同经典的类Zipf分布相去甚远，但是同最近几年提出的“延展指数”分布 [115]却拟合的很好，这样我们又得到了$D(.)$的分布模式。

H-MSG问题的最优解。为便于分析，我们从系统运行过程中抽取一个足够大的时间槽st，那么节点组G_k在时间槽st的稳定性为

$$\psi_k = 1 - \mathbb{P}(\phi_k(st)), \tag{5.7}$$

其中$\phi_k(st)$表示G_k在第st个时间槽为空集（不包含任何节点）。

定理 5.1: ψ_k是y_{k-1}和y_k的函数。

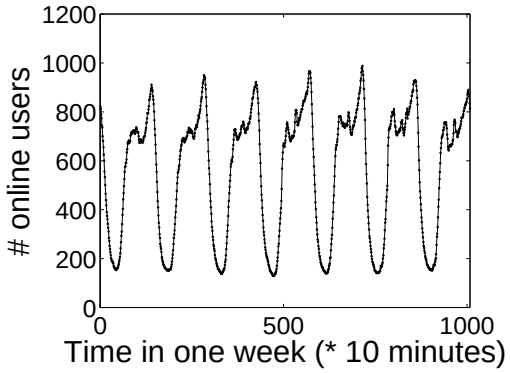


图 5.5 AmazingStore系统在线用户数

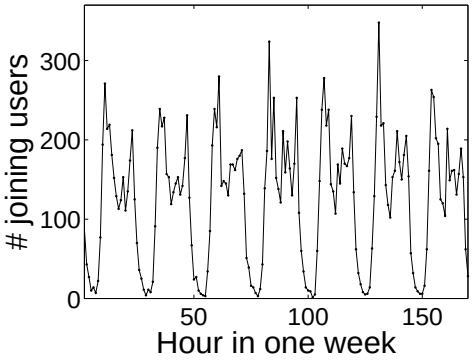


图 5.6 AmazingStore系统加入用户数

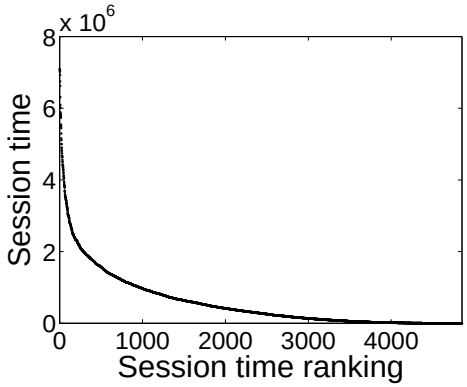


图 5.7 AmazingStore系统3个月内所有用户的会话时间模式

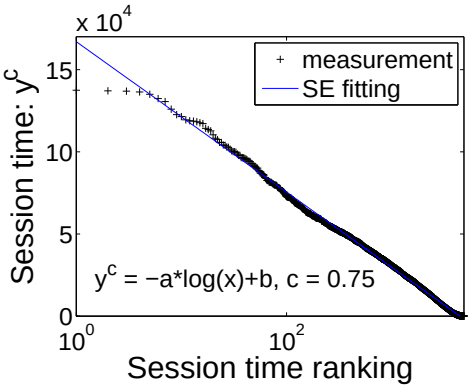


图 5.8 SE分布可以很好地拟合会话时间模式

定理5.1表明H-MSG问题确实是一个“合适的”优化问题，其证明见本文附件，可以在http://net.pku.edu.cn/ lzh/publications.html下载到。由公式(5.2)、公式(5.7)和定理5.1，我们可以获得推论5.2：

推论 5.2: H-MSG问题可以归结为一个合适的优化问题，其中$y_1, y_2, \ldots, y_{m-1}$是决策变量，目标是最小化$\mathbb{V}ar(\Psi)$。

这样一来，H-MSG问题也同时是可解的，因为它不再是一个NP难问题，所以我们又有了下面的推论：

推论 5.3: 在统计模型下，H-MSG问题既是合适的又是可解的。

5.1.4 性能评估

为了验证稳定性最优的端用户分组策略的有效性，我们使用了3个不同的数据集：（1）人工合成数据集；（2）来自一个真实的P2P文件分享系统“AmazingStore”的数据集；（3）来自一个真实的P2P流媒体系统“CoolFish”的数据集。这3个数据集互为补充，较为全面地考察了分组策略的性能：

- *人工合成数据集*。人工合成1000个节点，假设系统中节点的加入时刻满足泊松分布、节点的在线时间满足指数分布。
- *AmazingStore系统数据集*（网站链接http://www.amazingstore.org）。我们收集了该系统3个月的用户数据，共4854个节点。
- *CoolFish系统数据集*（网站链接http://www.cool-fish.org）。我们收集了该系统1个星期的用户数据，其中每一天的节点数都在300左右。

并且，我们还将所设计的分组策略同领域内相关工作（GiantOnly、Chord、Gnutella）进行了比较。在这3个数据集上的模拟实验都表明，我们设计的分组策略在系统稳定性与系统可扩展性之间取得了良好的权衡：系统稳定性得到大幅提升，同时系统可扩展性的损失是可以接受的。

（a）性能指标

(1) *Churn rate*（动态性）：动态性越高，节点组的稳定性越差。
(2) *Storage capacity*（存储容量）：以存储容量为代表刻画系统的可扩展性，存储容量越大，可扩展性越高。
(3) *Stable storage capacity*（稳定存储容量）：刻画了系统能够“稳定”提供的存储容量，它表达了系统在稳定性和可扩展性两方面的综合性能。
(4) *Maintenance overhead*（维护开销）：为实现分组策略必须支出的额外通信开销。

（b）人工合成数据集的实验结果

从图5.9可以看到，我们设计的节点分组策略（Grouping）的节点动态性比领域内相关工作GiantOnly略高、但远远低于Chord和Gnutella。从图5.10可以看到，分组策略的存储容量比Chord和Gnutella略低、但远远高于GiantOnly。综上所述，GiantOnly、Chord和Gnutella都是较为极端的系统，片面地追求了系统稳

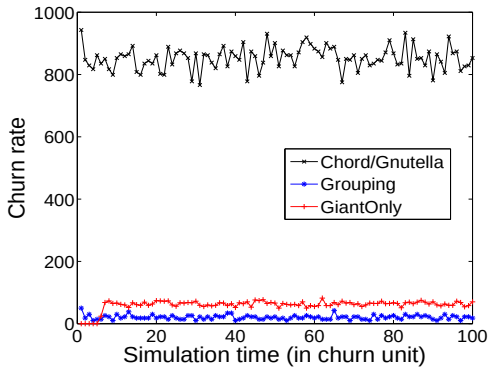


图 5.9　系统节点动态性

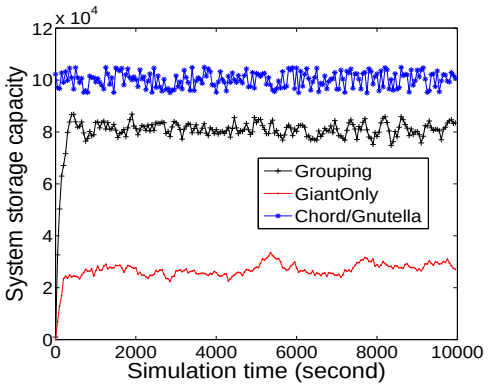


图 5.10　系统存储容量

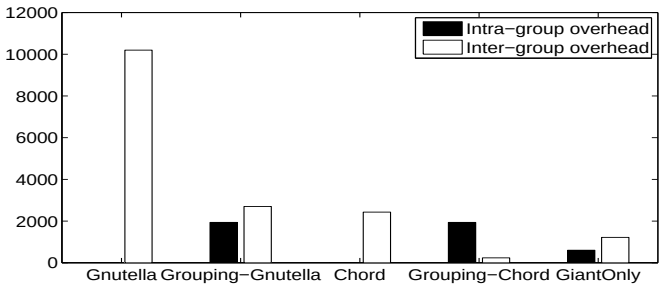


图 5.11　维护开销比较

定性或系统可扩展性；而我们设计的节点分组策略则在系统稳定性和系统可扩展性之间取得了良好的平衡。

图5.11记录了我们的分组策略和Gnutella、Chord以及GiantOnly的维护开销。可以看出，Gnutella和Chord只有组间开销，因为它们都采用水平网络结构，一个组就是一个节点，而其中尤以Gnutella的维护开销为最大、远远超过别的方案。Grouping-Gnutella的组间开销超过组内开销，但对于Grouping-Chord来说情况正好相反。GiantOnly的组内开销较小，因为每个组成员只发送其状态信息给组领导，而其组间维护消息是以*TTL*-洪泛方式进行的。

（c）AmazingStore数据集的实验结果

AmazingStore系统中4854个节点被分成396组，平均组稳定性为$\overline{\Psi} \approx 0.6$，这看起来不够高，但如果考虑到AmazingStore系统夜间8个小时（即23:00 – 7:00）几乎鲜有用户在线，0.6的稳定性就十分可观了。从图5.12、图5.13、

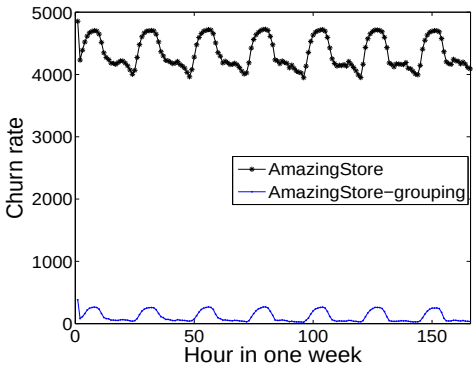


图 5.12　AmazingStore系统节点动态性

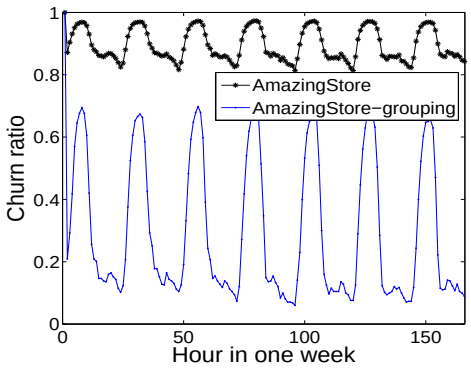


图 5.13　AmazingStore系统节点动态比例

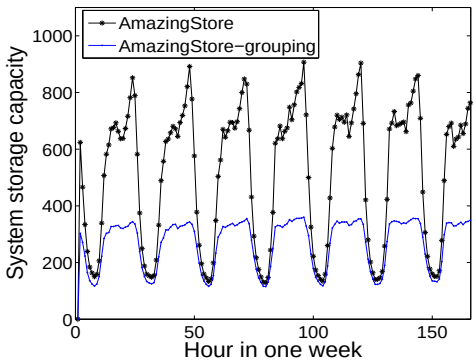


图 5.14　AmazingStore系统存储容量

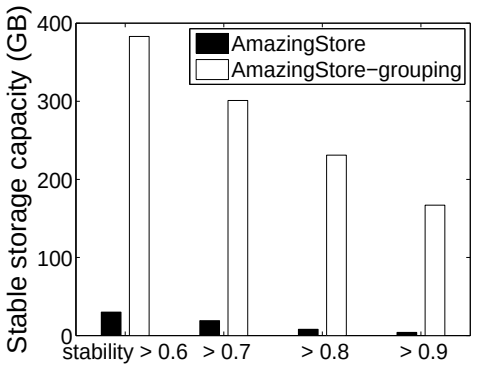


图 5.15　AmazingStore系统稳定存储容量

图5.14、图5.15可以看出，如果对AmazingStore这一P2P文件分享系统内的节点使用我们设计的分组策略，系统动态性将大幅下降（等价于系统稳定性大幅提升），同时系统稳定存储容量大幅上升。但是，系统存储容量将会下降，尤其是高峰时段，但这种损失在预料之中，是可以接受的。

（d）CoolFish数据集的实验结果

从图5.16、图5.17和图5.18可以看出，如果对CoolFish这一P2P流媒体系统内的节点使用我们设计的分组策略，系统动态性将大幅下降（等价于系统稳定性大幅提升），同时系统稳定带宽容量大幅上升。

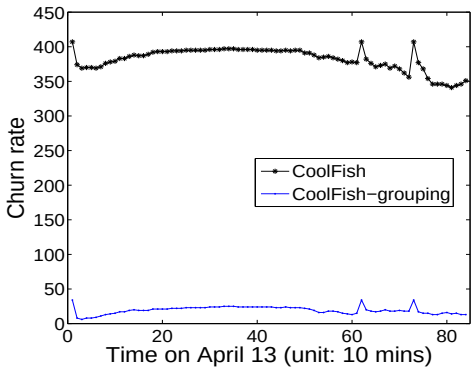


图 5.16 CoolFish系统节点动态性

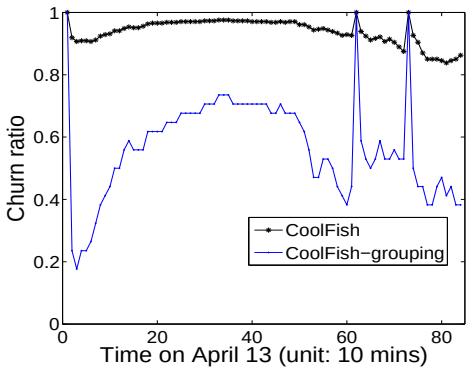


图 5.17 CoolFish系统节点动态比例

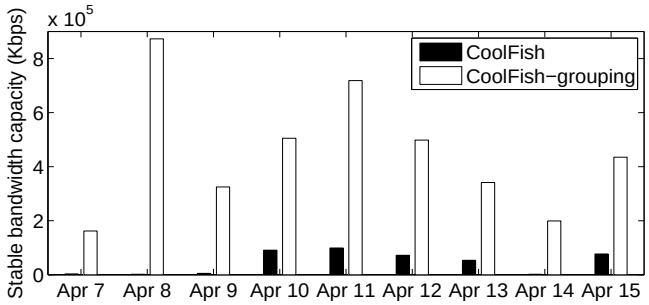


图 5.18 CoolFish系统稳定带宽

5.1.5 小结和进一步工作

用户构造云（或P2P系统）一直面临缺乏稳定节点的困境，受此驱动，本文研究了如何将大量的高动态性、高异构性的端用户节点分组“捆绑”成稳定的虚拟节点的策略。我们首先建立了端用户分组模型，然后提出一个同构化的分组策略，在保证系统可扩展性的前提下能够最大化系统稳定性。我们使用人工合成数据集、AmazingStore和CoolFish两个真实系统数据集对该策略进行了模拟实验，结果验证了我们提出的分组策略的有效性。

5.2 流媒体数据源的最快切换策略

5.2.1 背景、动机及工作简介

由用户构造的流媒体系统中可能存在一个或多个（流媒体）发布源，它们将源数据不断传播到系统中。对于一个多源系统来说，多个发布源可能以“串行”或“并行”的方式工作。比如说，在一个视频会议系统或远程教学系统中，每个成员结点都可以成为发布源，但通常任一时刻系统中仅有一个发布源，我们称这种方式为“串行”；而对于实时网络电视系统来说，对应于多个电视频道的多个发布源通常是同时存在、并行工作的，我们称这种方式为“并行”。本章考虑有多个发布源而它们之间串行工作的用户构造（也可称为P2P）流媒体系统，研究的关键问题是：**如何使得流媒体发布源之间能够快速切换，以尽量减少新发布源的启动时延。**

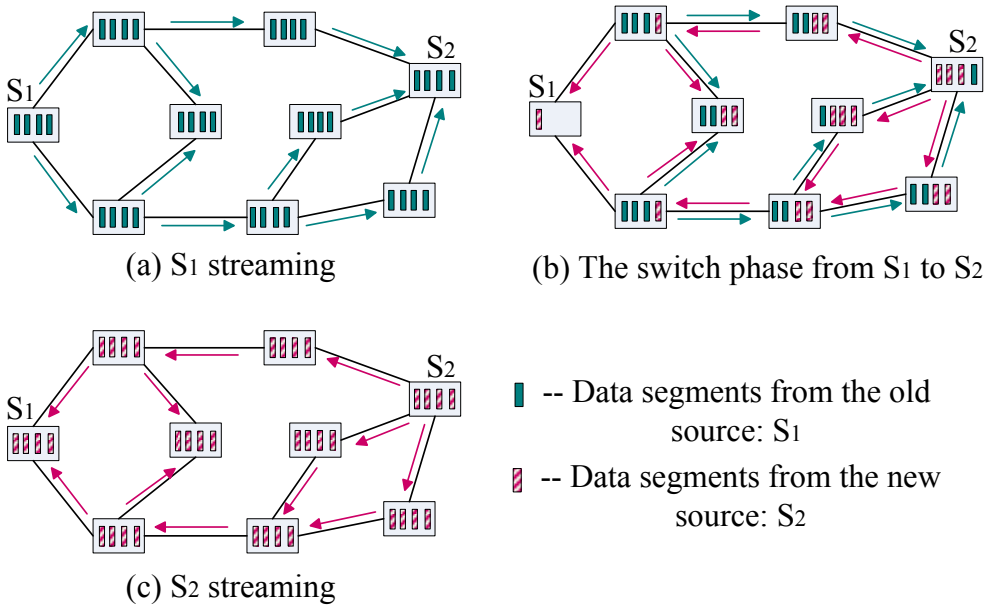


图 5.19　流媒体数据源切换过程示例

图5.19显示了一次简单的流媒体数据源切换过程中的数据分布与流向。这一过程可以划分为3个阶段：（a）起初，旧的发布源S_1向其邻居结点发布流媒体数据，这些数据被其它结点依次传播到整个系统中；（b）旧发布源S_1停止发布数据，新发布源S_2开始发布数据，系统中同时存在来自S_1和S_2的数据流；

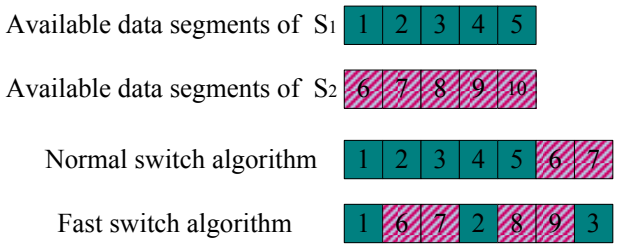


图 5.20　快速切换算法和传统切换算法的比较示例

（c）经过一段时间，系统中每个结点都完成了对来自S_1的流媒体的播放，只有来自S_2的数据在系统中不断传播。很明显，我们要研究的源切换问题可以归结为如何尽可能地缩短阶段（b）的持续时间。

更具体地说，我们需要设计一个合适的**源切换算法**来最小化每个结点的**源切换时间**（源切换时间等价于新发布源S_2的播放启动时延），而必须遵守的限制性前提是——某个结点可以开始对来自S_2的流媒体的播放当且仅当：1）该结点已经完成了对来自S_1的流媒体的播放，2）该结点已经接收到了足够多的来自S_2的数据（以保证S_2的“平滑启动”）。

我们首先给数据源切换过程建立了数学模型，从而将上述问题形式化为一个数学优化问题，然后推导出此数学优化问题的最优解。鉴于实际系统情况的复杂性与网络环境的动态性，提出一个称为**“流媒体数据源最快切换策略”**的实用贪心算法，通过交错旧数据源与新数据源的数据传递来趋近理论最优解。“传统源切换算法”并不交错旧源与新源的数据传递——它总是优先处理旧源数据的传递。图5.20中的示例比较了两者的不同：当前结点在每个数据调度周期中可以获取7个数据分片，而当前来自旧源的可用数据分片是1、2、3、4、5，来自新源的可用数据分片是6、7、8、9、10，传统切换算法优先处理的数据传递，因此它将获取1、2、3、4、5、6、7这7个数据分片，而**快速切换算法**根据它特殊的数据获取优先权计算原则交错处理、的数据传递，它将数据分片1、2、3与6、7、8、9的传递交错在一起。

我们在多个真实测量的覆盖网拓扑结构上做了大量模拟实验来证实快速源切换算法的有效性，覆盖网结点数目从100到10000不等。模拟实验的结果显示：我们提出的快速源切换算法相比**“传统源切换算法”**能节省20%-30%的切换时间，同时不会带来额外的通信开销，并且切换时间的减少比例随着系统规模的增大而趋于增加。最后，本文的贡献可大致总结为如下3个方面：

(1) 就我们目前所知，本文是P2P流媒体领域第一个对媒体发布源切换问题做出的深入研究。我们给源切换过程建立了数学模型，从而将源切换问题形式化为一个数学优化问题，并推导出此数学优化问题的最优解。
(2) 提出了一个称为“快速源切换算法”的实用贪心算法，它通过交错旧源与新源的数据传递来趋近理论上的最优解。
(3) 通过在多个真实测量的覆盖网拓扑结构上进行的大量模拟实验，证实了快速源切换算法的有效性。

5.2.2 相关工作综述

CoolStreaming使用基于网状协议的P2P网络多播技术构建了一个灵活、实用的P2P流媒体系统 [116]。它提供了对多个发布源的支持，但并没有描述其发布源切换机制。P2P流媒体系统AnySee [117]采用覆盖网间的优化方法（Inter-overlay Optimization）来减少数据源到接收端的时延，从而减少启动时延，但端到端时延与启动时延都不同于发布源切换时间。

Zhang等人观察到网状协议采用“拉”数据的方法虽然比“推”数据要节省带宽，却带来了更大的数据传播时延，因此他们设计了一个“推拉结合”的系统GridMedia [118]。他们将P2P流媒体系统中的数据分片分成两类：一类数据分片只在被请求获取时才传播，称为“拉数据”；另一类数据分片一旦结点收到就立即传播给邻居，称为“推数据”。GridMedia系统主要的设计目标是减少数据传播时延，它间接地减少了发布源切换时间。然而，推数据的方法必然带来相当大的通信开销，而且也不能从本质上保证源切换时间的减少比率。

Xu等人考虑了P2P流媒体系统中的数据率分配问题 [119]。P2P流媒体系统中每一个结点都连有若干带宽不等的邻居，Xu等人设计了一个名为OTS_{p2p}的数据率分配算法，能够保证当前结点具有最小的数据缓冲时延，从而减少启动时延。然而，OTS_{p2p}算法成立的前提非常苛刻，它要求系统结点的带宽满足严格的倍数关系，实际的P2P流媒体系统是无法满足该前提的。

5.2.3 切换过程建模

因为P2P流媒体系统的工作方式是纯分布式的，所以一个结点直到在其邻居结点中发现有来自新发布源的数据分片时才能得知发布源切换过程的开始，也就是说，运行于每个结点的源切换算法不会假设有任何源切换“先验”顺序的

表 5.1　源切换过程建模的相关参数

参数	说明
S_1	旧发布源。
S_2	新发布源。
Q	每当收集到来自S_1的Q个连续的数据分片时，就播放S_1流媒体。如果连续的数据分片数不够（$\leq Q$），播放将立刻暂停。
Q_1	来自S_1的数据分片还有Q_1个尚未获取。
Q_s	为启动S_2流媒体的播放所需获取的数据分片总数。
Q_2	为启动S_2流媒体的播放，还差Q_2个数据分片尚未获取。初始时，Q_2=Q_s。
p	每秒播放的数据分片数目。
I	当前结点的总输入带宽，以每秒输入数据分片数来衡量，是一个常数。
I_1	分配用来获取来自S_1的数据分片的输入带宽。I_1是动态调整的。
I_2	分配用来获取来自S_2的数据分片的输入带宽。I_2是动态调整的。
T_1	获取到来自S_1的所有数据分片的预期时间。
T_1'	完成S_1流媒体播放的预期时间。
T_2	获取到来自S_2的最初Q_s个数据分片的预期时间。

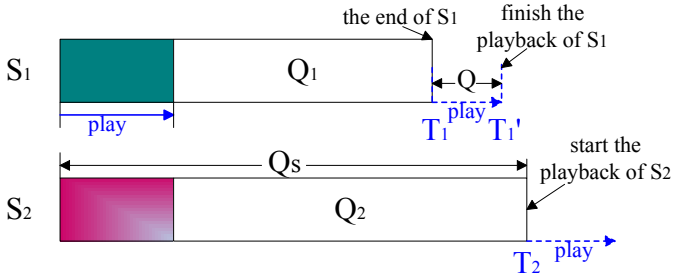


图 5.21　源切换过程的时序图

存在。当结点发现有新发布源的存在时，它就启动并执行源切换算法，然后在每个数据调度周期再次执行源切换算法，直到完成了旧发布源的所有数据的播放，也就是完成了源切换过程。

源切换过程建模所需的参数列举在表5.1中。我们使用图5.21来图形化表5.1中的参数以使抽象的参数更加具体。每当收集到来自S_1的Q个连续的数据分片时，就播放S_1流媒体，但是为启动S_2的播放，则需要获取Q_s个数据分片。在流行的实用P2P流媒体系统中，Q_s通常要比Q大很多，以保证新发布源的平滑启动。当前结点的总输入带宽I被被分配到I_1、I_2两个部分，以分别获取来自S_1、S_2的数据分片。I_1、I_2由源切换算法动态分配。

对照表5.1和图5.21，源切换问题可以被形式化成如下所示的数学优化问题：

优化目标：最小化T_2

$$\text{限制条件：}\begin{cases} I = I_1 + I_2; \\ T_1 = \frac{Q_1}{I_1}; \\ T_1' = T_1 + \frac{Q}{p}; \\ T_2 = \frac{Q_2}{I_2}; \\ T_2 \geq T_1'; \end{cases}$$

$$\text{上面的限制条件可以简化为：}\begin{cases} T_1' = \frac{Q_1}{I_1} + \frac{Q}{p}; \\ T_2 = \frac{Q_2}{I - I_1}; \\ T_2 \geq T_1'; \end{cases}$$

从而可以得到不等式：

$$\frac{Q_2}{I - I_1} \geq \frac{Q_1}{I_1} + \frac{Q}{p}; \tag{5.8}$$

上面的不等式可改写为：

$$I_1^2 + (\frac{p(Q_1 + Q_2)}{Q} - I)I_1 - \frac{pIQ_1}{Q} \geq 0; \tag{5.9}$$

解此不等式，可以得到

$$I_1 \geq r_1 \quad or \quad I_1 \leq r_1'; \tag{5.10}$$

$$r_1 = \frac{I - \frac{p(Q_1+Q_2)}{Q} + \sqrt{(\frac{p(Q_1+Q_2)}{Q} - I)^2 + \frac{4pIQ_1}{Q}}}{2} \tag{5.11}$$

$$r_1' = \frac{I - \frac{p(Q_1+Q_2)}{Q} - \sqrt{(\frac{p(Q_1+Q_2)}{Q} - I)^2 + \frac{4pIQ_1}{Q}}}{2} \tag{5.12}$$

很明显，$r_1' < 0$，因此$I_1 \geq r_1$是式5.11的唯一合理解。为实现最小化T_2的优化目标，应该让$I_1 = r_1$且$I_2 = r_2 = I - r_1$，这是源切换问题的理论最优解。

5.2.4 快速切换算法

上一小节得出了源切换问题的理论最优解，但当应用到实际P2P流媒体系统中时，上述理论最优解通常不能完全适用，主要原因在于实际的互联网环境比模型中的环境要复杂的多，并且面临更多的限制性条件。因此，我们需要设计一个能趋近理论最优解的实用算法。

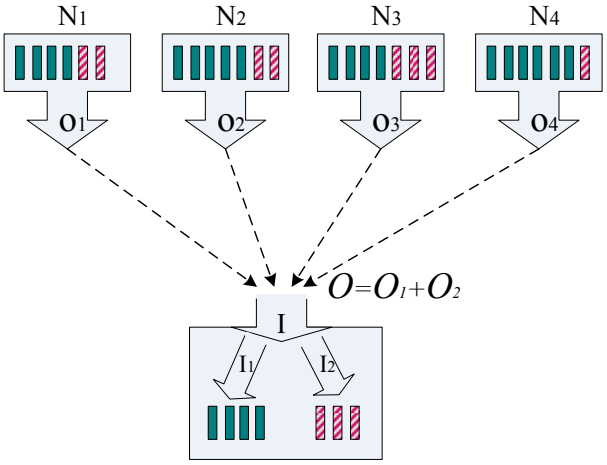


图 5.22　结点的局部工作环境示意图

图5.22描画了P2P流媒体系统中一个普通结点的局部工作环境。本地结点有4个邻居：N_1, N_2, N_3, N_4，4个邻居的输出带宽分别是o_1, o_2, o_3, o_4。假设O_1是针对源S_1的总输出带宽，O_2是针对源S_2的总输出带宽，那么上一小节中的数学优化问题可进一步归结为：

优化目标：最小化T_2

$$\text{限制条件：}\begin{cases} I_1 + I_2 \le I; \\ I_1 \le O_1; \\ I_2 \le O_2; \\ T_1 = \frac{Q_1}{I_1}; \\ T_1' = T_1 + \frac{Q}{p}; \\ T_2 = \frac{Q_2}{I_2}; \\ T_2 \ge T_1'; \end{cases}$$

在增加了更多的限制条件后，上一小节得出的理论最优解$I_1 = r_1, I_2 = r_2$仅当$r_1 \le O_1$且$r_2 \le O_2$时才成立。因此，当$r_1 > O_1$或$r_1 > O_1$时，优化的目标将改

表 5.2　快速源切换算法的相关参数

参数	说明
τ	数据调度周期。
id_i	数据分片D_i的id。
n_i	能够提供D_i的邻居数目。
R_{i_j}	从第j个提供者那里获取D_i的速率。
R_i	数据分片D_i的最大获取速率。
id_{play}	当前正在播放的数据分片的id。
id_{end}	S_1的最后一个数据分片的id。
id_{begin}	S_2的第一个数据分片的id。我们设定$id_{begin} = id_{end} + 1$。
t_i	数据分片D_i的过期时间。
B	缓存大小，即缓存中能放多少分片。
p_{i_j}	数据分片D_i在第j个提供者的缓存中的位置。缓存采用先进先出的替换策略，位置指的是到缓存尾部的距离。
$urgency_i$	数据分片D_i的紧迫性。
$rarity_i$	数据分片D_i的稀缺性。
$priority_i$	数据分片D_i的获取优先权。

为最大化本地结点的输入吞吐量，从而现实环境下的源切换问题的最优解是：

- 情形1：当$r_1 \leq O_1$且$r_2 \leq O_2$时，$I_1 = r_1, I_2 = r_2$;
- 情形2：当$r_1 \leq O_1$且$r_2 > O_2$时，$I_1 = \min(O_1, I - O_2), I_2 = O_2$;
- 情形3：当$r_1 > O_1$且$r_2 \leq O_2$时，$I_1 = O_1, I_2 = \min(O_2, I - O_1)$;
- 情形4：当$r_1 > O_1$且$r_2 > O_2$时，$I_1 = O_1, I_2 = O_2$;

这样，计算源切换问题的最优解的关键是首先计算出O_1和O_2，更确切地，从数据分片的微观角度来看，是计算两个集合$\mathbb{O}_1$和$\mathbb{O}_2$，其中$O_1 = |\mathbb{O}_1|$，$O_2 = |\mathbb{O}_2|$。集合$\mathbb{O}_1$和$\mathbb{O}_2$中的数据分片都按照获取优先权降序排列。计算集合$\mathbb{O}_1$和$\mathbb{O}_2$所需要的参数列举在表5.2中，实际上这些参数正是我们下面将要设计的快速源切换算法的相关参数。

在计算一个数据分片的获取优先权时，我们综合考虑了数据分片的稀缺性与紧迫性。首先计算数据分片D_i的紧迫性：

$$R_i = \max\{R_{i_1}, R_{i_2}, \cdots, R_{i_{n_i}}\} \tag{5.13}$$

$$t_i = \frac{id_i - id_{play}}{p} - \frac{1}{R_i} \tag{5.14}$$

$$\text{那么} \quad urgency_i = \frac{1}{t_i}. \tag{5.15}$$

我们将数据分片i的稀缺性理解为它在其所有提供者的缓存中都被替换掉的概率，这比过去的文献中普遍将分片的稀缺性理解为$rarity_i = \frac{1}{n_i}$要更为合理：

$$rarity_i = (\frac{p_{i_1}}{B}) \times (\frac{p_{i_2}}{B}) \times \cdots \times (\frac{p_{i_{n_i}}}{B}) \tag{5.16}$$

最终数据分片i的获取优先权为：

$$priority_i = \max\{urgency_i, rarity_i\}" \tag{5.17}$$

得出每个数据分片的获取优先权后，我们设计的快速源切换算法就能计算出集合$\mathbb{O}_1$和$\mathbb{O}_2$、从而安排数据分片的获取过程，见算法1。数据分片按优先权降序排列，比如排列成形如$D_1, D_2, D_3, \cdots, D_m$，通常来自源$S_1$和源$S_2$的数据分片在这个序列中是交错排列的。对一个数据分片D_i来说，可能有多个邻居能提供D_i，通常会选择能够最快传送D_i的邻居作为D_i的提供者。然而，这样的选择方式可能会出现冲突，比如当两个数据分片选择了同一个邻居作为提供者时，其中一个数据分片必须等待或者重选提供者。因此，为数据分片选择合适的提供者可表述为如下调度问题：如何为每一个数据分片选择一个合适的提供者，以使得过期或被替换的数据分片数目最少？实际上，即使是这个问题的一个简化版的特例（并行机调度问题）都已被证明是NP难的 [120]，因此我们使用算法1所示的贪心调度算法来试图尽早取得高优先权的数据分片，而不必追求总体最优的效果。

计算出集合$\mathbb{O}_1$和$\mathbb{O}_2$之后，I_1和I_2的计算按照前面讲过的最优解的4种情形之一来计算，这样数据分片的获取过程是很直接的。

5.2.5 性能评估

（a）实验环境

为了评价快速源切换算法的性能，我们在30幅真实P2P网络拓扑上进行了模拟实验。这些拓扑数据来自于http://dss.clip2.com上收集的无结构P2P网络拓扑测量结果（遗憾的是，该网站目前已不可用）。拓扑数据中包含每个结点

Algorithm 1 快速源切换算法

输入：
按照获取优先权降序排列的数据分片$D_1, D_2, D_3, \cdots, D_m$;
每个数据分片的提供者集合$S_1, S_2, S_3, \cdots, S_m$;
结点j的发送数据率R(j);
结点j处的预期排队时间$\tau(j)$，初始时$\tau(j)$=0.

步骤1：计算集合$\mathbb{O}_1$和$\mathbb{O}_2$
for $i = 1$ to m **do**
  设置D_i的最早获取时间$t_{min} = \infty$;
  假设S_i中包含k个提供者$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$;
  for $j = 1$ to k **do**
    计算从S_{i_j}处获取S_{i_j}的预期时间: $t_{trans} = \frac{1}{R(S_{i_j})}$;
    if $t_{trans} + \tau(S_{i_j}) < t_{min}$ **and** $t_{trans} + \tau(S_{i_j}) < \tau$ **then**
      $t_{min} \leftarrow t_{trans} + \tau(S_{i_j})$; $supplier_i \leftarrow S_{i_j}$;
    end if
  end for
  if $supplier_i \neq null$ **then**
    $\tau(supplier_i) \leftarrow t_{min}$;
    将D_i加入相应的集合$\mathbb{O}_1$或$\mathbb{O}_1$;
  end if
end for

步骤2：安排数据获取
根据$\mathbb{O}_1$、$\mathbb{O}_2$、r_1、r_1计算I_1和I_2;
获取$\mathbb{O}_1$中的前$\mathbb{O}_1$个数据分片;
获取$\mathbb{O}_2$中的前$\mathbb{O}_2$个数据分片;

的ID、IP地址、端口号、PING时间（从某个测量结点发出PING消息）等信息，我们仅使用ID、IP地址和PING时间信息。这30 幅网络拓扑的结点规模从100到10000不等，平均结点度从1到3.5不等，因为这样的平均结点度对流媒体系统来说过低，所以我们在拓扑中随机加入了一些边让每个结点连接$M = 5$个邻居。根据模拟实验的经验，$M = 5$是一个比较合适的选择，使用更大的M并不能带来更多好处。

流媒体发布速率是300 Kbps，每个数据分片包含30 Kb，因此播放速率$p = \frac{300}{30} = 10$。每个结点维护包含$B = 600$个数据分片的缓存，相当于60秒的流媒体数据。结点的接收带宽分布在300 Kbps 到1 Mbps之间，平均接收带宽为450 Kbps，也就是说$I \in [10, 33]$，平均$I = 15$。结点的发送带宽和接收带宽

具有类似的分布，唯一的例外是媒体源接收带宽为0而发送带宽很高，通常达到$I = 100$。数据调度周期$\tau = 1$秒。

在每次模拟中，首先让系统运行足够的时间以到达其稳定状态，然后停止旧源的数据发布、启动新源的数据发布。需要注意的是，在下文的每幅实验结果图中，时刻“0”均指旧源S_1停止、新源S_2启动的时刻。每当收集到来自S_1的$Q = 10$个连续的数据分片时，就播放S_1，但是为启动S_1的播放，则需要获取$Q_s = 50$个数据分片。

我们比较了快速源切换算法和前文中讲过的传统源切换算法。这里再重复一次传统源切换算法的工作原理：假设本地结点为n，当n的邻居可以向n提供来自S_1和S_2的数据分片时，n将优先获取来自S_1的数据。如果在获取S_1的数据之后仍然有剩余输入带宽，就把剩余的输入带宽分配给S_2的数据获取。

（b）性能指标

我们主要使用下面3个性能指标来评价源切换算法：

(1) *新源S_2的平均准备时间*，即平均源切换时间：所有结点准备足够的S_2数据分片来启动S_2播放的平均时间；
(2) *切换时间减少比率*：快速源切换算法比传统算法减少的切换时间比率；
(3) *通信开销*：在每个调度周期结点和其邻居交换数据可用性信息，*通信开销*定义为交换数据可用性信息的开销占实际流媒体数据传输开销的比例。

此外，还测量了其它一些辅助的性能指标来帮助更好地理解源切换过程。

（c）静态环境的实验结果

我们首先对1000个结点的静态网络环境下的数据传递比率进行跟踪，从而在细致、微观的程度上对比快速源切换算法与传统源切换算法的性能差异。从图5.23和图5.24的实验结果中可以看出，传统源切换算法能够更快地收集到播放S_1所需要的数据，但是在收集启动S_2放所需数据的方面却要慢一些，这与直观上的感觉是一致的，也就是说，快速源切换算法折中了收集S_1数据的时间与收集S_2数据的时间，从而使整个源切换过程加快。

我们进一步测量了静态网络环境下旧源S_1的平均完成时间与新源S_2的平均准备时间，结点规模从100到8000不等，以使得测量结果更具通用性。图5.25中

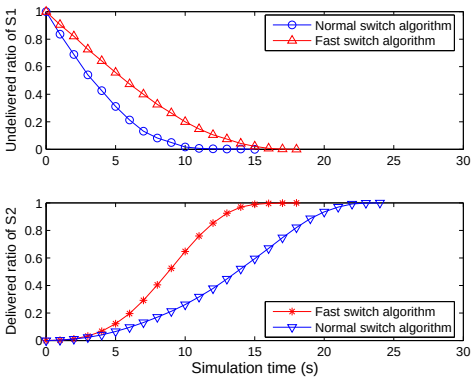


图 5.23 包含1000个节点的静态网络的数据传递比率

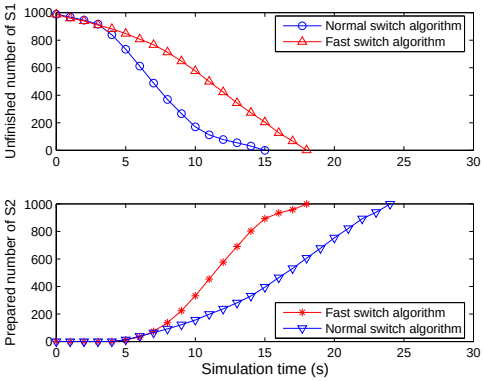


图 5.24 包含1000个节点的静态网络的数据块传递数目

的柱状图描画了测量结果。对于每个规模的网络环境，从左到右都有4条方柱对应4项指标：1）使用传统源切换算法，旧源S_1的平均完成时间；2）使用快速源切换算法，旧源S_1的平均完成时间；3）使用快速源切换算法，新源S_2的平均准备时间；4）使用传统源切换算法，新源S_2的平均准备时间。在每个规模的网络环境下，实验结果都趋于一致：快速源切换算法缩小了旧源S_1的平均完成时间与新源S_2的平均准备时间之间的差异，从而减少了源切换时间。为了更清晰地描画出快速源切换算法比传统源切换算法减少的切换时间，我们将两种算法各自的切换时间及减少的比率描画在图5.26中，可以看出：快速源切换算法相比传统源切换算法能减少约20%-30%的切换时间，并且减少的比率随着网络规模的增大而趋于增加。

此外，我们还测量了两种算法在不同规模的网络中运行的通信开销。结点缓存$B = 600$个数据分片，所以我们采用600比特的位图来记录缓存的数据可用性信息，比特1代表该分片可用，比特0代表该分片不可用。缓存的第一个分片需要20比特来记录其信息，因为数据源在一天内最多发布$10 \times 3600 \times 24 = 864000 \in (2^{19}, 2^{20})$个数据分片。因此，获取一个邻居的数据可用性信息总共需要620比特的通信开销。每个数据分片包含30 Kb的数据，假设每个结点每秒能获得$p = 10$个数据分片，这意味着每个结点的播放连续度都为1.0，那么控制开销将是$\frac{620 \times M}{30 \times 1024 \times 10} = \frac{5}{495} \approx 1\%$。图5.27所示的通信开销比1%要略高，因为实际上很少有结点的数据获取速率能始终保持数据播放速率的水平。快速源切换算法的通信开销比传统源切换算法略低，因为前者通过缩

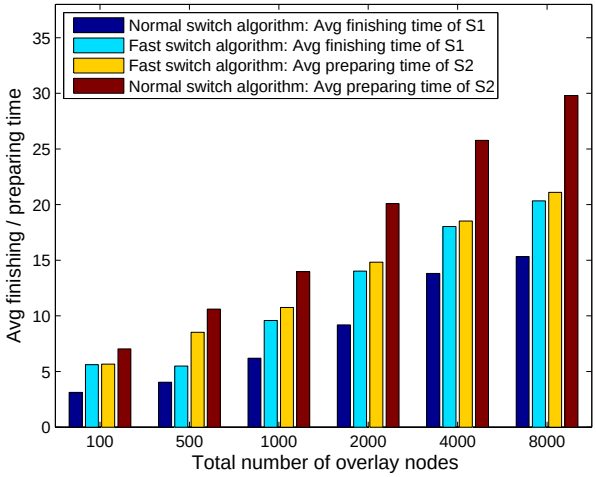


图 5.25　静态环境下旧源S_1的平均完成时间和新源S_2的平均准备时间

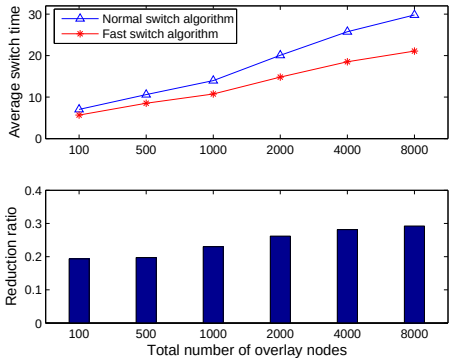


图 5.26　静态环境下的平均切换时间和减少比率

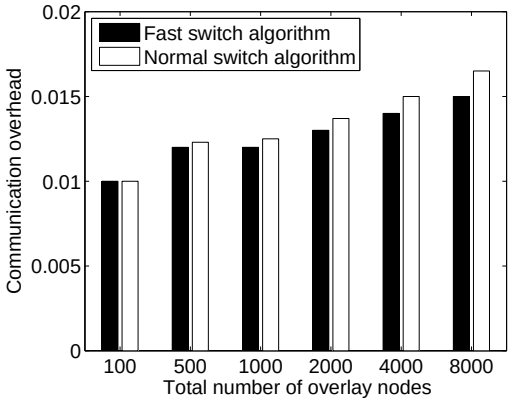


图 5.27　静态环境下的通信开销

短源切换时间的努力，间接地提高了带宽利用率。

（d）动态环境的实验结果

为了构建一个动态网络环境，每个周期随机让5%的结点失效、再让5%的新结点加入，新加入结点不需要获取过去播放过的所有数据，它们只需要获取其邻居结点正在播放或将要播放的数据，也就是说新加入结点是跟随其邻居的播放步伐。动态网络环境下的实验步骤与静态网络环境下的基本相同，实验结果在图5.28、5.29、5.30、5.31和5.32中。总体上看，动态网络环境下的实验结

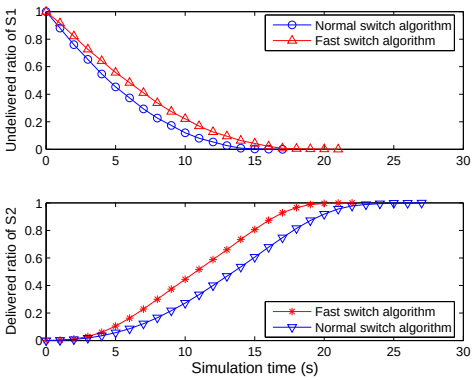


图 5.28　包含1000个节点的动态网络的数据传递比率

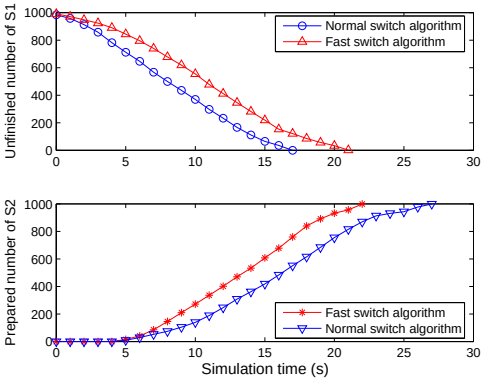


图 5.29　包含1000个节点的动态网络的数据块传递数目

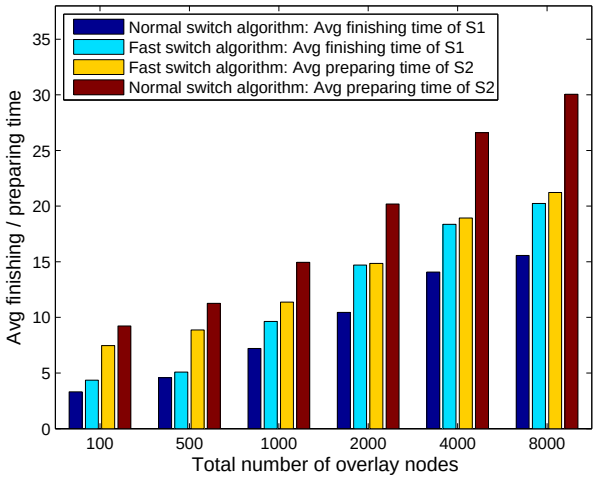


图 5.30　动态环境下旧源S_1的平均完成时间和新源S_2的平均准备时间

果和静态网络环境下的基本一致，说明我们设计的算法对网络环境具有普适性。

5.2.6 小结和进一步工作

本文给P2P流媒体系统的媒体发布源切换过程建立了数学模型，从而将源切换问题形式化为一个数学优化问题，然后推导出此数学优化问题的最优解。鉴于实际系统情况的复杂性与网络环境的动态性，我们提出了快速源切换算法这一实用贪心算法，它通过交错旧源与新源的数据传递来趋近理论上的最优解。

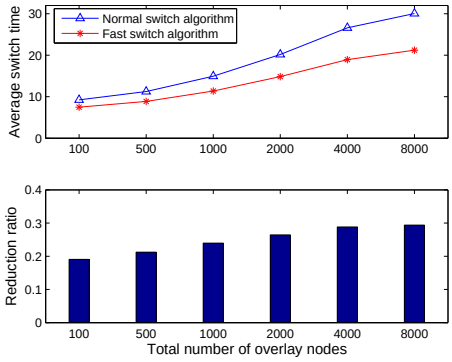


图 5.31　动态环境下的平均切换时间和减少比率

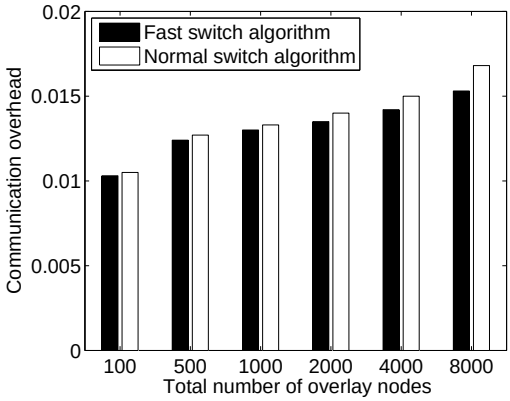


图 5.32　动态环境下的通信开销

模拟实验的结果证实了快速切换算法的有效性。本文的工作针对的是发布源之间串行工作的多源P2P流媒体系统，下一步我们希望能扩展本文的工作到发布源之间并行工作的多源P2P流媒体系统中。

第六章　结束语

本文是我读博4年研究工作的汇总，它们并非在4年前已有规划，而是一段边做边想、边干边学、不断反思和转换的过程，在结语中大致回顾研究的线索如下：

- 读博的第一年尚未意识到“云计算”对互联网内容分发可能产生的巨大影响和积极意义，一直处于探索方向和寻找课题的状态，其间做了2份较为传统的研究工作、即P2P流媒体系统的“优分组”和“快切换”，研究平台为中小规模的AmazingStore和CoolFish系统。
- 读博的第二和第三年完全转入“基于云计算的互联网内容分发”，并意识到互联网“重云轻端”的两极分化是必然趋势、而研究的难点在于处理用户端的高度异构性，其间做了4份较为创新的研究工作、即“云跟踪”“云调度”“云下载”和“云转码”，研究平台主要为大规模的腾讯QQ旋风系统。
- 读博的最后一年发现云存储服务在全球范围内迅速流行、但“云存储中的内容分发”却问题很多，其中不乏具有深刻意义的机制设计问题，是一个值得投入的前沿课题，所以又做了2份工作、即“云同步”和“云节流”，广泛研究了以Dropbox、Box、Google Drive、OneDrive、Ubuntu One和SugarSync等为代表的多个主流云存储系统。

互联网内容分发是一个历史悠久但日新月异的基础研究领域，它不断受惠于其它科学领域的成果（比如光纤的发明、交换机的革新和未来可能走向实用的量子通信），也影响了整个人类社会的生活方式（比如在线视频、电话会议、云端办公等）。结合本文的工作，我们认为下面几个问题还值得进一步研究：

- **CDN租用云进行内容分发**。过去的十多年一直是大型网站租用CDN的内容分发服务，本文1.3节也讲过Hulu公司租用多家CDN进行内容分发的例子。反过来，CDN租用云进行内容分发不仅可能、而且必需，因为CDN也要利用云计算的弹性自扩展功能灵活地调整其设施投入和服务质量[121]。
- **更节流的差分同步算法**。Dropbox、SugarSync等云存储系统所采用的rsync差分同步算法在面对大多数数据更新模式时可以有效节省流量，因

为rsync一般只传输新文件和旧文件之间的二值差分。虽然如此，rsync算法计算二值差分是基于定长数据块的，一旦数据块的长度选取的不好、或者具体数据更新模式恰好和所选取的数据块长度“错位”，rsync的节流效果可能十分微弱——比如我们使用Adobe Acrobat 软件从一本PDF格式的电子书中删除一页，可以观察到Dropbox所产生的数据流量常常和整个电子书的容量相当。数据存储领域曾经使用过cdc 算法来高效消重 [122]，但至今尚未见到cdc在云存储内容分发中的应用，可能的原因有二：（1）cdc算法比rsync算法复杂得多，实现难度大；（2）诸如Dropbox这样的系统如果想从rsync切换到cdc不能一蹴而就、需要一个较长的切换过程。这两方面原因都值得深入探究。

- **操作系统对内容分发的支持和影响**。本文对内容分发的研究主要在Windows和Linux两个最流行的操作系统上开展，但我们已经观察到：很多系统内容分发的性能不仅和网络状况相关、还和操作系统（特别是文件系统）的特性密不可分。比如说，同样的设备、同样的网络环境、处理同样的数据更新模式，Dropbox在Linux、Windows、Mac OS三个操作系统上的表现就很不一样。此外，最近几年愈发流行的几个移动操作系统：iOS、Android与Windows Phone和传统的PC操作系统从设计到实现都有巨大差别。因此，进一步深入研究操作系统对内容分发的支持和影响，在有利于内容分发的同时，操作系统亦可从中受惠。
- **SDN（软件定义网络）中的内容分发**。最近几年，一个新名词“软件定义网络”（Software Defined Network、即SDN）在学术界和工业界掀起研究热潮，其中最著名的斯坦福“Openflow” [123]技术方案已经被思科、华为、NEC等多家网络巨头采纳。过去几十年里，作为互联网内容分发的基础支撑设施，路由器和交换机在人们心中一直是个“硬件”概念——厂商一旦生产出来其功能就基本固化，用户一旦购买就只能进行最简单的配置操作。然而SDN的出现革新了这个概念，它认为路由器和交换机应该同个人电脑一样支持丰富的“软件”功能，可以在其上进行灵活的编程和广泛的实验，而最终目标是重塑整个互联网内容分发的形态。

参考文献

[1] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. In *Proceedings of the 17th ACM International Conference on Multimedia (ACM-MM)*. Oct. 19-23, 2009, Beijing, China.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM (CACM)*, 53(4):50–58, 2010.

[3] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*. Nov. 1-3, 2010, Melbourne, Australia.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*. Oct. 19-22, 2003, Bolton Landing, NY, USA.

[5] V.K. Adhikari, S. Jain, G. Ranjan, and Z.-L. Zhang. Understanding Data-center Driven Content Distribution. In *Proceedings of the ACM CoNEXT Student Workshop*. Nov. 30 - Dec. 3, 2010, Philadelphia, USA.

[6] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen. A Server's Perspective of Internet Streaming Delivery to Mobile Devices. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM)*. Mar. 25-30, 2012, Orlando, FL, USA.

[7] Y. Liu, L. Guo, F. Li, and S. Chen. An Empirical Evaluation of Battery Power Consumption for Streaming Data Transmission to Mobile Devices. In *Proceedings of the 19th ACM International Conference on Multimedia (ACM-MM)*. Mar. 25-30, 2011, Scottsdale, USA.

[8] V.K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z.-L. Zhang. A Tale of Three CDNs: An Active Measurement Study of Hulu and its CDNs. In *Proceedings of the 15th IEEE Global Internet Symposium*. Mar. 25-30, 2012, Orlando, FL, USA.

[9] V.K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. Vivisecting Youtube: An Active Measurement Study. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM) Mini-conference*. Mar. 25-30, 2012, Orlando, FL, USA.

[10] V.K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM)*. Mar. 25-30, 2012, Orlando, FL, USA.

[11] M. Meulpolder, L. D'Acunto, M. Capota, M. Wojciechowski, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips. Public and Private BitTorrent Communities: A Measurement Study. In *Proceedings of the 9th International Workshop on Peer-to-peer Systems (IPTPS)*. Apr. 27, 2010, San Jose, USA.

[12] Z. Liu, P. Dhungel, D. Wu, C. Zhang, and K.W. Ross. Understanding and Improving Incentives in Private P2P Communities. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 21-25, 2010, Genoa, Italy.

[13] C. Zhang, P. Dhungel, D. Wu, and K.W. Ross. Unraveling the BitTorrent Ecosystem. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(7):1164–1177, 2011.

[14] D.S. Menasche, A.A.A Rocha, B. Li, D. Towsley, and A. Venkataramani. Content Availability and Bundling in Swarming Systems. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. Dec. 1-4, 2009, Rome, Italy.

[15] N. Lev-tov, N. Carlsson, Z. Li, C. Williamson, and S. Zhang. Dynamic File-selection Policies for Bundling in BitTorrent-like Systems. In *Proceedings of the 18th IEEE/ACM International Workshop on Quality of Service (IWQoS)*. Jun. 16-18, 2010, Beijing, China.

[16] J. Han, S. Kim, T. Chung, T.T. Kwon, H. Kim, and Y. Choi. Bundling Practice in BitTorrent: What, how, and why. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. June. 11-15, 2012, London, UK.

[17] Y. Huang, T.Z.J. Fu, D.M. Chiu, J. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data communication (SIGCOMM)*. Aug. 17-22, 2008, Seattle, WA, USA.

[18] C. Wu, B. Li, and S. Zhao. On Dynamic Server Provisioning in Multi-channel P2P Live Streaming. *IEEE/ACM Transactions on Networking (TON)*, 19(5):1317–1330, 2011.

[19] Z. Li, Y. Huang, G. Liu, F. Wang, Y. Liu, Z.-L. Zhang, and Y. Dai. Challenges, Designs and Performances of Large-scale Open-P2SP Content Distribution. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 24(11):2181–2191, 2013.

[20] P. Dhungel, K. Ross, M. Steiner, Y. Tian, and X. Hei. Xunlei: Peer-assisted Download Acceleration on a Massive Scale. In *Proceedings of the 13th Passive and Active Measurement conference (PAM)*. March 12-14, 2012, Vienna, Austria.

[21] C. Ly, C.H. Hsu, and M. Hefeeda. Improving Online Gaming Quality using Detour Paths. In *Proceedings of the 18th ACM International Conference on Multimedia (ACM-MM)*. Oct. 25-29, 2010, Firenze, Italy.

[22] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proceedings of the 8th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Dec. 10-13, 2012, Nice, France.

[23] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaption in Dynamic HTTP Streaming. In *Proceedings of the 8th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Dec. 10-13, 2012, Nice, France.

[24] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B.Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai. Efficient Batched Synchronization in Dropbox-like Cloud Storage Services. In *Proceedings of the 14th ACM/IFIP/USENIX International Middleware Conference (Middleware) Regular Paper*. Dec. 9-13, 2013, Beijing, China.

[25] Z. Li and J. Li. Deficiency of Scientific Research behind the Price War of Cloud Storage Services（云存储价格战背后的科研缺失）. *Communications of China Computer Federation (CCCF《中国计算机学会通讯》) 封面文章*, 10(8):36–41, 2014.

[26] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang. Towards Network-level Efficiency for Cloud Storage Services. In *Proceedings of the 14th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC) Long Paper*. Nov. 5-7, 2014, Vancouver, Canada.

[27] Z. Li, Z.-L. Zhang, and Y. Dai. Coarse-grained Cloud Synchronization Mechanism Design May Lead to Severe Traffic Overuse. *Journal of Tsinghua Science and Technology (《清华学报》英文版)*, 18(3):286–297, 2013.

[28] Z. Li, Y. Huang, G. Liu, and Y. Dai. CloudTracker: Accelerating Internet Content Distribution by Bridging Cloud Servers and Peer Swarms. In *Proceedings of the 19th ACM International Conference on Multimedia (ACM-MM) Doctoral Symposium*. Nov. 28 - Dec. 1, 2011, Scottsdale, Arizona, USA.

[29] Z. Li, T. Zhang, Y. Huang, Z.-L. Zhang, and Y. Dai. Maximizing the Bandwidth Multiplier Effect for Hybrid Cloud-P2P Content Distribution. In *Proceedings of the 20th IEEE/ACM International Workshop on Quality of Service (IWQoS) Full paper*. Jun. 4-5, 2012, Coimbra, Portugal.

[30] Z. Li, Y. Huang, and Y. Dai. Construction of Tencent's Video Cloud and Its Implications for IOT&WSN. In *Proceedings of the 4th International Workshop on Internet of Things and Wireless Sensor Network (IOT&WSN)*. Dec. 22-25, 2012, Wuxi, Jiangsu, China.

[31] Y. Huang, Z. Li, G. Liu, and Y. Dai. Cloud Download: Using Cloud Utilities to Achieve High-quality Content Distribution for Unpopular Videos. In *Proceedings of the 19th ACM*

International Conference on Multimedia (ACM-MM) Long Paper. Nov. 28 - Dec. 1, 2011, Scottsdale, Arizona, USA.

[32] Z. Li, G. Liu, Z. Ji, and R. Zimmermann. Towards Cost-effective Cloud Downloading with Tencent Big Data. *Journal of Computer Science and Technology (JCST)*, 2015.

[33] Z. Li, C. Wilson, T. Xu, Y. Liu, Z. Lu, and Y. Wang. Offline Downloading in China: A Comparative Study. In *Proceedings of the 15th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC) Long Paper*. Oct. 28-30, 2015, Tokyo, Japan.

[34] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices. In *Proceedings of the 22nd SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. Jun. 7-8, 2012, Toronto, Canada.

[35] Z. Li, J. Wu, J. Xie, T. Zhang, G. Chen, and Y. Dai. Stability-Optimal Grouping Strategy of Peer-to-Peer Systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(12):2079–2087, 2011.

[36] J. Xie, Z. Li, J. Wu, and G. Chen. On Maximum Stability with Enhanced Scalability in High-Churn DHT Deployment. In *Proceedings of the 38th International Conference on Parallel Processing (ICPP)*. Sep. 22-25, 2009, Vienna, Austria.

[37] Z. Li, J. Cao, G. Chen, and Y. Liu. On the Source Switching Problem of Peer-to-Peer Streaming. *Journal of Parallel and Distributed Computing (JPDC)*, 70(5):537–546, 2010.

[38] Z. Li, J. Cao, G. Chen, and Y. Liu. Fast Source Switching for Gossip-based Peer-to-Peer Streaming. In *Proceedings of the 37th International Conference on Parallel Processing (ICPP)*. Sep. 8-12, 2008, Portland Oregon, USA.

[39] Dropbox user number news. "Dropbox is now the data fabric tying together devices for 100M registered users who save 1B files a day". In *http://techcrunch.com/2012/11/13/dropbox-100-million*.

[40] Binary diff. *http://en.wikipedia.org/wiki/Diff*.

[41] I. Drago, M. Mellia, M.M. Munafò, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 12th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*. Nov. 14-16, 2012, Boston, USA.

[42] Dropbox traces. *http://traces.simpleweb.org/wiki/Dropbox_Traces*.

[43] Wireshark. *http://www.wireshark.org*.

[44] inotify man page. *http://linux.die.net/man/7/inotify*.

[45] rsync web site. *http://www.samba.org/rsync*.

[46] W. Hu, T. Yang, and J.N. Matthews. The good, the bad and the ugly of consumer cloud storage. *ACM SIGOPS Operating Systems Review*, 44(3):110–115, 2010.

[47] H. Wang, R. Shea, F. Wang, and J. Liu. On the Impact of Virtualization on Dropbox-like Cloud File Storage/Synchronization Services. In *Proceedings of the 20th IEEE/ACM International Workshop on Quality of Service (IWQoS) Full paper*. Jun. 4-5, 2012, Coimbra, Portugal.

[48] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security & Privacy*, 8(6):40–47, 2010.

[49] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of Ownership in Remote Storage Systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, pages 491–500. ACM, 2011.

[50] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl. Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space. In *Proceedings of the 20th USENIX Conference on Security*, pages 5–5. USENIX Association, 2011.

[51] A. Bergen, Y. Coady, and R. McGeer. Client Bandwidth: The Forgotten Metric of Online Storage Providers. In *Proceedings of the 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pages 543–548. IEEE, 2011.

[52] K.R. Jackson et al. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–168. IEEE, 2010.

[53] B. Calder et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 143–157. ACM, 2011.

[54] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz. Design Implications for Enterprise Storage Systems via Multi-dimensional Trace Analysis. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 43–56. ACM, 2011.

[55] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of Backup Workloads in Production Systems. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*, pages 4–4. USENIX Association, 2012.

[56] M. Vrable, S. Savage, and G.M. Voelker. Cumulus: Filesystem Backup to the Cloud. *ACM Transactions on Storage (TOS)*, 5(4):14, 2009.

[57] M. Vrable, S. Savage, and G.M. Voelker. Bluesky: A Cloud-backed File System for the Enterprise. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*, pages 19–19. USENIX Association, 2012.

[58] P. Shilane, M. Huang, G. Wallace, and W. Hsu. WAN Optimized Replication of Backup Datasets using Stream-informed Delta Compression. volume 8, page 13. ACM, 2012.

[59] P. Mahajan et al. Depot: Cloud Storage with Minimal Trust. *ACM Transactions on Computer Systems (TOCS)*, 29(4):12, 2011.

[60] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-clouds. In *Proceedings of the 6th Conference on Computer Systems (EuroSys)*, pages 31–46. ACM, 2011.

[61] Dropbox CLI (Command Line Interface). *http://www.dropboxwiki.com/Using_Dropbox_CLI*.

[62] Ubuntu Dropbox client. *http://linux.dropbox.com/packages/ubuntu/nautilus-dropbox_0.7.1_i386.deb*.

[63] fsnotify. *https://github.com/howeyc/fsnotify*.

[64] How fast is SkyDrive growing? *http://www.liveside.net/2012/10/27/how-fast-is-skydrive-growing/*.

[65] Google Drive Now Has 10 Million Users: Available on iOS and Chrome OS. *http://techcrunch.com/2012/06/28/google-drive-now-has-10-million-users-available-on-ios-and-chrome-os-offline-editing-in-docs*.

[66] Why It's Dangerous to Rely on the Cloud. *http://itknowledgeexchange.techtarget.com/storage-disaster-recovery/why-its-dangerous-to-rely-on-the-cloud-at-least-if-you-use-comcast*.

[67] Cloud Storage Isn't Cheap: How the Price of Cloud Storage Compares to Traditional Storage. *http://www.nasuni.com/blog/39-cloud_storage_isnt_cheap_how_the_price_of_cloud*.

[68] What Happens When the Cloud Meets a Bandwidth Cap. *http://gigaom.com/2011/05/04/what-happens-when-the-cloud-meets-a-bandwidth-cap*.

[69] Hidden Costs of Cloud Storage. *http://www.onlinefilestorage.com/hidden-costs-of-cloud-storage-1756*.

[70] Bandwidth costs for cloud storage. *http://blog.dshr.org/2012/11/bandwidth-costs-for-cloud-storage.html*.

[71] Amazon S3 pricing policy (2013). *http://aws.amazon.com/s3/#pricing*.

[72] China Mobile's 18-card pupular wireless plan. *http://www.10086.cn/18card*.

[73] PUE (Power Usage Effectiveness). *http://en.wikipedia.org/wiki/Power_usage_effectiveness*.

[74] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking Personal Cloud Storage. In *Proceedings of the 13th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*. Oct. 23-25, 2013, Barcelona, Spain.

[75] A question about the default chunk size of rsync. *http://lists.samba.org/archive/rsync/2001-November/000595.html*.

[76] Ye Sun, Fangming Liu, Bo Li, Baochun Li, and Xinyan Zhang. FS2You: Peer-assisted semi-persistent online storage at a large scale. In *Proceedings of the 28th Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 873–881. Apr. 19-25, 2009, Rio de Janeiro, Brazil, 2009.

[77] PV (page view) wiki page. *http://en.wikipedia.org/wiki/Page_view*.

[78] PTC (paid-to-click) wiki pag. *http://en.wikipedia.org/wiki/Paid_To_Click*.

[79] NetSession. *http://www.akamai.com/client*.

[80] P. Aditya, M. Zhao, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, and B. Wishon. Reliable Client Accounting for P2P-Infrastructure Hybrids. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*. Apr. 25-27, 2012, San Jose, CA, US.

[81] M. Zhang, W. John, and C. Chen. Architecture and Download Behavior of Xunlei: A Measurement-based Study. In *Proceedings of the 2nd IEEE International Conference on Education Technology and Computer (ICETC)*. Jun. 22-24, 2010, Shanghai, China.

[82] M. Zhang, C. Chen, and N. Brownlee. A Measurement-based Study of Xunlei. In *Proceedings of the 10th PAM Student Workshop*, pages 1–3, 2009.

[83] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving Energy in Data Center Networks. In *Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation (NSDI)*. Apr. 28-30, 2010, San Jose, CA, US.

[84] Z. Liu, C. Wu, B. Li, and S. Zhao. UUSee: Large-scale Operational On-demand Streaming with Random Network Coding. In *Proceedings of the 29th Annual IEEE International Conference on Computer Communications (INFOCOM)*. Mar. 15-19, 2010, San Diego, CA, USA.

[85] Fangming Liu, Shijun Shen, Bo Li, Baochun Li, Hao Yin, and Sanli Li. Novasky: Cinematic-quality VoD in a P2P Storage Cloud. In *Proceedings of the 30th Annual IEEE International Conference on Computer Communications (INFOCOM)*. Apr. 10-15, 2011, Shanghai, China.

[86] R. Peterson and E. Sirer. Antfarm: Efficient Content Distribution with Managed Swarms. In *Proceedings of the USENIX Conference on Network System Design and Implementation (NSDI)*. Apr. 22-24, 2009, Boston, MA, US.

[87] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[88] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[89] L. Armijo. Minimization of Functions having Lipschitz Continuous First Partial Derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[90] CoolFish web site. *http://www.cool-fish.org*.

[91] CSTNet web site. *http://www.cstnet.net.cn*.

[92] The Cisco Visual Networking Index report. *http://newsroom.cisco.com/dlls/2008/ekits/Cisco_Visual_Networking_Index_061608.pdf*.

[93] J. Kangasharju, J. Roberts, and K.W. Ross. Object Replication Strategies in Content Distribution Networks. *Computer Communications*, 25(4):376–383, 2002.

[94] K. Roth and K. McKenney. *Energy Consumption by Consumer Electronics in US Residences*. Final Report to the Consumer Electronics Association (CEA), 2007.

[95] J.R. Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-peer Systems (IPTPS)*, pages 251–260. Cambridge, MA, March 2002.

[96] A. Singh, T.W. Ngan, P. Druschel, and D.S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–12. IEEE, 2006.

[97] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proceedings of the 1st International Workshop on Peer-to-peer Systems (IPTPS)*, pages 261–269. Cambridge, MA, March 2002.

[98] D. Wu, Y.T. Hou, W. Zhu, Y.Q. Zhang, and J.M. Peha. Streaming Video over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):282–300, 2001.

[99] H. Xie, Y.R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Provider Portal for Applications. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data communication (SIGCOMM)*, pages 351–362. Aug. 17-22, 2008, Seattle, WA, USA.

[100] D.R. Choffnes and F.E. Bustamante. Taming the Torrent: a Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data communication (SIGCOMM)*, pages 363–374. Aug. 17-22, 2008, Seattle, WA, USA, 2008.

[101] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.

[102] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 126–134, 1999.

[103] W.B. Croft, D. Metzler, and T. Strohman. *Search engines: Information retrieval in practice*. Addison-Wesley Reading, 2010.

[104] F. Wang, J. Liu, and Y. Xiong. Stable Peers: Existence, Importance, and Application in Peer-to-peer Live Video Streaming. In *Proceedings of the 27th International Conference on Computer Communications (INFOCOM)*, pages 1364–1372. IEEE, 2008.

[105] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 73–84. ACM, 2005.

[106] R. Matei, A. Iamnitchi, and P. Foster. Mapping the Gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[107] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160, 2001.

[108] P. Godfrey, S. Shenker, and I. Stoica. Minimizing Churn in Distributed Systems. In *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM)*. Sep. 11-15, 2006, Pisa, Italy.

[109] M.K.H. Yeung and Y.-K. Kwok. Game Theoretic Peer Selection for Resilient Peer-to-peer Media Streaming Systems. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS)*, pages 817–824. IEEE, 2008.

[110] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 492–499. IEEE, 2001.

[111] C.C. Wang and K. Harfoush. On the stability-scalability tradeoff of DHT deployment. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 2207–2215. IEEE, 2007.

[112] J. Kangasharju, K.W. Ross, and D.A. Turner. Optimizing file availability in peer-to-peer content distribution. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1973–1981. IEEE, 2007.

[113] S. Saroiu, P.K. Gummadi, and S.D. Gribble. Measurement Study of Peer-to-peer File Sharing Systems. In *Electronic Imaging 2002*, pages 156–170. International Society for Optics and Photonics, 2001.

[114] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-peer File-sharing Workload. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 314–329. ACM, 2003.

[115] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang. The Stretched Exponential Distribution of Internet Media Access Patterns. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 283–294. ACM, 2008.

[116] X. Zhang, J. Liu, B. Li, and T.S.P. Yum. CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming. pages 2102–2111, 2005.

[117] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng. AnySee: Peer-to-Peer Live Streaming. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–10, 2006.

[118] M. Zhang, J.G. Luo, L. Zhao, and S.Q. Yang. A Peer-to-peer Network for Live Media Streaming using a Push-pull Approach. *Proceedings of the 13th ACM International Conference on Multimedia (ACM-MM) Short Paper*, pages 287–290, 2005.

[119] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 363–371, 2002.

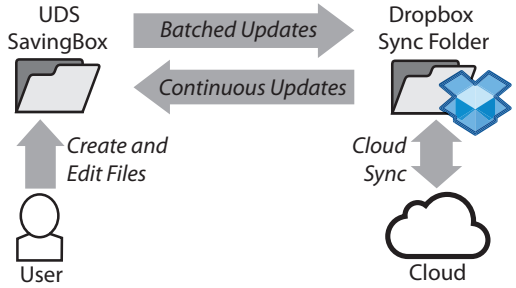
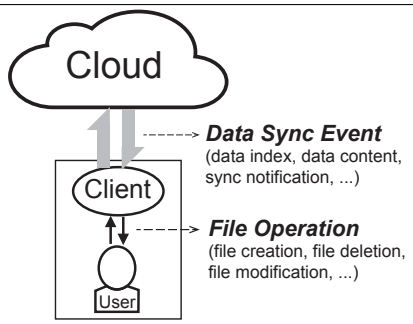
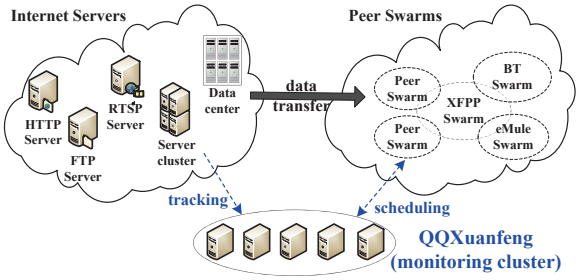
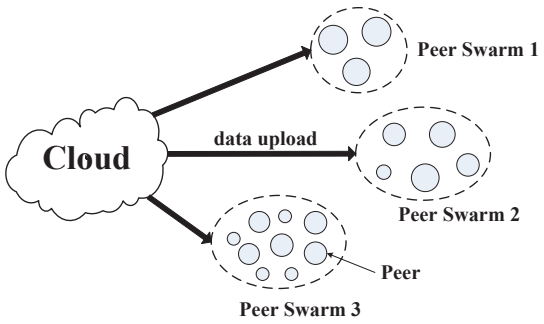
[120] T.T. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to algorithms. *MIT Press Cambridge, MA, USA*, 1990.

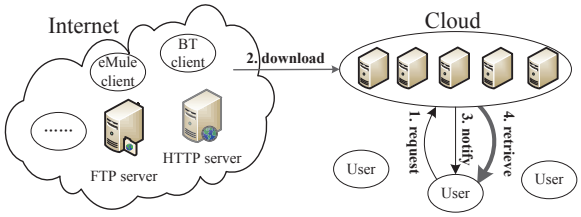
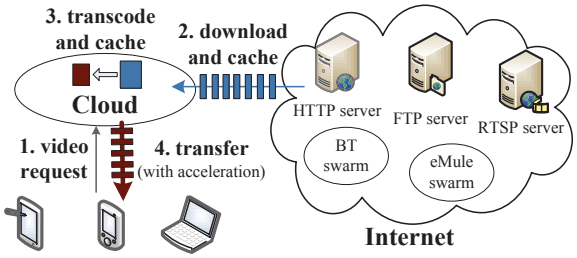
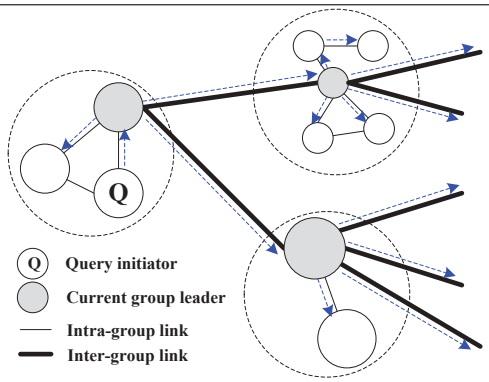
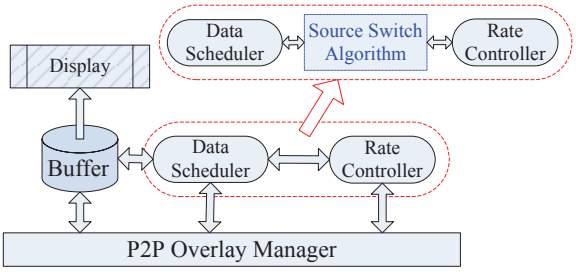
[121] H. Yin, X. Liu, G. Min, and C. Lin. Content Delivery Networks: A Bridge between Emerging Applications and Future IP Networks. *IEEE Network*, 24(4):52–56, 2010.

[122] A. Muthitacharoen, B. Chen, and D. Mazieres. A Low-bandwidth Network File System. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 174–187. ACM, 2001.

[123] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

附录 A　研究工作简图

工作名称 （简称）	背景系统 （及其链接）	工作原理图	发表论文 （第一/通讯作者）
云存储的高效批同步算法 （云同步）	Dropbox www.dropbox.com		Middleware'13 《中国计算机学会通讯》'14
云存储的节流效率研究 （云节流）	Google Drive Dropbox iCloud SkyDrive IDrive sync ubuntu one 百度云 360云盘 SugarSync		IMC'14 《清华学报》英文版'13
云跟踪系统的挑战设计与性能 （云跟踪）	QQ旋风 xf.qq.com 迅雷看看 www.kankan.com www.xunlei.com		*TPDS'13* ACM-MM'11 (Doctoral Symposium)
最大化带宽放大效应的云调度算法 （云调度）	QQ旋风 xf.qq.com FISH www.cool-fish.org		IWQoS'12 IOT&WSN'12

服务冷门视频的云下载系统 （云下载）	Download Offline 离线下载—专属带宽·无须挂机·更高速稳定 vip.qq.com/freedom/freedo_down_offline.html		IMC'15 ACM-MM'11 (Long Paper) *JCST'15*
适应移动视频的云转码系统 （云转码）	 xf.qq.com		NOSSDAV'12
稳定性最优的端用户分组策略 （优分组）	AmazingStore 共享潮流 保留经典 amazingstore.org FISH www.cool-fish.org		*TPDS'11* ICPP'09
流媒体数据源的最快切换策略 （快切换）	FISH www.cool-fish.org		*JPDC'10* ICPP'08

个人简历、在学期间的研究成果

个人简历

李振华，男，1983年8月7日出生于江苏省盐城市阜宁县

- 2001年考入南京大学计算机科学与技术系，分别于2005年、2008年获得学士和硕士学位；其间在香港理工大学计算机系访问半年；
- 2008年至2009年在Marvell微电子（上海）研发中心担任软件工程师，从事蓝光高清视频编解码芯片的研发；
- 2009年考入北京大学信息科学技术学院（网络与信息系统研究所），攻读博士学位至今；其间曾在腾讯研究院（上海）QQ旋风组实习，并在美国明尼苏达大学双城校区公派留学一年。

CCF、CAAI、ACM 和IEEE会员。国际会议***Middleware'15***、***WASA'15***、***APSCC'15***、***SIMPLEX'15***、***ICoC'15***、***CNCC'14***、***APSCC'14***、***SIMPLEX'14***、***ICNP'13 PhD Forum***、***ICPP'12***、***ICCCN'11*** 和***HotPOST'11*** 程序委员会成员（TPC Member）。读博期间连续4年获得“北京大学博士生校长奖学金”，2012年获得“北京大学学术创新奖”和“北京大学信息学院学术十杰奖”，2010年获得“董氏东方奖学金”。2015年获得（2014年度）“教育部自然科学一等奖”（第4完成人）。更多信息详见个人主页：http://www.greenorbs.org/people/lzh。

读博期间（包括博士毕业两年内）发表论文列表

*表示通讯作者，按时间逆序排列。

[1] Zhenhua Li, Gang Liu, Zhiyuan Ji*, and Roger Zimmermann. **Towards Cost-effective Cloud Downloading with Tencent Big Data**. *Journal of Computer Science and Technology* (***JCST***), Accepted in 2015. (SCI索引，影响因子：0.642)

[2] Zhenhua Li*, Christo Wilson, Tianyin Xu, Yao Liu, Zhen Lu, and Yinlong Wang. **Offline Downloading in China: A Comparative Study**. *The 15th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference* (***IMC***) Long Paper,

Oct. 28-30, 2015, Tokyo, Japan. (录取率：44/169 = 26%，**IMC**为计算机系统性能领域顶级会议)

[3] Quanlu Zhang, Shenglong Li, Zhenhua Li, Yuanjian Xing, Zhi Yang, and Yafei Dai. **CHARM: A Cost-efficient Multi-cloud Data Hosting Scheme with High Availability**. *IEEE Transactions on Cloud Computing* (***TCC***), Accepted in 2015. (**TCC**为云计算专业领域顶级期刊)

[4] Jian Li, Zhenhua Li*, Yao Liu, and Zhi-Li Zhang. **Do Twin Clouds Make Smoothness for Transoceanic Video Telephony?** *The 44th International Conference on Parallel Processing* (***ICPP***), Sep. 1-4, 2015, Beijing, China. (录取率：99/305 = 32.5%)

[5] Yiyang Zhao, Chen Qian, Liangyi Gong, Zhenhua Li*, and Yunhao Liu. **LMDD: Light-weight Magnetic-based Door Detection with Your Smartphone**. *The 44th International Conference on Parallel Processing* (***ICPP***), Sep. 1-4, 2015, Beijing, China. (录取率：99/305 = 32.5%)

[6] Yao Liu, Mengbai Xiao, Min Zhang, Xin Li, Mian Dong, Zhan Ma, Zhenhua Li, and Songqing Chen. **Content-Adaptive Display Power Saving in Internet Mobile Streaming**. *The 25th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video* (***NOSSDAV***), Mar. 20, 2015, Portland, Oregon, USA. (录取率：12/48 = 25%)

[7] Yao Liu, Sam Blasiak, Weijun Xiao, Zhenhua Li, and Songqing Chen. **A Quantitative Study of Video Duplicate Levels in YouTube**. *The 16th Passive and Acitve Measurements Conference* (***PAM***), Mar. 19-20, 2015, New York City, NY, USA. (录取率：27/100 = 27%)

[8] Xiaobo Ma, Junjie Zhang, Zhenhua Li, Jianfeng Li, Jing Tao*, Xiaohong Guan, John C.S. Lui, and Don Towsley. **Accurate DNS Query Characteristics Estimation via Active Probing**?. *Journal of Network and Computer Applications* (***JNCA***), Vol. 47, Jan. 2015, Elsevier, Pages 72-84. (SCI索引，影响因子：1.772)

[9] Jianming Lv*, Tieying Zhang, Zhenhua Li, and Xueqi Cheng. **PACOM: Parasitic Anonymous Communication in the BitTorrent Network**. *Computer Networks* (***COMNET***), Vol. 74, Part A, Dec. 2014, Elsevier, pp. 13-33. (SCI索引，影响因子：1.282)

[10] Zhenhua Li*, Cheng Jin, Tianyin Xu, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang. **Towards Network-level Efficiency for Cloud Storage Services**. *The 14th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference* (***IMC***) Long Paper, Nov. 5-7, 2014, Vancouver, Canada. (录取率：42/188 = 22%，本文为本届IMC会议上来自大中华区的唯一正式论文)

[11] Zhenhua Li* and Jian Li. **Deficiency of Scientific Research behind the Price War of Cloud Storage Services**（**云存储价格战背后的科研缺失**）. *Communications of China Computer Federation* (***CCCF***《*中国计算机学会通讯*》), Vol. 10, No. 8, Aug. 2014, pp. 36-41. (本文得到刊物主编李国杰院士的书面赞扬，被选为本期刊物的封面文章)

[12] Zhenhua Li*, Christo Wilson, Zhefu Jiang, Yao Liu, Ben Y. Zhao, Cheng Jin, Zhi-Li Zhang, and Yafei Dai. **Efficient Batched Synchronization in Dropbox-like Cloud Storage Services**. *The 14th ACM/IFIP/USENIX International Middleware Conference* (***Middleware***) Regular paper, Dec. 9-13, 2013, Beijing, China. (录取率：24/128 = 18.8%，**Middleware**为计算机系统中间件领域顶级会议，本文为本届会议上来自大中华区的唯一正式论文)

[13] Zhenhua Li*, He Xiao, Linsong Cheng, Zhen Lu, Jian Li, Christo Wilson, Yao Liu, Yunhao Liu, and Yafei Dai. **T-CloudDisk: A Tunable Cloud Storage Service for Flexible Batched Synchronization**. *The 14th ACM/IFIP/USENIX International Middleware Conference* (***Middleware***) Live Demo, Dec. 9-13, 2013, Beijing, China.

[14] Zhenhua Li*, Yan Huang, Gang Liu, Fuchen Wang, Yunhao Liu, Zhi-Li Zhang, and Yafei Dai. **Challenges, Designs and Performances of Large-scale Open-P2SP Content Distribution**. *IEEE Transactions on Parallel and Distributed Systems* (***TPDS***), Vol. 24, No. 11, Nov. 2013, pp. 2181-2191. (**TPDS**为并行与分布式系统领域顶级期刊，SCI索引，影响因子：1.992)

[15] Yifeng Yu, Zhenhua Li, Yuanjian Xing, and Yafei Dai*. **Efficient Metadata Management in Cloud Storage Data Deduplication**. *The 4th CCF National Conference on Service Computing* (***NCSC***), Aug. 4-6, 2013, Enshi, Hubei, China.

[16] Zhenhua Li, Zhi-Li Zhang, and Yafei Dai*. **Coarse-grained Cloud Synchro-**

nization Mechanism Design May Lead to Severe Traffic Overuse. *Journal of Tsinghua Science and Technology* (《清华学报》英文版), Vol. 18, No. 3, Jun. 2013, Elsevier. (EI索引)

[17] Zhenhua Li*, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. **Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices**. *The 22nd SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video* (***NOSSDAV***), Jun. 7-8, 2012, Toronto, Canada. (EI索引，录取率：17/47 = 36%)

[18] Zhenhua Li*, Tieying Zhang, Yan Huang, Zhi-Li Zhang, and Yafei Dai. **Maximizing the Bandwidth Multiplier Effect for Hybrid Cloud-P2P Content Distribution**. *The 20th IEEE/ACM International Workshop on Quality of Service* (***IWQoS***) Full paper, Jun. 4-5, 2012, Coimbra, Portugal. (EI索引，录取率: 24/110 = 21.8%)

[19] Yunqin Zhong*, Jizhong Han, Tieying Zhang, Zhenhua Li, Jinyun Fang, and Guihai Chen. **Towards Parallel Spatial Query Processing for Big Spatial Data**. *The International Workshop on High Performance Data Intensive Computing* (***HPDIC***), May 25, 2012, Shanghai, China. (EI索引)

[20] Tieying Zhang*, Xueqi Cheng, Jianming Lv, Zhenhua Li, and Weisong Shi. **Providing Hierarchical Lookup Service for P2P-VoD Systems**. *ACM Transactions on Multimedia Computing Communications and Applications* (***TOMCCAP***), Vol. 8, No. 1, Feb. 2012. (SCI索引)

[21] Zhenhua Li*, Jie Wu, Junfeng Xie, Tieying Zhang, Guihai Chen, and Yafei Dai. **Stability-Optimal Grouping Strategy of Peer-to-Peer Systems**. *IEEE Transactions on Parallel and Distributed Systems* (**TPDS**), Vol. 22, No. 12, Dec. 2011, pp. 2079-2087. (**TPDS**为并行与分布式系统领域顶级期刊，SCI索引，影响因子：1.992)

[22] Yan Huang, Zhenhua Li*, Gang Liu, and Yafei Dai. **Cloud Download: Using Cloud Utilities to Achieve High-quality Content Distribution for Unpopular Videos**. *The 19th ACM International Conference on Multimedia* (***ACM-MM***) Long Paper, Nov. 28 - Dec. 1, 2011, Scottsdale, Arizona, USA. (录取率：58/341 = 17%，**ACM-MM**为计算机多媒体领域顶级会议)（注：并列第一作者）

[23] Zhenhua Li*, Yan Huang, Gang Liu, and Yafei Dai. **CloudTracker: Accelerating Internet Content Distribution by Bridging Cloud Servers and Peer Swarms**. *The 19th ACM International Conference on Multimedia* (***ACM-MM***) Doctoral Symposium, Nov. 28 - Dec. 1, 2011, Scottsdale, Arizona, USA.

[24] Xu Zhang*, Zhenhua Li, Tieying Zhang, Liangpeng He, and Guihai Chen. **RELookup: Providing Resilient and Efficient Lookup Service for P2P-VoD Streaming**. *The 5th International Workshop on Peer-to-Peer Networked Virtual Environments* (***P2PNVE***), Dec. 7-9, 2011, Tainan, Taiwan. (EI索引)

[25] Yue Cao*, Zhenhua Li, and Yafei Dai. **Resource Storage Strategy On AmazingStore Off-line Download System**. *Journal of Chinese SciencePaper Online* (《中国科技论文在线》), Vol. 20, Sep. 2011. (被推荐为“精品论文”)

[26] Tieying Zhang*, Zhenhua Li, Huawei Shen, Yan Huang, and Xueqi Cheng. **A White-box Empirical Study of P2P-VoD Systems: Several Unconventional New Findings**. *The 20th IEEE International Conference on Computer Communication Network* (***ICCCN***), Jul. 31 - Aug. 4, 2011, Hawaii, USA. (EI索引，录取率：130/450 = 29%)

[27] Tieying Zhang*, Zhenhua Li, Xueqi Cheng, and Xianghui Sun. **Multi-Task Downloading for P2P-VoD: An Empirical Perspective**. *The 16th International Conference on Parallel and Distributed Systems* (***ICPADS***), Dec. 8-10, 2010, Shanghai, China. (EI索引)

[28] Yuanjian Xing*, Zhenhua Li, and Yafei Dai. **PeerDedupe: Insights into the Peer-assisted Sampling Deduplication**. *The 10th IEEE International Conference on Peer-to-Peer Computing* (***P2P***), Aug. 25-27, 2010, Delft, Netherland. (EI索引，录取率：27/121 = 22.3%)

[29] Zhenhua Li, Jiannong Cao*, Guihai Chen, and Yan Liu. **On the Source Switching Problem of Peer-to-Peer Streaming**. *Journal of Parallel and Distributed Computing* (***JPDC***), Vol. 70, No. 5, May 2010, pp. 537-546, Elsevier. (**JPDC**为并行与分布式系统领域知名期刊，SCI索引，影响因子：1.135)

[30] Zhenhua Li* (advisor: Yafei Dai). **10 years of P2P: where is it going?**（**P2P十年：何去何从？**）. *Communications of China Computer Federation* (***CCCF***《中国计算机学会通讯》), Vol. 6, No. 1, Jan. 2010, pp. 28-32.

[31] Junfeng Xie, Zhenhua Li*, Guihai Chen, and Jie Wu. **On Maximum Stability**

with Enhanced Scalability in High-Churn DHT Deployment. *The 38th International Conference on Parallel Processing* (***ICPP***), Sep. 22-25, 2009, Vienna, Austria. (EI索引，录取率：71/220 = 32.3%)

读博期间参与研究课题

[1] 国家重点基础研究发展计划（973）课题–2011CB302305：“**云存储服务和保障机制研究**”，2011 – 2015，参与；

[2] 国家自然科学基金课题–61073015：“**基于协作的云存储外延服务优化机制研究**”，2011 – 2013，参与；

[3] 国家自然科学基金课题–61073152：“**支持高质量P2P流媒体服务的有效技术的研究**”，2011 – 2013，参与；

[4] 国家重大科技专项（863）课题：“**新型移动业务控制网络的分布式用户数据管理技术研究**”，2010 – 2012，参与；

[5] 腾讯－北京大学联合项目：“**P2P核心算法研究**”，2011 – 2012，参与；

[6] 微软亚洲研究院－北京大学联合项目：“**基于协作的分布式计算**”，2010 – 2011，参与；

[7] 国家自然科学基金课题–60873051,“**P2P存储系统中可用性、可靠性和安全性问题研究**”, 2009 – 2011，参与。

致　谢

四年的读博岁月每每感觉无比漫长，而当写下这段致谢的时候又恍然倏忽间如白驹过隙。我的所有研究工作在浩瀚的学术界不过是沧海一粟，不管爱与不爱、将来很可能都是历史的尘埃；但这四年的经历于我的人生是一种修行、一段攀登和一次涅槃。　千千万万的中国博士生中我大概是比较幸运的一个，每当面临抉择、每逢遇到困难，常有贵人相助、逢凶化吉，所以我要真挚地感谢诸多前辈、朋友和同行。

首先感谢我的读博导师代亚非教授。2009年初我申请国内读博，联系了北大、清华、中科院诸多教授，代老师最快、最直接给我明确的答复，并一直帮助我获取历年考博材料、申请博士生校长奖学金。读博的四年里，代老师给予我广泛的学术指导，同时还帮助我与腾讯研究院合作开展研究、申请教育部公派留学项目、申请北京大学“学术创新奖”及信息学院“学术十杰奖”等。此外，代老师主持开发的AmazingStore文件分享系统也给我的科研活动提供了有力的支撑。在此一并感谢北大网络所的李晓明教授、张铭教授、高军教授、罗英伟教授、闫宏飞副教授、顾小林副教授、严伟副教授、孙艳春副教授、胡俊峰副教授、崔斌研究员、肖臻研究员，以及学院教务员田军老师、全秋燕老师、王卫红老师等。

感谢我的公派留学联合培养导师、美国明尼苏达大学的张志力教授（Zhi-Li Zhang，Qwest Chair Professor、McKnight Distinguished University Professor和IEEE Fellow）。在美国的一年里，张老师给予我最大程度的自由和信任，并一直向我传达学术研究的境界和品味，特别是不要为了发表论文而堆砌缺乏实际意义的理论和公式。

感谢我的工业界合作导师、腾讯研究院技术总监、QQ旋风项目组负责人黄琰先生（现已加盟“百度杀毒”）。和腾讯研究院合作开展研究给我带来一个坚实的平台和一系列学术成果。在此一并感谢腾讯研究院的分布式系统架构师刘刚先生、以及数据库系统架构师王福臣先生（现已加盟“百度杀毒”）。

感谢清华大学的刘云浩教授（IEEE Fellow）。2006年夏天在桂林第一次见到刘老师，那时他是香港科技大学的助理教授，我是屌丝；2012年秋天在北京再次见到刘老师，他已经是国际学术界的风云人物，我还是屌丝。极其难得的是，六年过去刘老师依然还能记得我，并将我招入清华大学工作，实属知遇之

恩。在此一并感谢清华大学的赵弋洋老师、何源老师、郭振格博士、翟季冬助理教授、崔琳老师、辛爽老师、张子慧老师、马志楠老师等。

感谢我的硕士阶段导师、上海交大计算机系副主任兼南京大学计算机系副主任陈贵海教授。多年来陈老师一直是我人生的“伯乐”，多次为我穿针引线、推荐去处，比如代老师、张老师、刘老师都是陈老师引荐我认识的。我读博期间和陈老师一直保持联系与合作，他在科研和生活上都给予我诸多关怀。在此一并感谢上海交大的吴帆副教授（也是我硕士师兄），以及南京大学的窦万春教授、路通副教授（也是我的表哥）、谢磊副教授、吴小兵助理教授（也是我硕士师兄）等。

感谢香港理工大学计算机系主任曹建农教授（IEEE Fellow）。我硕士阶段曾跟随曹老师学习数月，他对我态度宽厚而要求严格，促使我那段时间颇为高产。我读博期间和曹老师一直保持联系与合作，读博的最后一年曹老师还盛情邀请我做他的博后。在此一并感谢香港理工大学的刘焱副教授、楼炜助理教授。

感谢UCSB的赵燕斌副教授。我读博的第一年有幸给赵老师夫妇担任课程助教，领略了他们对学术前沿的生动理解；读博的最后一年再次和赵老师相遇并合作，他的观点总是犀利而切实。在此还要感谢赵老师的博士、美国东北大学的Christo Wilson助理教授，他对研究的细心和耐心、特别是对英文写作炉火纯青的把握，令我印象深刻。

感谢美国天普大学的计算机系主任吴杰教授（Laura H. Carnell Professor和IEEE Fellow）。吴老师对待每一篇合作的论文严谨求实、一丝不苟的治学态度令人佩服。

感谢多伦多大学的李葆春教授（Bell Chair Professor和IEEE Fellow）。李老师是网络通信领域的国际知名学者，我读博的第一年曾经写过一篇关于P2P研究的评论性文章，其中提到了李老师的工作，没想到发表不久李老师就主动联系我。读博期间和李老师一直保持联系，他对我的部分工作也有点拨，读博的最后一年还热情邀请我申请他的博后。在此一并感谢美国里海大学的Mooi Chuah副教授和香港中文大学的Patrick Lee 副教授邀请我申请他们的博后。

感谢新加坡国立大学的Roger Zimmermann教授。2011年末我在美国亚利桑那州参加*ACM-MM'11*会议并演讲我们的论文，Roger教授正是我所在的Session Chair，当他发现我因投影仪问题而困窘时立刻前来帮忙解决，并在演讲之后和

我专门讨论了我们的工作。半年以后，我和Roger教授合作另外一篇论文，他的谦逊和礼貌是对“绅士风度”的生动诠释。

感谢中科院计算所的张铁赢助理研究员，我读博期间和他的合作不仅愉快而且高产，此外他主持开发的CoolFish视频点播系统也给我的科研活动提供了有力的支撑。在此一并感谢中科院计算所的刘志勇教授、程学旗教授、李振宇副教授和钟运琴助理研究员，以及华南理工大学的吕建明助理教授、西安交通大学的马小博博士。

感谢三位同行博士给予我计算机研究的重大启迪和持久帮助，他们是：康奈尔大学的江哲夫博士（在读）、UCSD的徐天音博士（在读）和George Mason大学的刘钥博士（已获得SUNY-Binghamton 大学的教职），分别代表了Linux 骨灰级黑客、系统研究典范和系统测量典范。在此一并感谢肯塔基大学的钱辰助理教授、香港科技大学的许斌（Pan Hui）助理教授、同济大学的张大强助理教授、东北大学的贾子熙助理教授、杜克大学的陈阳博士后、UIUC的李宏兴博士后、多伦多大学的黄威博士、UC Irvine的卜颖毅博士、约翰霍普金斯大学的展安东博士、香港理工大学的李韬博士、明尼苏达大学的金程博士和耶鲁大学的翟恩南博士等。

感谢北京航空航天大学的蒋竞助理教授，在我公派留学的一年里（那时她还没有从北大毕业）不辞辛劳帮助我办理了很多手续，包括代表我申请信息学院“学术十杰奖”。

感谢美国明尼苏达州的传教士Robert & Joanne Kraftson夫妇。他们长期和耐心地向我传播上帝的福音，虽然我至今未能完全接受基督教信仰，但却100%地感受到他们作为凡人传达给我的温暖与仁爱。在此一并感谢热情接待我的Mark & Jaycelyn Colestock夫妇和“友爱中华”组织（China Outreach Community）的领导人Glenn Kenadjian。

最后，感谢我的爱人夏洁媛陪伴我度过人生最贫困和漫长的四年，并为我生了一个可爱的宝贝，使我的生命能够部分延续、灵魂能够部分传承。在此一并感谢我的父母和岳父母。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　月　日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

按照学校要求提交学位论文的印刷本和电子版本；

学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务；

学校可以采用影印、缩印、数字化或其它复制手段保存论文；

在不以赢利为目的的前提下，学校可以公布论文的部分或全部内容。

（保密的论文在解密后应遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　月　日