软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(7):1980–1996 [doi: 10.13328/j.cnki.jos.005971]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向移动 Web 应用的浏览器缓存性能度量与优化*

马 郓[1], 刘譞哲[2], 梅 宏[2]

[1](清华大学 软件学院,北京 100084)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 马郓, E-mail: yunma@tsinghua.edu.cn

摘 要: 随着移动互联网的飞速发展,用户越来越多地通过移动设备访问 Web 应用.浏览器为 Web 应用提供基本的计算、渲染等运行时支撑,其缓存机制可以支持 Web 应用直接从本地而不是通过网络来获取可复用资源,不仅能够减少整体的执行时间从而提升应用加载速度,还能够减少网络流量使用和电池电量消耗,从而保证移动 Web 用户体验.近年来,围绕面向移动 Web 应用的浏览器缓存优化得到了国内外学术界和工业界的广泛关注.然而,现有研究工作大多都是从网络层面关注浏览器缓存的整体性能,未充分考虑移动互联网用户访问行为的差异性和动态性,以及Web应用自身持续演化对浏览器实际缓存性能的影响.针对这一问题,首先设计了一种新型主动式缓存度量实验,通过仿真用户的访问行为来分析移动 Web 应用实际资源使用情况,揭示了浏览器缓存的理论性能上限和实际性能之间的巨大差距,并发现了造成这一差距的 3 个主要原因:重复请求别名资源、启发式过期时间和保守的过期时间配置.基于此发现,从应用层和平台层分别提出了两种浏览器缓存性能优化方案,并实现了原型系统.实验结果表明,采用两种方法分别平均可减少 8%~51%和 4%~58%的网络流量,且系统开销较小.

关键词: 移动 Web 应用;浏览器缓存;性能度量;性能优化

中图法分类号: TP311

中文引用格式: 马郓,刘譞哲,梅宏.面向移动 Web 应用的浏览器缓存性能度量与优化.软件学报,2020,31(7):1980–1996. http://www.jos.org.cn/1000-9825/5971.htm

英文引用格式: Ma Y, Liu XZ, Mei H. Measurement and optimization of browser cache performance for mobile Web applications. Ruan Jian Xue Bao/Journal of Software, 2020,31(7):1980–1996 (in Chinese). http://www.jos.org.cn/1000-9825/5971.htm

Measurement and Optimization of Browser Cache Performance for Mobile Web Applications

MA Yun[1], LIU Xuan-Zhe[2], MEI Hong[2]

[1](School of Software, Tsinghua University, Beijing 100084, China)

[2](Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

Abstract: With the rapid development of the mobile Internet, users are increasingly accessing Web applications through mobile devices. Browsers provide the runtime support for Web applications, such as computation and rendering. The browser cache supports Web applications to obtain reusable resources directly from the local storage rather than downloading from the network. It not only improves the application loading speed, but also reduces network traffic usage and battery power consumption, ensuring the experience of mobile Web users. In recent years, attentions have been paid by both industry and academy on optimizing the browser cache performance of mobile Web applications. However, most of the existing research work focuses on the overall performance of the browser cache from the network level, and does not fully consider the impact of user access behaviors and application evolution on the performance of the browser cache. To address the issue, this study designs a proactive measurement experiment, which simulates the user access behavior and obtains the resources of the mobile Web applications. Experiment results reveal the huge gap between the ideal and actual performance of the browser cache, and dig out three main root causes to the gap: Resource aliases, heuristic caching strategies, and conservative cache time

* 基金项目: 中国博士后科学基金
Foundation item: China Postdoctoral Science Foundation
收稿时间: 2019-03-17; 修改时间: 2019-07-04; 采用时间: 2019-09-03

configuration. Based on the findings, in order to improve the browser cache performance of mobile Web applications, this study also proposes two optimization methods from the application layer and the platform layer, respectively, and implements the corresponding prototype systems. Evaluation results show that the two proposed methods can save the network traffic by 8%~51% and 4%~58% on average, respectively, and the system overhead is small.

Key words: mobile Web application; browser cache; performance measurement; performance optimization

自 20 世纪 90 年代初图灵奖获得者 Tim Berners-Lee 发明万维网(World Wide Web)以来,Web 应用已成为规模最大、范围最广、用户最多的互联网应用.近年来,随着移动互联网的发展以及移动设备(智能手机、平板电脑、可穿戴设备等)的普及,移动终端设备成为用户访问互联网的最主要入口.据 ComScore 统计,2014 年起,来自移动端的 Web 应用访问流量已经超过了 PC 端[1].相对于通过 PC 访问的传统 Web 应用,移动 Web 应用表现出一些新的特征.一方面,移动设备计算能力和电池续航能力均相对有限,用户的移动性使其所处的网络环境动态多变(如在 Wi-Fi 和 Cellular 之间频繁切换),因此移动 Web 应用的用户体验更加综合[2]:不仅关注 Web 应用的访问性能(主要表现为应用页面的加载时间),同时应用的网络流量使用和电池电量消耗也是用户非常关心的要素.另一方面,原生移动应用(Native App)和应用商店生态的发展,也使得移动 Web 应用呈现出“App 化”的使用模式[3],即,用户往往在相对较短的时间间隔内频繁重复访问少数特定 Web 应用.因此,如何保障综合的用户体验,成为移动 Web 应用所关注的重要研究问题.

作为 Web 应用的运行容器,浏览器为 Web 应用的运行提供计算、渲染等基本服务,其缓存机制是保证多次重复访问同一 Web 应用时优化用户体验的重要技术.浏览器缓存的基本功能是把 Web 应用的文本、图片等资源保存在本地存储空间中,当用户再次访问同一应用时,浏览器可以直接地从本地获取资源,从而减少网络请求和传输流量.由于 Web 应用加载的绝大部分时间消耗在从服务器获取资源的过程上[4],所以网络传输的减少也可以提升应用加载速度.最后,资源加载过程中的网络请求是移动 Web 应用电量消耗的主要来源[5],因此更多地从本地获取资源还可以减少电量的消耗,进而增加移动设备的待机时间.

浏览器缓存一直以来是 Web 系统的研究重点,学术界和产业界针对移动 Web 应用的浏览器缓存也开展了不少研究工作,取得了一些进展和成果.然而,现有工作大多基于插桩的客户端设备或者互联网服务供应商获取的网络数据,从网络层面关注浏览器缓存的整体性能,并未充分考虑端到端性能,即特定用户访问特定应用时的浏览器缓存性能.端到端性能主要取决于用户访问行为和应用自身演化两个因素,Web 应用在访问时即时更新的特点,使得用户以不同频率访问 Web 应用时的缓存性能会有明显不同.而现有数据集受限于偏倚(biased)或片面的用户访问行为以及仅仅包含特定时刻的 Web 应用快照,无法适用于端到端性能的研究.

针对上述问题,本文首先设计了一种新型主动式缓存度量实验,研究用户频繁访问特定移动 Web 应用时浏览器缓存的性能,以发现造成缓存性能低效的原因;然后基于实验发现,从应用层和平台层分别提出了两种浏览器缓存性能优化方法并实现了原型系统;最后在流行 Web 应用上评估方法的有效性.本文的主要贡献包括:

- 设计了主动式、规模化的移动 Web 应用浏览器缓存度量实验,揭示了浏览器缓存的理论性能上限和实际性能之间的巨大差距,并发现了造成这一差距的 3 个主要原因:重复请求别名资源、启发式过期时间和保守的过期时间配置.
- 提出了两种浏览器缓存优化方法并实现了原型系统.一种是在应用层基于资源包的缓存优化方法,将 Web 应用相对稳定的资源配置到资源包中,利用 HTML5 的应用缓存机制实现从资源包加载资源.一种是在平台层基于云-端融合的缓存优化方法,通过云端预先加载 Web 应用获取资源状态,然后与终端的缓存资源进行同步,实现缓存的精确控制.
- 在流行 Web 应用上评估了方法的有效性.结果表明,基于资源包的缓存优化方法平均可以减少 8%~51%的网络流量,基于云-端融合的缓存优化方法平均可以减少 4%~58%的网络流量,并且系统的额外开销较小.

本文第 1 节介绍移动 Web 应用浏览器缓存性能度量实验.第 2 节介绍基于资源包的缓存优化方法.第 3 节介绍基于云-端融合的缓存优化方法.第 4 节介绍方法效果评估.第 5 节将本文工作与相关工作进行比较.第 6 节总结全文并展望未来的工作.

1 移动 Web 应用的浏览器缓存性能度量实验

本文提出一种主动式方法来收集移动 Web 应用的资源更新历史,进而可以针对不同用户访问间隔,研究浏览器缓存的性能.本节首先介绍数据收集方法,然后定义度量指标并建立浏览器缓存性能的度量模型,进而分析浏览器缓存的理想性能和实际性能之间的差距,最后剖析造成这一差距的原因.

1.1 数据收集

为了分析用户访问行为和应用自身演化对浏览器缓存的影响,本文设计了主动式度量方法,具体流程为:在较长一段时间内、以较短的时间间隔、周期性地持续不断地用浏览器访问 Web 应用.每次访问前,先清空浏览器缓存,然后记录访问时请求和获取的所有 Web 资源.较短的时间间隔不仅可以获取 Web 应用尽可能完整的资源更新历史,而且能够仿真以不同时间间隔访问 Web 应用的用户行为.较长的数据收集时间可以保证有足够多的数据以消除分布偏差.

为了进行对比分析,本文从 Alexa 网站排名[6]中选取了两组移动 Web 应用.第 1 组(Top 组)包含了 Alexa 排名前 100 的移动 Web 应用,它们的缓存配置在预期上是被经过精心设计和优化的,所以其缓存性能可以被认为是目前的最优水平.另一组(Random 组)包含了从 Alexa 排名前 1 000 000 的移动 Web 应用中随机选取的 100 个应用,本文假定它们的缓存性能代表目前的平均水平.进一步地,本文过滤掉无法访问的应用以及完全通过 HTTPS 协议访问的 Web 应用;另外,同样的 Web 应用但域名不同的,仅保留一个主域名的应用,如 google.com、google.in、google.de、google.jp 等仅保留 google.com.根据此原则过滤之后,数据集中剩余 146 个 Web 应用,其中 Top 组 55 个、Random 组 91 个.

对于数据集中的每一个 Web 应用,我们在一周时间内、每隔 30 分钟访问一次应用的首页,记录访问时获得的所有 Web 资源.本文认为 30 分钟的时间间隔能够涵盖 Web 应用的资源变化情况.在一周时间内,每个 Web 应用都被访问了超过 300 次,总共收集到 157GB 的数据:Top 组 73GB,Random 组 84GB.

1.2 度量指标

当用户第 1 次访问某个 Web 应用时,浏览器缓存为空,Web 应用的所有资源都需要从网络下载到本地.给定再访问间隔(revisting interval,简称 RI)Δt,同一个 Web 应用被第 2 次访问时,该应用的部分资源即可从缓存中加载,而另外一些资源仍需要从网络下载.下面考察同一个 Web 应用在第 2 次访问时的浏览器缓存行为.

定义 $S_t=\{r_t\}$ 为 t 时刻浏览器在缓存为空时所获取到的资源集合,其中,$r_t\in\langle URL,Content\rangle$ 是一个具体的资源,URL(包括域名、端口、路径、参数)作为标识来区分不同的资源,Content 是资源的具体内容.一个资源随着时间的变化可以被更新,因而 S_t 代表了一个 Web 应用在 t 时刻的最新资源情况.

定义 $CA_t(\Delta t)=\{r_t\in S_t\mid \exists r_{t-\Delta t}\in S_{t-\Delta t}, r_t.Content=r_{t-\Delta t}.Content\}$ 为一个 Web 应用在 t 时刻的所有资源中已经在 Δt 时间之前被下载的资源集合.需要注意的是,该定义并没有根据 URL 来判断一个资源是否已经被下载,而是根据资源的内容是否一致.实际上,$CA_t(\Delta t)$是 Web 应用被再次访问时可以直接从缓存中获取的资源集合.

基于 S_t 和 $CA_t(\Delta t)$,定义 Web 应用经过Δt 时间后的可缓存率$\Psi(\Delta t)$为 $CA_t(\Delta t)$在 S_t 中所占的比例,即

$$\Psi(\Delta t)=\frac{CA_t(\Delta t)}{S_t},$$

$\Psi(\Delta t)$度量了一个 Web 应用在Δt 时间后可从缓存中加载资源的比例.$\Psi(\Delta t)$越大,表明越多的资源是可被缓存的.

定义 $C_t(\Delta t)=\{r_{t-\Delta t}\in S_{t-\Delta t}\mid r_{t-\Delta t}\ is\ configured\ to\ be\ cached\}$ 为在 t 时刻浏览器缓存中的资源集合.$C_t(\Delta t)$代表了在 t–Δt 时刻被下载、进而存储在缓存中的资源. $C_t(\Delta t)\subseteq S_{t-\Delta t}$, 因为并不是所有 $S_{t-\Delta t}$ 中的资源都配置为可被浏览器缓存的.

根据浏览器缓存机制,可以将 $C_t(\Delta t)$进一步分解为 EX_t 和 FR_t,即 $C_t(\Delta t)=EX_t\cup FR_t$. 其中,$EX_t$ 表示在 t 时刻已经过期的资源,这些资源会被浏览器判断为可能发生修改,所以无法从缓存中获取;FR_t 表示在 t 时刻没有过期的资源,这些资源可以从缓存中获取.当一个 Web 应用被访问时,对于每一个资源请求,如果浏览器在 FR_t 集合中找到所请求的 URL,则该资源将会被从缓存中获取,本文称其为缓存命中(hit).定义为 $H_t=\{r_t\in S_t\mid \exists r\in FR_t, r_t.$

$URL=r.URL\}$ Web 应用在 t 时刻实际从缓存中获取的资源集合.另一方面,不论所请求的资源 URL 存在于 EX_t 集合中,或者在整个 $C_t(\Delta t)$集合中都不存在,浏览器都会从网络下载该资源,本文称这种情况为缓存失效(miss).定义 $M_t=\{r_t\in S_t\mid(\exists r\in EX_t,r_t.URL=r.URL)\vee(\forall r\in C_t(\Delta t),r_t.URL\neq r.URL)\}$ 为 Web 应用在 t 时刻从网络获取的资源集合.

由于 Web 资源的缓存策略可能没有被有效配置,因此在缓存中过期的资源可能实际上并没有发生变化,在缓存中没有过期的资源可能实际上已经发生变化了.此外,由于同一资源的 URL 在多次访问时会发生变化,所以仅仅通过 URL 来判断是否存在缓存资源可能会发生错误.因此,本文将 H_t 集合和 M_t 集合进一步分解为正确和错误两种情况,给出如下 4 组定义.

- 正命中(positive hit,$PH_t(\Delta t)$):给定 $r_t\in H_t,r_t\in PH_t(\Delta t)\Leftrightarrow\exists r_{t-\Delta t}\in S_{t-\Delta t},r_t.URL=r_{t-\Delta t}.URL\wedge r_t.Content=r_{t-\Delta t}.Content$.$PH_t(\Delta t)$ 中的资源是从缓存中获取的,而且资源状态和服务器上的最新版本一致.$PH_t(\Delta t)$中的资源越多表示越多的资源被正确地从缓存中获取.
- 误命中(negative hit,$NH_t(\Delta t)$):给定 $r_t\in H_t,r_t\in NH_t(\Delta t)\Leftrightarrow\exists r_{t-\Delta t}\in S_{t-\Delta t},r_t.URL=r_{t-\Delta t}.URL\wedge r_t.Content\neq r_{t-\Delta t}.Content$.$NH_t(\Delta t)$ 中的资源是从缓存中获取的,但是资源在服务器已经被更新,此时浏览器应该从服务器上获取最新的资源内容.$NH_t(\Delta t)$中的资源越多,表示越多的过期资源被用于渲染 Web 应用,可能会导致 Web 应用的功能出现问题.
- 正失效(postive miss,$PM_t(\Delta t)$):给定 $r_t\in M_t,r_t\in PM_t(\Delta t)\Leftrightarrow\forall r_{t-\Delta t}\in S_{t-\Delta t},r_t.Content\neq r_{t-\Delta t}.Content.PM_t(\Delta t)$ 中的资源是从网络获取的,而且资源的内容并没有出现在 t–Δt 时刻的资源集合 $S_{t-\Delta t}$ 中,所以此时从网络获取资源是正确的行为.$PM_t(\Delta t)$中的资源越多,表明 Web 应用有越多新的或变化的资源.
- 误失效(negative miss,$NM_t(\Delta t)$):给定 $r_t\in M_t,r_t\in NM_t(\Delta t)\Leftrightarrow\exists r_{t-\Delta t}\in S_{t-\Delta t},r_t.Content=r_{t-\Delta t}.Content.NM_t(\Delta t)$ 中的资源是从网络获取的,但是资源的内容曾经在 t–Δt 时刻下载到本地,所以此时从网络获取资源是错误的行为.$NM_t(\Delta t)$中的资源越多,表明越多的网络传输是冗余的.

根据上述定义,一个 Web 应用经过 Δt 时间后的实际缓存率 $\Phi(\Delta t)$可以定义为正确地从缓存中获取的资源 $PH_t(\Delta t)$在所有可以从缓存中获取的资源 $PH_t(\Delta t)\cup NM_t(\Delta t)$ 中的比例,即

$$\Phi(\Delta t)=\frac{PH_t(\Delta t)}{PH_t(\Delta t)\cup NM_t(\Delta t)}.$$

本文对于每一指标选取了资源大小和资源个数作为具体的数值,分别用下标 s 和 n 来表示.例如,Ψ_s 和Ψ_n 分别表示采用资源大小和资源个数计算的 Web 应用可缓存率.

1.3 基于决策树的缓存行为模型

为了计算上述指标,本文通过关联资源的缓存配置与实际更新情况,根据 HTTP 协议规范[7],建立了基于决策树的缓存行为模型,如图 1 所示.图中展示了每种缓存配置情况下(用圆角矩形表示)资源内容变化情况(实线表示发生变化,虚线表示未发生变化)与缓存的 4 种行为(用 4 种图形填充表示)之间的关系.

对每一个资源请求,浏览器通过 URL 在缓存中查找相应的资源.如果该 URL 在缓存中能够找到(URLHit),那么浏览器缓存开始工作.首先检查资源的缓存时间以判断该资源是否过期.HTTP 协议规定了两种过期策略:一种是由服务器显式配置的过期时间,另一种是由浏览器赋予的启发式过期时间.对于启发式过期时间策略(Heuristic),由于无法明确在给定时刻下资源是否被判断为过期,所以无论资源内容是否发生变化,4 种缓存行为都可能出现.在最坏情况下,可以假定启发式策略下的资源变化(ContentChanged)状态为误命中,资源未变化(ContentNotChanged)状态为误失效.若无特殊说明,下文的分析和讨论都针对最差情况.对于服务器指定过期时间策略(ServerSpecified),可以分为 3 种情况:不存储(NoStore),表示资源不能在磁盘上被永久存储;不缓存(NoCache),表示资源虽然不能被浏览器缓存,但却可以在磁盘上永久存储;明确的过期时间(TimeCache),表示资源在过期时间之前可以从缓存获取.对于不存储和不缓存,资源变化状态的缓存行为都是正失效,而资源未变化状态的缓存行为都是误失效.对于明确的过期时间,又可针对缓存中的资源是否过期进一步加以分类:如果资源

被判定为过期(Expired),那么根据缓存的验证机制,通过判断是否配置最近更新时间(last modified time)或验证标志(Etag)又可分为验证(Validated)和未验证(NotValidated),这两种情况下资源变化状态的缓存行为都是正失效,而资源未变化状态的缓存行为都是误失效;如果资源被判定为未过期(Fresh),若资源没有发生变化,则缓存行为是正命中,若资源发生变化,则缓存行为是误命中.

如果 URL 没有在缓存中找到(URLMiss),浏览器会从网络获取资源.但是这种情况并不意味着资源无法从缓存中获取.根据前述讨论,URL 不同的资源,其内容可能是一样的(ContentFound),所以此时的缓存行为是误失效.最后一种状态是新资源(New),表示该资源是全新的,URL 和内容在缓存中都无法找到,这种资源需要从服务器下载,因而其缓存行为是正失效.

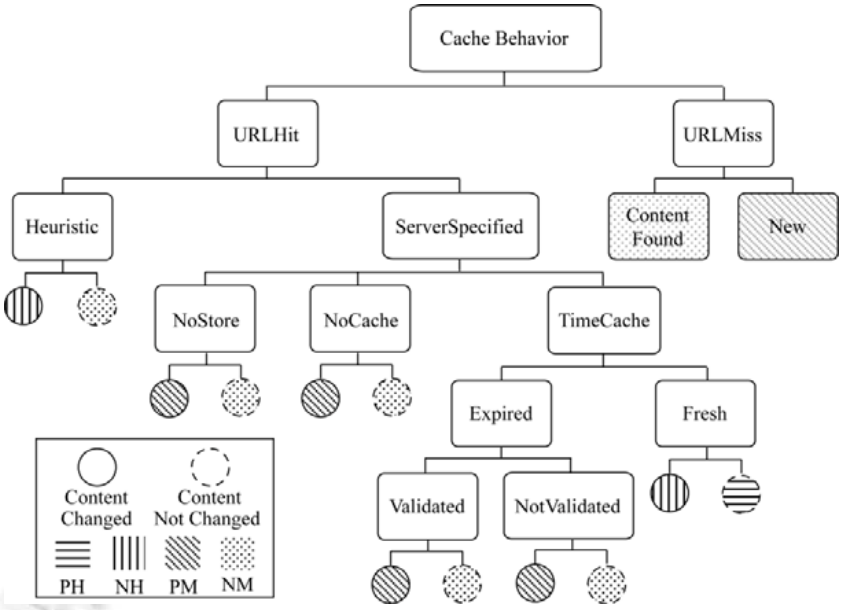


Fig.1 Cache behavior taxonomy

图 1 缓存行为模型

1.4 实验结果

1.4.1 缓存的理想性能

为了分析 Web 应用的资源有多大比例能够被浏览器缓存,本文对 Top 组和 Random 组中的每个 Web 应用计算可缓存率Ψ,表 1 给出了在 24 小时、6 小时、0.5 小时的再访问间隔下 Web 应用总体资源和各类别资源可缓存率Ψ的中位数,Ψ_s 和 Ψ_n 分别表示采用资源大小和资源个数计算的 Web 应用可缓存率.

总体而言,1 天之内再次访问 Web 应用时大多数资源可以从缓存中获取,所以缓存对于提升 Web 应用的性能是有帮助的.对于 Top 组的 Web 应用,从大小来看,56.4%~72.7%的资源是可被缓存的;从个数来看,69.5%~83.2%的资源是可被缓存的;Random 组的 Web 应用可缓存率更高,从大小来看,63.2%~79.0%的资源可被缓存,从个数来看,72.1%~84.5%的资源可被缓存.随着再访问间隔的缩短,可缓存率 Ψ 会有所增加,原因是时间间隔越短,Web 应用资源发生变化的可能性就越小,所以资源越有可能在缓存中命中.

缓存对于 Random 组的 Web 应用比 Top 组的更加有效.对于每个再访问间隔,Random 组的可缓存率 Ψ_s 和 Ψ_n 比 Top 组的都要大.例如,对于 24 小时的再访问间隔,Random 组整体资源可缓存率 Ψ_s 比 Top 组的高 6.8%,表明 Random 组 Web 应用的资源比 Top 组的更有可能被缓存.该现象的原因是普通 Web 应用相比于流行 Web 应用更不容易发生变化.

对于不同类型的资源,HTML 资源的可缓存率最低,所以从缓存中获益最少,其原因是 HTML 资源大多是通过后台服务器动态生成的,广告、推荐信息等动态内容会导致每次访问的 HTML 都有差异.CSS 资源的可缓存

率最高,原因可以从 CSS 的功能方面进行解释:虽然 Web 应用的内容可能经常变化,但是应用的界面布局和风格样式却是比较稳定的,因而 CSS 资源很少发生变化.Top 组 Web 应用的 JavaScript 资源比 Random 组的更容易被缓存.Top 组 JavaScript 资源可缓存率随再访问间隔的变化不大(70.9%~72.3%),而 Random 组随再访问间隔的变化有较大的变动(54.1%~72.1%).由于 JavaScript 资源主要负责实现 Web 应用的功能,因此这一现象表明,Top 组 Web 应用在功能上比 Random 组更加稳定.图片资源主要在短的再访问间隔下可缓存率较高.随着再访问间隔的增加,图片资源的可缓存率Ψ有明显的下降(Top 组降低 30%,Random 组降低 20%).其原因是在较长一段时间过后,新的图片资源会取代旧的图片资源.

Table 1 Cacheablity ratio Ψ of aggregated resources and resource types

表 1 Web 应用总体和各类型资源的可缓存率Ψ

	24h		6h		0.5h	
	Ψ_s(%)	Ψ_n(%)	Ψ_s(%)	Ψ_n(%)	Ψ_s(%)	Ψ_n(%)
All_{Top}	56.4	69.5	64.7	76.0	72.7	83.2
All_{Random}	63.2	72.1	73.5	78.8	79.0	84.5
$HTML_{Top}$	4.5	34.2	7.2	39.6	9.1	41.1
$HTML_{Random}$	22.4	44.0	27.8	49.3	32.7	51.0
CSS_{Top}	74.6	86.9	79.5	86.9	82.4	89.4
CSS_{Random}	72.0	90.3	81.7	89.8	84.8	91.4
JS_{Top}	70.9	84.5	69.7	82.4	72.3	83.4
JS_{Random}	54.1	69.8	66.6	77.4	72.1	80.6
$Image_{Top}$	51.0	67.2	68.3	78.3	81.4	88.8
$Image_{Random}$	74.0	76.0	83.7	82.8	91.4	89.7

1.4.2 缓存的实际性能

为了分析 Web 应用的可缓存资源有多大比例实际可以从缓存中获取,本文对 Top 组和 Random 组中的每个 Web 应用计算实际缓存率Φ,表 2 给出了在 24 小时、6 小时、0.5 小时的再访问间隔下 Web 应用总体资源和各类别资源实际缓存率Φ的中位数,Φ_s 和 Φ_n 分别表示采用资源大小和资源个数计算的 Web 应用实际缓存率.

Table 2 Actual cache ratio Φ of aggregated resources and resource types

表 2 Web 应用总体和各类型资源的实际缓存率Φ

	24h		6h		0.5h	
	Φ_s(%)	Φ_n(%)	Φ_s(%)	Φ_n(%)	Φ_s(%)	Φ_n(%)
All_{Top}	80.7	48.8	85.2	56.6	88.5	64.9
All_{Random}	42.4	32.7	48.9	39.6	53.8	44.4
$HTML_{Top}$	2.8	3.8	2.1	4.9	16.4	12.9
$HTML_{Random}$	39.4	11.2	40.9	13.0	47.2	19.2
CSS_{Top}	82.6	52.7	87.3	55.9	88.1	67.7
CSS_{Random}	44.6	23.5	55.2	42.3	57.9	45.7
JS_{Top}	74.9	37.8	76.9	44.5	83.7	52.4
JS_{Random}	51.6	29.4	63.2	38.9	69.2	48.8
$Image_{Top}$	84.5	55.7	90.8	63.3	93.0	70.6
$Image_{Random}$	36.6	35.4	41.9	42.3	45.2	45.4

Random 组 Web 应用的实际缓存率比 Top 组的都要低,因而 Random 组并没有很好地利用缓存.从资源大小来看,Top 组应用在所有给定再访问间隔的情况下都有超过 80%的可缓存资源实际可以从缓存中获取,而 Random 组应用只有再访问间隔为 0.5 小时的最好情况下才有大约 50%的可缓存资源实际可以从缓存中获取,小资源的缓存没有被很好利用,实际缓存率较低.虽然从资源大小来看 Top 组 Web 应用的实际缓存率Φ_s较高,但从资源个数来衡量的实际缓存率Φ_n却相对较低.例如,对于 24 小时的再访问间隔,Top 组Φ_s都超过 80%,但其Φ_n却没有超过 50%.该现象表明,Top 组 Web 应用更多关注了大资源的缓存,而忽略了小资源的缓存.从资源大小来看,小资源所占总资源的比例并不是很高,但从数量来看,小资源却占总资源的绝大多数.在现有浏览器的工作机制下,小资源的实际缓存率低会导致网络连接的大量浪费.

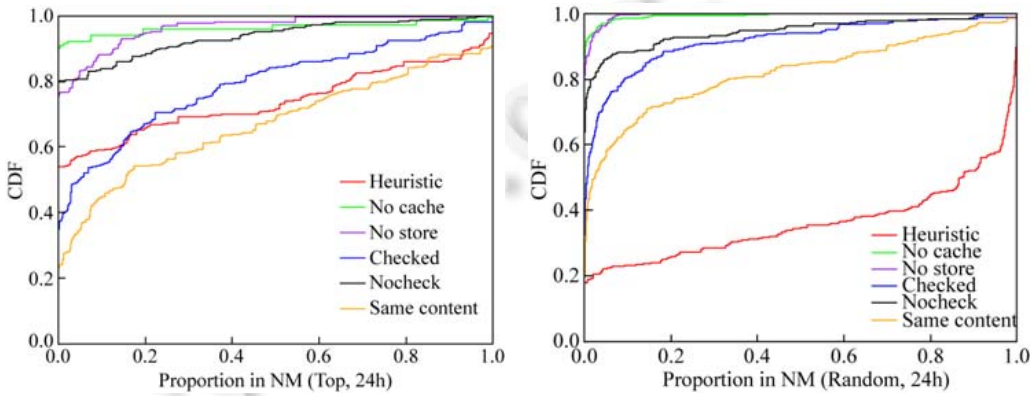
对于不同类型的资源,HTML 资源的可缓存性几乎没有被利用.Top 组 Web 应用 HTML 资源的实际缓存率Φ_s和Φ_n都比 Random 组的要低.在 24 小时再访问间隔的最差情况下,Top 组 HTML 资源的Φ_s仅为 2.8%.Random 组 HTML 资源在 0.5 小时再访问间隔的最好情况下Φ_s也没有超过 50%.因为无法预测 HTML 资源发生变化的

时间,Web 应用很难对 HTML 资源设置缓存.因此,绝大多数 Web 应用放弃了 HTML 资源的缓存.对于 Top 组 Web 应用,实际缓存率最高的是图片(84%~93%);而对于 Random 组 Web 应用,实际缓存率最高的是 JavaScript (51%~70%).Random 组 Web 应用的图片资源实际缓存率较低,Φ_s和Φ_n在最好情况下都没有超过 50%.

综上所述,移动 Web 应用浏览器缓存的理想性能和实际性能之间存在巨大差距.对于 Top 组 Web 应用,这种差距在小资源上更加显著.对于 Random 组 Web 应用,所有类型资源的可缓存性都没有被很好利用.

1.5 原因分析

误失效是浏览器缓存的理想性能和实际性能之间差距的主要来源,即可以从缓存获取的资源却通过网络重新下载.根据图 1 的缓存行为模型,导致资源误失效的缓存行为一共有 6 种:启发式过期时间(Heuristic)、不缓存(NoCache)、不存储(NoStore)、过期后验证(Checked)、过期后未验证(NoCheck)、URL 不同但内容相同的别名资源(Same Content).通过将这 6 种情况进一步加以分解,可以计算每种情况在总的误失效行为中所占比例,图 2 展示了 Top 组和 Random 组在 24 小时再访问间隔下每种情况的累积概率分布(CDF).



(a) Top 组 (b) Random 组

Fig.2 The distribution of breakdown of negative miss

图 2 Web 应用整体误失效行为分解的分布

从图中可以发现,别名资源(Same Content)、启发式过期时间(Heuristic)和保守的过期时间配置(Checked, NoCheck)是导致误失效行为的 3 个最主要原因.

别名资源对 Top 组和 Random 组 Web 应用的缓存性能都有显著影响.这种情况发生的原因是新请求资源的内容曾经被缓存过,但是 URL 却在缓存中找不到.根据 HTTP 协议规范,浏览器以 URL 作为资源标识,不同 URL 被认为是完全不同的资源,所以在查找缓存时也是以 URL 为索引的.但在 Web 应用的实际开发实践中,由于资源版本控制和服务器端资源层级管理架构的需要,相同内容的资源常常会被开发者赋予不同的 URL,导致别名资源的产生,造成缓存失效.

Web 开发者采用启发式过期时间通常是由于没有意识到浏览器缓存的重要性,所以没有为资源配置显式的过期时间.启发式过期时间对于 Random 组的 Web 应用是非常严重的问题,因为大多数 Web 开发者并没有配置缓存的意识,加之在 PC 环境下浏览器缓存的效果有限,所以常被忽略.然而,在 Top 组的 Web 应用中,启发式过期时间也是导致误失效行为的第二大因素.

由于开发者难以估计资源的更新时间,所以资源通常被配置较短的过期时间.本文发现 Top 组和 Random 组分别有 40%和 24%的资源被配置了小于 1 天的过期时间.该结果表明,流行 Web 应用在过期时间的配置上更加保守,因为它们的资源变化更为频繁.然而,通过进一步分析每个资源的实际更新周期,本文发现在缓存时间小于 1 天的资源中,Top 组和 Random 组分别有 89%和 86%的资源在整个 1 周的时间内都没有发生任何变化.因此,这些资源的过期时间可以配置得更久.

综上,浏览器缓存需综合考虑用户访问行为以及移动 Web 应用资源的演化情况来维护本地资源,从而提高缓存命中率,保障移动用户体验.

1.6 影响分析结果的因素

本文选取 Alexa 排名前 100 中的 55 个 Web 应用和排名前 1 000 000 中的 91 个 Web 应用作为研究对象,Web 应用的总数较少,可能不足以代表流行 Web 应用和普通 Web 应用的一般情况,对结论的普适性造成一定影响.

一个 Web 应用通常会由许多页面构成,但本文仅研究了每个 Web 应用的首页,没有考虑其他页面.考虑到首页是用户访问一个 Web 应用时最先浏览的页面,因而 Web 开发者会采取诸多措施来优化首页的浏览体验;同时,由于 Web 应用的整体风格会与首页一致,所以首页的不少资源在其他页面中也会出现,特别是布局、功能、装饰性图片等等.因而,首页的缓存性能可以在一定程度上代表该应用的整体水平.

本文没有分析通过 HTTPS 协议来访问的 Web 应用,可能会忽略一些重要发现.但是在过滤仅通过 HTTPS 访问的应用时,仅过滤掉 4 个 Top 组应用(Facebook、Twitter、Linkedin、Netflix)和 3 个 Random 组应用(Delta、Smartrecruiters、Polarpersonaltrainer),所以,现有数据结果不会受到 HTTPS 协议的影响.此外,HTTPS 的很多行为与 HTTP 是一致的,因此本文的结论可以很大程度上推广到 HTTPS 的 Web 应用.

2 基于资源包的缓存优化方法

为了优化移动 Web 应用的浏览器缓存性能,一种思路是在应用层面借鉴原生应用的本地资源管理方法.原生应用通过安装包来管理其本地资源,安装包中包含了运行原生应用所需的基本代码和数据.用户使用前需要下载安装包;安装后,代码和数据就都存储在本地空间中.原生应用运行时,通过安装包存储在本地的资源一律从终端获取,无需与服务器确认状态.为了支持应用的升级和更新,开发者会实现更新检查逻辑:在应用启动时向服务器发送请求,判断本地当前版本是否为最新版;当发现有更新时,提示用户下载并安装新的安装包,从而统一更新本地存储空间中的代码和数据.因此,原生应用通过安装包实现了对本地资源的精确管理.

从上述思想出发,本节提出一种基于资源包的移动 Web 应用本地资源管理机制,优化浏览器缓存性能.该机制结合了原生应用基于安装包的管理方法,并且保持了 Web 应用即时更新的特性.具体而言,移动 Web 应用可以将较为稳定不变的资源封装在资源包中;在运行时,凡是在资源包中的资源一律从本地获取,而不是从网络下载;不在资源包中的资源则通过浏览器的默认机制加载.同时,Web 应用增加资源包更新检查逻辑,被访问时可自动检查资源包是否发生变化;当资源包发生变化需要被更新时,其内部的资源按照浏览器的默认机制进行更新.

2.1 资源包的生成与维护

由于资源包中的资源一律从本地获取,所以为了保持资源状态的一致性,资源包中任何资源的更新都会导致整个资源包的刷新.因而,我们需要将长期不变且更新时间比较一致的资源配置在资源包中,使资源包能够在一段时间内保持稳定且更新时带来的额外开销最小.本文通过 4 个步骤为 Web 应用生成和维护资源包.

获取 Web 应用的资源更新.按照固定的时间间隔,获取每个时间点下 Web 应用的全部资源信息,包括 URL 地址、内容的哈希值、实际大小、缓存时间(如果为启发式过期时间,则设置为 1 小时),存储到资源库中.

资源别名归一化.在资源库中识别别名资源,并将它们归一化成一个抽象资源,然后通过资源映射文件来记录抽象资源与具体资源之间的 URL 对应关系.资源包使用抽象资源的 URL 模板来标识资源,因此,URL 不同但内容相同的别名资源也可以在资源包中命中,避免被重复下载.通过观察别名资源的 URL 特征,本文总结出两种类型的别名.第 1 种是出现在 URL 参数部分的随机串,这些随机串由客户端 JavaScript 脚本(如 Math.random())或服务器端脚本生成,例如 d.jpg?892 和 d.jpg?157 实际上是 d.jpg 图片资源的两个别名.在这种情况下,URL 只在参数部分有所区别.另一种情况的别名出现在 URL 前缀上.在不同时刻访问移动 Web 应用时,资源可能会通过不同的 CDN 服务器获取.在这种情况下,URL 的路径和参数部分是相同的,只是域名部分会由于目标 CDN 服务器的不同而有所差异,如 a.cdn.com 和 b.cdn.com.根据两种别名类型的特征,我们采用最长公共子串算法找出别名之间的公共部分,然后用正则表达式来代替变化的部分.例如,图片 d.jpg 的两个别名 d.jpg?892 和 d.jpg?157

可以用正则表达式“d.jpg\?*”来表示.

资源更新时间预测.为了选择较为稳定的资源封装到资源包中,我们需要知道每个资源的可能更新时间.本文假设资源的变化更新历史能够反映该资源未来的更新趋势.例如,如果一个资源每天更新 1 次,那么在第 2 天它很有可能发生更新.因此,本文建立了回归模型来预测资源更新时间.模型的输入为资源的历史更新情况,通过更新历史中发生变化的时刻来预测下一次变化出现的时刻.值得注意的是,有时候资源更新会有个别异常点,例如一个资源可能在历史上每天更新 1 次,而突然出现 1 小时后就更新的情况.此时,不宜激进地缩短更新时间的预测,应该根据其后续的更新情况来判断.因此,我们使用了基于梯度下降法的线性回归模型[8].

基于用户访问分布的资源包生成.根据资源更新时间的预测结果,选取最佳的资源集合,生成资源包或更新现有资源包.我们采用一种基于收益的算法来生成资源包.假设放入资源包中的资源可以为 Web 应用的用户减少网络流量,不同的资源组合为用户减少的流量是不同的,所以最优的资源包应该是节省网络流量最大的资源组合.由于用户的访问频率并不相同,所以资源包为不同用户节省的流量也会有很大差异.为了能够度量资源包的收益,定义一个用户访问分布函数σ来表示 Web 应用不同再访问间隔的用户比例;进而可以计算平均节省流量来定量表示资源包的收益.由于一个资源集合的总体更新时间取决于集合中更新最频繁的资源,我们对资源集合按更新时间从短到长进行枚举.给定一个更新时间后,将一个资源 H_j 放入资源包中能够节约的传输流量可以表示为

$$benefit(H_j)=(H_j.predictedTime-H_j.cacheDuration)\times H_j.size,$$

其含义为:一个资源通过放入资源包所节省的流量,是由于该资源放入资源包后所预期达到的缓存时间与之前设置的缓存时间之差造成的.上式乘以用户访问分布就是总体上所能节省的流量.因此,对于给定的更新时间 T_i,

$$benefit(M_i)=\sum_j \sigma(H_j.cacheDuration,T_i)\times H_j.size,$$

其中,$\sigma(i,j)$表示再访问间隔在 i 和 j 之间的用户比例之和.由此可以枚举计算所有可能组合的收益.最后选择收益最大的组合,即所有 $benefit(M_i)$中的最大值,作为候选资源包.为了降低资源包的更新频率,只有当新生成资源包的收益值超过原有资源包一定阈值之后,才会用新资源包替换原有的资源包.

需要说明的是,如果 Web 应用的资源在两次获取更新的间隔中发生了变化而用户恰好在这期间访问了 Web 应用,那么此时的资源包状态就与 Web 应用的最新状态不一致.一种解决方案是 ReWAP 提供某种通知机制,当 Web 应用发生变化时由 Web 服务器主动通知 ReWAP 更新资源状态并重新生成资源包,这样可以保证资源包的状态一直与原应用状态保持一致.

2.2 基于资源包的资源加载

为了在浏览器的现有缓存机制之上使移动 Web 应用具备基于资源包的资源管理能力,本文设计了一个客户端运行容器,图 3 展示了该运行容器的组成部分和运行流程.资源包运行容器拥有一个应用特定的存储空间用于存储资源包的资源.这个空间是每个 Web 应用所独有的,不与其他应用共享.

当用户访问 Web 应用时,资源包运行容器被首先加载到浏览器中,然后通过 3 个步骤来加载目标 Web 应用.

检查资源包的更新状态.当资源包运行容器加载完毕后,首先检查资源包是否发生更新.如果没有更新,则运行容器直接开始加载目标 Web 应用.如果发生更新,或第 1 次访问 Web 应用,那么运行容器首先需要更新应用特定存储空间中的资源.只有当资源包中的资源都更新完毕后,再开始加载目标 Web 应用.这样就可以保证 Web 应用在加载过程中从本地资源包中获取的资源都是最新的.

更新本地资源包.当资源包发生变化、资源包中的资源需要更新时,运行容器会采用标准的 Web 资源加载机制来更新应用特定存储空间中的每一个资源.具体而言,对于一个资源,运行容器首先检查其缓存是否过期,如果没有过期,则该资源无需更新;否则,运行容器向原服务器发送验证请求以检查该资源是否真正过期.只有当资源真正发生变化时,资源的实际内容才会被从网络下载到本地,保存到应用特定的存储空间中.

加载资源.当资源包完成更新后,运行容器开始加载目标 Web 应用.在加载过程中,运行容器拦截应用发出的所有资源请求.对于每个资源请求,运行容器通过正则表达式匹配来比较请求 URL 和资源包中的 URL 模板.

如果找到匹配的模板,则表明该资源可以从资源包中加载,可通过资源映射文件找到对应具体资源的 URL,进而从应用特定的存储空间中获取该资源.如果没有找到匹配的 URL 模板,则该请求会被发送到浏览器内核中,通过浏览器的标准方式从缓存或者网络获取.

需要注意的是,资源包和资源映射文件本身也保存在应用特定的存储空间中,这两个文件采用浏览器现有的缓存机制来获取.

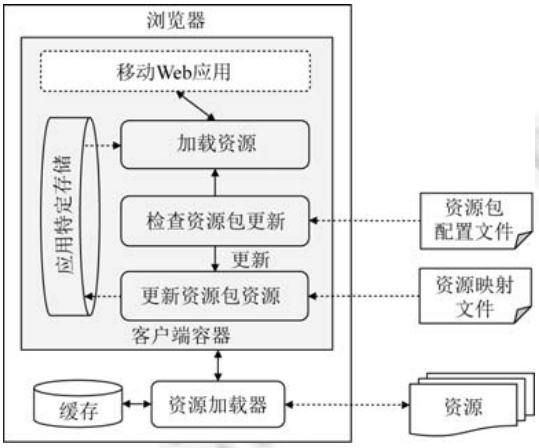


Fig.3 The structure of the runtime container supporting the resource packages

图 3 支持资源包的运行容器

2.3 系统实现

本文基于 HTML 5 的 Application Cache(下文简写作 AppCache)技术[9]实现了支持资源包管理机制的 ReWAP 系统.AppCache 是 HTML 5 针对 Web 应用的离线访问而提供的新接口,开发者可以通过 manifest 文件,声明哪些资源可以被离线使用.manifest 文件主要包含两个部分:CACHE 部分显式列出了离线和在线状态下可以从 AppCache 中获取的资源,NETWORK 部分列出了需要绕过 AppCache 机制、通过浏览器标准机制来获取的资源.当 Web 应用配置了 AppCache 之后,其加载流程会有所不同.当 Web 应用被首次加载时,manifest 文件被同时下载,manifest 文件 CACHE 部分声明的所有资源也会被同时获取到本地并存储在 AppCache 中.当一个资源被请求时,浏览器首先检查请求 URL 是否可以在 AppCache 中找到;如果能够找到,则直接从 AppCache 中获取;否则,资源会通过标准机制加载.当 Web 应用在浏览器在线状态下被再次访问时,浏览器会首先通过验证请求检查 manifest 文件是否发生变化.如果没有变化,则直接加载 Web 应用,资源优先从 AppCache 中获取;如果发生变化,浏览器会通过现有机制重新获取 manifest 文件 CACHE 部分的所有资源,然后再继续加载 Web 应用.

基于 AppCache,资源包配置文件可以直接采用 AppCache 的 manifest 文件来实现,Web 应用后端负责生成和维护 manifest 文件.由于 manifest 文件只能接受具体 URL,而 ReWAP 的资源包配置文件是抽象资源的 URL 模板,所以我们在资源映射文件中存储抽象资源 URL 模板和具体资源实际 URL 之间的关系,即可利用 AppCache 的 manifest 文件来加载目标资源.此外,根据 AppCache 的工作原理,支持 AppCache 的 HTML 页面自己本身也会被配置到 AppCache 中,所以本文采用代理 HTML 页面来支持 AppCache.为了能够加载目标 Web 应用,代理 HTML 页面注册了一个 onload 事件的监听器,当 HTML 页面加载完成后,监听器被触发,执行从目标 Web 应用获取根 HTML 资源的逻辑;然后通过动态修改 DOM 树触发目标 Web 应用的加载过程,可以保证用户每次访问 ReWAP 重构的 Web 应用时,HTML 资源都是最新的.

3 基于云-端融合的缓存优化方法

优化浏览器缓存性能的另一种思路是在平台层面重构浏览器的资源加载流程,利用云端辅助刷新客户端

浏览器的缓存状态,从而使 Web 应用的所有资源均可从缓存中获取.根据这一思路,本节提出一种基于云-端融合的缓存优化方法.具体而言,当用户访问某一 Web 应用时,首先在云端预先加载该应用,获取各资源的最新状态,然后结合客户端浏览器的缓存资源情况,将更新的资源同步到本地,刷新缓存状态,进而从缓存中加载目标 Web 应用的资源.

3.1 基于资源加载图的资源同步

在云端预先加载 Web 应用后可以获得各个资源的最新状态,通过比对浏览器缓存的资源状态,就可以仅仅将发生变化的资源和缓存中不存在的资源下载到本地,从而避免冗余传输.由于浏览器的资源解析和资源加载是同时进行的过程,即浏览器解析到的 Web 资源会立即被获取并用于渲染页面,因此,如果将需要更新的资源整体下载到本地后再开始浏览器的加载过程,就会增加页面加载时间,影响用户的访问速度.为了解决这一问题,本文提出资源加载图模型来刻画 Web 资源加载过程,进而基于资源加载图实现资源状态在云端和终端的同步,保证浏览器可以从缓存中及时加载正确的资源.

当云端完成对 Web 应用的渲染之后,可以获得一个资源加载序列

$$R=\{r_1,r_2,...,r_n\},\ r_i=\langle startTime,endTime\rangle.$$

序列中的每个元素包含一个资源加载的开始时间和结束时间.定义资源加载图为一个有向无环图 $G=\langle V,E\rangle$,其中,V 是 Web 应用的资源集合,V 中的每个节点以 URL 作为标识,同时保存资源内容的 MD5 校验码和资源实体.E 是资源之间的依赖关系,E 中的每条边$\langle v_1,v_2\rangle$的定义如下:资源 v_1 的加载结束时间早于资源 v_2 的开始加载时间,并且这个时间差在所有满足该条件的资源中是最小的.根 HTML 资源的父节点是空,并且它是所有按照上述规则无法找到父节点的资源的父节点.资源加载图可以随着 Web 资源的加载过程动态构建和增量式地维护.当一个资源被开始加载时,就可以通过开始时间来确定该资源在图中的位置.根据这一性质,在 Web 应用的加载过程中,可以不断生成部分资源加载图以标识整个应用的加载状态.

当浏览器开始加载 Web 应用的根 HTML 资源时,如果该 Web 应用之前被访问过,首先将相关资源的 MD5 校验码拼接成一个资源指纹串,然后将该指纹串和根 HTML 资源的请求一并发给云端进行处理.如果 Web 应用是第 1 次被访问,则只将根 HTML 资源的请求发送到云端.云端收到请求,根据 Web 应用的 URL 获取最新的资源加载图.然后,对比资源加载图的资源 MD5 值和指纹串的资源 MD5 值:如果资源加载图中的 MD5 值在指纹串中出现,则标记该资源状态为本地可用;否则标记为需要同步.进一步地,云端将更新过的资源加载图返回给浏览器,并且所有 MD5 值未出现在指纹串中的资源(即需要同步的资源)会按照优先级依次从云端同步到浏览器缓存中.在资源加载图中,资源节点的层次越深,表明依赖的其他资源越多,加载时刻就越晚;反之,资源节点的层次越浅,表明该资源可能越早地被请求.换言之,资源加载图中浅层资源的同步优先级要高于深层资源.因此可以根据资源加载图的拓扑序列来确定资源的同步优先级.

浏览器接收到更新的资源加载图之后,根据资源状态开始加载过程:如果一个资源是本地可用,则直接从缓存中取出资源;如果一个资源是需要同步状态的,则当资源完成同步后再加载.云端基于优先级的同步机制可以尽可能地保证当资源被请求时已完成同步.如果一个资源请求在资源加载图中并未找到,则根据下一节介绍的方法来进行资源映射.

3.2 别名资源映射

由于 Web 应用存在别名资源,所以当请求的 URL 无法直接在资源加载图中找到时,浏览器需要确定缓存中是否存在一个资源,该资源的 URL 与请求 URL 不同,但实际是具有同一内容的资源,可直接从缓存加载该资源,无需从网络重复下载资源内容.为此,本文设计了基于资源加载图和 URL 相似度的算法来查找别名资源.

基于资源加载图的资源定位.浏览器加载 Web 应用时可以同时生成一个资源加载图.由于绝大部分资源都是从云端同步到终端的,所以在终端生成的资源加载图与从云端获取的完整资源加载图在结构上是相近的.本文假设同一个 Web 应用在多次渲染时,其资源加载顺序不会发生显著变化.因此,同一个资源在浏览器和云端的资源加载图中应该处于相近的位置.假设从云端获得的资源加载图为 $G_1=\langle V_1,E_1\rangle$,浏览器渲染网页过程中构建

的资源加载图为 $G_2=\langle V_2,E_2\rangle$.给定当前浏览器请求的资源,对于 G_1 中的任一资源节点 $v_e(v_e\in V_1)$, v_t 和 v_e 之间的匹配度可以通过二者之间相同的兄弟节点来度量:

$$M=\frac{\left|S(G_1,v_e)\cap S(G_2,v_t)\right|}{\left|S(G_1,v_e)\cup S(G_2,v_t)\right|},$$

其中,$S(G,v)$表示 v 在资源加载图 G 中的兄弟节点集合.该公式表明,匹配度 M 在 0 到 1 之间.如果 v_t 和 v_e 的 M 值为 0,则认为 v_e 不是 v_t 潜在的别名资源.

基于 URL 相似度的资源匹配.别名资源的 URL 之间存在一定的相似性.本文将 URL 通过常见的分隔符(/、?、=)分成多个部分,得到组成 URL 的单词.通过比较两个 URL 组成单词的相似度,可以定量地度量两个 URL 之间的相似度:两个 URL 相同的单词越多,那么其对应的资源内容相同的可能性就越大.URL 的不同组成部分对判断两个 URL 是否为同一资源的不同别名的重要程度存在差异,所以本文设计了一种动态算法来为不同资源计算其别名 URL 模式.假设已经获取到同一资源内容的不同 URL,首先对这些 URL 切分单词,接着计算出 URL 每一部分发生变化时,该资源保持不变的可能性,用段落关键系数α_{seg}来表示,计算方法为

$$\alpha_{seg}=\frac{n_{moststring}}{n_{URL}+1},$$

其中,$n_{moststring}$表示在该段落中最频繁单词的出现次数,n_{URL}表示同一内容资源的所有 URL 个数.该系数可以在云端动态学习,并存储在资源加载图中.对于两个待匹配的 URL,首先同样对 URL 进行划分;然后,对于内容不同的段落,匹配值设置为 1;对于内容相同的段落,匹配值设置为 $1-\alpha_{seg}$(α_{seg}是相应段落的关键系数);最后,将所有段落的匹配值相乘,可以得到两个 URL 的相似度:

$$Similarity=\prod_{i=1}^{segment\,\sin\,URL}(1-x_i\cdot\alpha_i),$$

其中,α_i为第 i 个段落的关键系数,x_i表示两个 URL 在该段落的字符串值是否相同,若相同,则 x_i=1,若不同,则 x_i=0.根据上式,两个 URL 的相似度始终在 0 到 1 之间.

对于每个待匹配的请求 URL,为了确定在资源加载图中是否存在内容相同的别名资源,首先对待匹配资源与加载图中的每个资源计算位置匹配度 M 和 URL 相似度 *Similarity*,将这两个值相乘,得到最终匹配度.由于目标资源不一定存在,因此需要为匹配度设置一个阈值:当匹配度超过阈值时才认为两个资源是匹配的;如果待匹配资源与资源加载图中所有资源的匹配度都低于阈值,则可以判定缓存中不存在目标资源,即向原 Web 服务器发送请求获取资源.

3.3 系统实现

本文采用双代理架构实现了云-端融合缓存优化系统 SWAROVsky.图 4 展示了 SWAROVsky 系统的整体结构,包括终端代理服务器和云端代理服务器两部分.终端代理服务器接收浏览器的请求,并向浏览器返回资源响应.终端代理服务器向在个人云上搭建的云端代理服务器请求资源,由云端代理服务器从互联网上下载资源,并有选择地向终端进行推送.在此过程中,通过 URL 匹配算法、资源校验码比对、数据压缩与网站预取等手段达到最大程度地减少传输流量并提高加载速度的目的.

用户在个人云上配置需要被优化的 Web 应用列表.对每个 Web 应用,云端代理服务器通过一个浏览器引擎不断访问,记录访问过程中所有的 Web 资源加载信息(图 4 中的①),构建资源加载图,相关数据存储在云端资源库中(图 4 中的②).

当用户在通过浏览器访问一个 Web 应用时,所有的资源请求会被终端代理服务器拦截并由资源映射器处理(图 4 中的③).如果某个资源请求是获取 Web 应用的根 HTML 资源,那么终端代理服务器的资源同步器会与云端代理服务器进行资源同步,将 Web 应用加载所需的资源同步到本地(图 4 中的④和⑤).当终端代理服务器收到资源加载图之后,资源映射器开始响应浏览器的资源请求,并不需要等待所有的资源都传回终端.对于每个资源请求,如果请求 URL 可以直接在资源加载图中找到,那么终端代理服务器会构建响应信息返回给浏览器(图 4 中的⑦);如果请求 URL 无法在资源加载图中找到,那么资源映射器确定是否存在别名资源,如果找到别名资源,

则会根据别名资源来构造响应信息返回给浏览器;否则,资源映射器会发送网络请求,交由原 Web 服务器处理(图 4 中的⑧).当响应信息构造完成后,终端代理服务器随即将其返回给浏览器(图 4 中的⑨).

本文的原型系统目前只针对采用 HTTP 协议的 Web 应用.由于 HTTPS 的默认机制不支持代理、个人云等中间节点,因此目前实现无法直接支持采用 HTTPS 协议的 Web 应用.一种解决方案是将代理服务器设置为可信中间节点,即浏览器和终端代理服务器以及云端代理服务器和原服务器之间可以采用两个独立的 HTTPS 连接进行通信;对于旁路机制,终端代理服务器可以通过一个新的独立 HTTPS 连接请求原服务器.

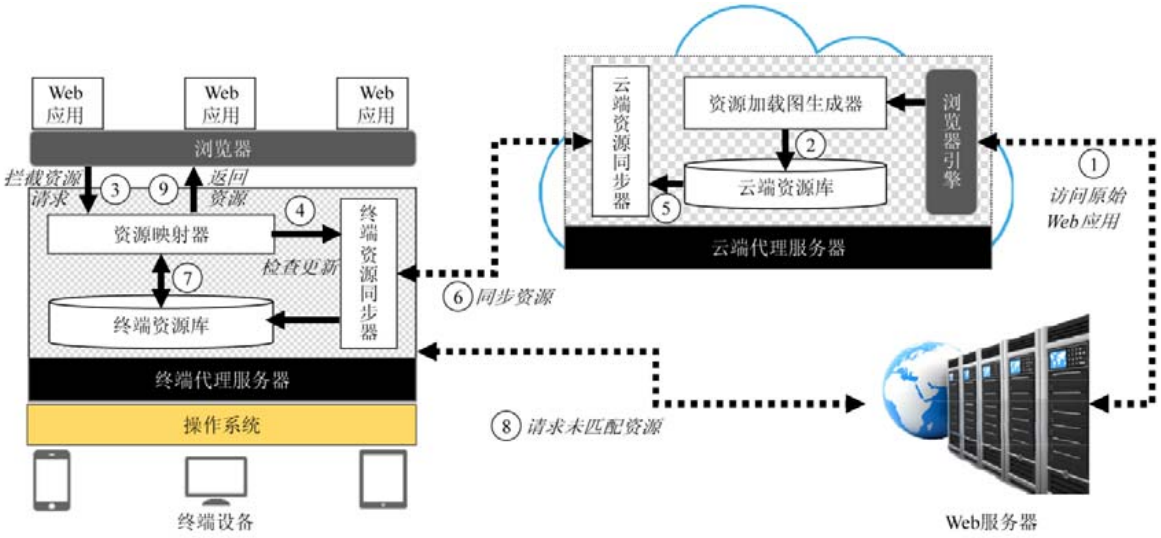


Fig.4 The architecture of SWAROVsky system

图 4 SWAROVsky 系统架构

4 效果评估

本文从 Alexa 排名前 500 的移动 Web 应用中选取了 50 个应用的首页作为开展实验评估的对象,采用仿真实验的方法来精确评估缓存性能的提升.数据收集方法与第 1 节的度量实验类似,通过以较短的周期收集移动 Web 应用在一段时间的资源演化情况,可以仿真用户以不同频率访问移动 Web 应用的行为,进而通过理论计算或实际测量的方式比较原有机制和本文实现的 ReWAP 和 SWAROVsky 系统的缓存性能.本文采用节省的数据流量来定量评估缓存性能的提升.

4.1 ReWAP系统性能评估

由于 ReWAP 需要对现有移动 Web 应用的实现进行少量修改,在没有应用源码的情况下,本文采用仿真计算的方法来分析 ReWAP 所带来的缓存性能提升.通过将收集到的移动 Web 应用资源演化序列,可以计算出在每一时刻的资源包配置文件.接着,通过维护一个理想的浏览器运行仿真器和一个 ReWAP 运行容器的仿真器,可以为每个 Web 应用计算用户在不同再访问间隔下访问时的具体流量消耗,从而可以比较二者之间的流量差异.本文以 30 分钟为粒度、评估用户在再访问间隔从 30 分钟到 1 天的各种情况下(即,30 分钟、1 小时、1.5 小时、…、24 小时)ReWAP 为用户节省流量的平均值.

图 5 展示了实验结果.在 0.5 小时~24 小时的再访问间隔下,相比于传统的浏览器缓存机制,ReWAP 在平均情况下能够节省 8%~51%之间的流量.随着再访问间隔的增加,节省的流量逐渐变少,其原因是资源包发生了变化,需要通过网络更新,带来额外的流量消耗.对于每个再访问间隔,流量节省的分布方差都非常大.在最优情况下,尤其是再访问间隔在 5 小时以内时,流量节省的最大比例甚至接近 100%.相反地,当再访问间隔在 14 小时以上时(图中编号为 29~49 的箱线图),最大的节省流量比例仅为 50%.

为了评估 ReWAP 系统的额外性能开销,本文采用生成测试页面的方式来定量测量.这些测试页面包含的资源个数从 20~100 不等,每个资源大小都是 100KB,采用 JavaScript 的方式动态获取.假设页面的所有资源都配置到资源包中,并且有 10%的资源是通过归一化得到的别名资源.我们为每个测试页面生成对应的 ReWAP 版本,然后依次访问这些测试页面.在每个页面的访问过程中,通过 Android Monitor 工具记录进程的 CPU 和内存负载.结果表明,相比于原始页面,采用 ReWAP 后的应用 CPU 负载增加了 15%,内存没有显著变化.

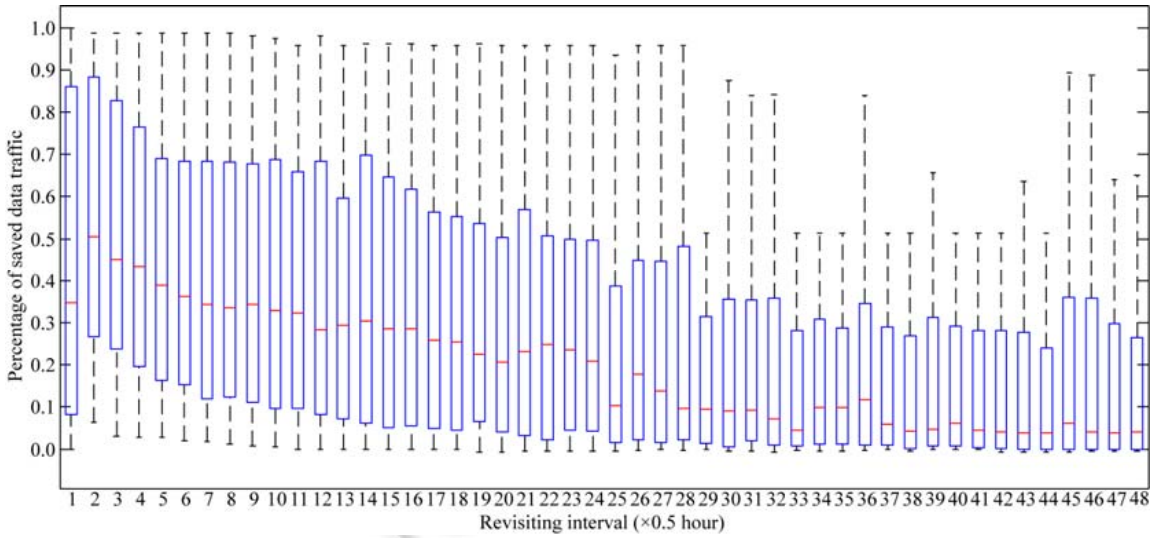


Fig.5 The distribution of saved network traffic under different revisiting intervals by ReWAP

图 5 不同再访问间隔下 ReWAP 节省网络流量的分布

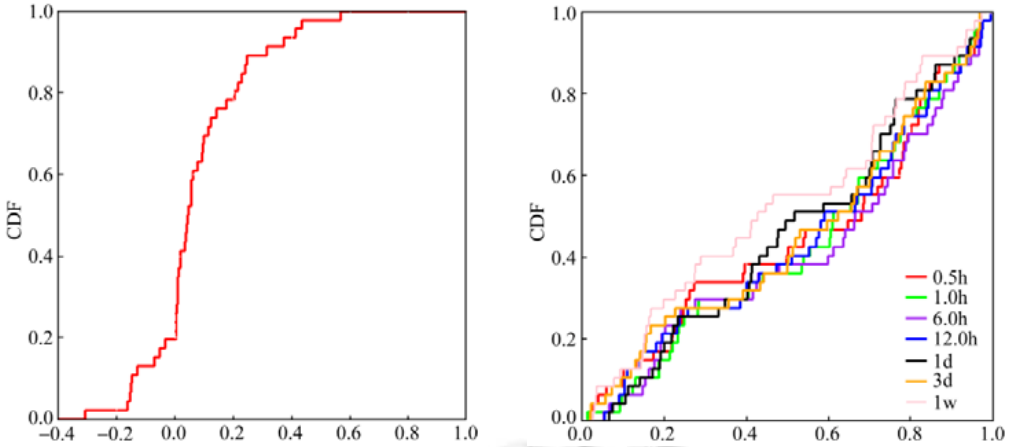
4.2 SWAROVsky系统性能评估

SWAROVsky 系统基于双代理架构实现基于云-端融合的缓存优化,无需对 Web 应用进行修改,所以可在真实环境下评估效果.本文在局域网环境部署了 SWAROVsky 系统,终端使用三星 N7100 手机,采用 Firefox 浏览器来访问移动 Web 应用.云端代理服务器部署在一台联想 T420i 笔记本电脑上.电脑与手机连在同一个 WiFi 下.本文通过网络控制软件为手机和电脑之间设置了 150ms 的网络延迟,以仿真手机的真实网络环境.为了验证在不同再访问间隔的情况下 SWAROVsky 系统的有效性,本文将 50 个 Web 应用的资源数据部署在一台回放服务器上,作为 Web 应用的服务器为前端访问提供资源.

针对第 1 次访问 Web 应用和以不同再访问间隔再次访问同一 Web 应用两种情况,我们测量了使用浏览器直接访问和采用 SWAROVsky 系统访问时的网络流量,进而可以计算流量减少的比例.图 6 展示了实验结果,横坐标代表流量减少比例,纵坐标代表累积分布.采用 SWAROVsky 系统后,Web 应用在第 1 次访问时网络流量减少比例的中位数是 4%,原因是系统的数据流压缩可以减少网络流量.在大约 20%的情况下网络流量有略微增加,其原因是系统在资源同步时需要传输额外信息.当 Web 应用被再次访问时,网络流量减少比例的中位数是 57.6%.不同再访问间隔下的减少比例有所差异,整体来看,SWAROVsky 系统对较短再访问间隔下的性能提升高于较长再访问间隔.

在开展仿真实验的同时,我们记录了终端代理服务器和云端代理服务器的 CPU 和内存使用情况,可以用来评估 SWAROVsky 系统的运行开销.终端代理服务器的 CPU 使用率平均为 2.7%,仅仅是 Firefox 的 1/10.而且在实验过程中,终端代理服务器的 CPU 使用率并没有发生明显变化,而 Firefox 的 CPU 使用率变化较为显著.终端代理服务器的内存使用率同样仅为 Firefox 浏览器的 1/10.随着 Web 应用访问的增加,终端代理服务器的内存使用率变化也不是特别明显.这些结果表明了终端代理服务器是轻量级的,不会对浏览器造成额外的用户体验损耗.对于云端代理服务器,资源同步器的初始内存使用是 37MB,同步资源时的内存使用大约为 70MB.资源加载

图生成器的内存使用为 100MB,当访问 100 个 Web 应用之后,内存使用增加约 30%,CPU 的使用率为 10%.总体来讲,云端代理服务器的运行开销较小,适合在个人云上部署.



(a) 第 1 次访问 (b) 不同再访问间隔下再次访问

Fig.6 The distribution of saved network traffic by SWAROVsky

图 6 SWAROVsky 节省网络流量的分布

5 相关工作

缓存是优化 Web 应用性能的重要手段.在 PC 时代,Web 缓存主要关注服务器端和网络缓存,以减少整个互联网的网络负载,主要技术手段包括代理缓存(proxy caching)、反向代理缓存(reverse proxy caching)、多级缓存(caching hierarchy)等,研究工作主要集中在缓存服务器结构、内容替换策略和缓存一致性等问题[10,11].

随着移动互联网的兴起,移动计算设备已经成为互联网访问的第一入口.移动计算设备经常通过电信蜂窝网络接入互联网.蜂窝网络带宽小、延迟高、不稳定、按流量计费的特性,使得浏览器缓存的重要性日益突出.研究者对移动 Web 应用的浏览器缓存性能开展了度量实验.Wang 等人[12]根据 24 个 iPhone 手机用户 1 年的浏览器访问数据分析了浏览器缓存性能,发现 60%的 Web 资源无法在浏览器缓存中命中.Qian 等人[13]根据基站收集的 1 天 Web 流量数据和 20 个手机用户 5 个月的 Web 浏览记录,研究了移动浏览器缓存的实现缺陷所导致的冗余传输问题,他们发现,两个数据集中分别有 18%和 20%的 Web 流量是冗余的,进而分析了缓存大小限制、未完全遵循 RFC 2616 规范等 12 个原因.Qian 等人的后续工作[14]揭示了 500 个流行网站的缓存低效问题:26%的根 HTML 文件无法缓存、23%的资源被配置了小于 1 小时的过期时间.Wang 等人[15]还研究了以 HTML、CSS、JS 资源的具体内容为判断依据的细粒度缓存的性能,发现在 Web 资源较长时间内仅有部分内容发生变化,验证了细粒度缓存的有效性.

针对移动 Web 应用的浏览器缓存性能问题,Wang 等人[12]提出推测式加载机制来支持同一 Web 应用的不同页面的缓存复用.Zhang 等人[16]在移动操作系统层设置专门的缓存管理模块来处理 HTTP 请求,解决浏览器缓存在实现方面的问题.Lymberopoulos 等人[17]和 Butkiewicz 等人[18]根据用户的 Web 浏览行为建立预测模型,对用户将要访问的 Web 应用通过预取(prefecthing)的方式提前刷新缓存,从而加快应用加载时间.然而,预取的内容如果没有被用户访问则会造成额外的流量开销.针对此问题,Baumann 等人[19]设计了选择式预取策略来决定预取的时机和内容.Pande 等人[20]利用 HTML 5 的 Service Worker 机制实现对移动 Web 应用缓存资源的精确更新.Netravali 等人[21]提出远程控制缓存思想,通过在服务器端重写 URL 以解决别名资源问题.

对比相关工作,本文的度量实验没有依赖被动收集的用户 Web 浏览数据集,而是采用主动式方法,研究 Web 应用的资源演化和缓存配置对浏览器缓存性能造成的影响,进而优化用户频繁周期性访问相同 Web 应用时的缓存性能.在缓存性能优化的总体效果上,本文方法节省的网络流量与近期相关工作[21]的水平相当.

6 总结与展望

本文设计并开展了主动式、规模化的移动 Web 应用浏览器缓存性能度量实验,揭示了浏览器缓存的理论性能上限和实际性能之间的巨大差距,发现了重复请求别名资源、启发式过期时间和保守的过期时间配置这三大影响浏览器缓存性能的原因.针对发现的问题,提出了两种浏览器缓存性能优化方法并实现了原型系统:一种是在应用层基于资源包的缓存优化方法,将 Web 应用相对稳定的资源配置到资源包中,利用 HTML 5 的应用缓存机制实现从资源包加载资源;另一种是在平台层基于云-端融合的缓存优化方法,通过云端预先加载 Web 应用获取资源状态,然后与终端的缓存资源进行同步,实现缓存的精确控制.实验结果表明,用户在 1 天内再次访问相同 Web 应用时,采用本文提出的两种方法分别平均可减少 8%~51%和 4%~58%的数据流量,且系统开销较小.

基于本文工作,未来可以通过分析和预测用户访问的 Web 应用,进一步优化浏览器缓存的整体性能.同时,可以考虑细粒度缓存方法,通过分解 Web 资源的具体内容,实现增量更新,减少缓存失效时的更新开销.

References:

[1] ComSore. Mobile Internet usage Skyrockets in past 4 years to overtake desktop as most used digital platform. https://www.comscore.com/Insights/Blog/Mobile-Internet-Usage-Skyrockets-in-Past-4-Years-to-Overtake-Desktop-as-Most-Used-Digital-Platform

[2] Huang G, Liu X, Zhang Y. A mobile Web application platform with synergy of cloud and client. SCIENCE CHINA Information Sciences, 2013,(1):24–44 (in Chinese with English abstract).

[3] Serrano N, Hernantes J, Gallardo G. Mobile Web apps. IEEE Software, 2013,30(5):22–27.

[4] Wang X, Balasubramanian A, Krishnamurthy A, Wetherall D. Demystifying page load performance with Wprof. In: Proc. of the 10th USENIX Conf. on Networked Systems Design and Implementation (NSDI 2013). 2013. 473–486.

[5] Li D, Hao S, Gui J, William H. An empirical study of the energy consumption of Android applications. In: Proc. of the 30th IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME 2014). 2014. 121–130.

[6] Alexa. http://www.alexa.com/

[7] RFC 2616. http://www.w3.org/Protocols/rfc2616/rfc2616.txt

[8] Zhang T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: Proc. of the 21st Int'l Conf. on Machine Learning (ICML 2004). 2004. 919–926.

[9] Application cache. https://www.w3.org/TR/html5/browsers.html#offline

[10] Wang J. A survey of Web caching schemes for the Internet. ACM SIGCOMM Computer Communication Review, 1999,29(5):36–46.

[11] Barish G, Obraczke K. World Wide Web caching: Trends and techniques. IEEE Communications Magazine, 2000,38(5):178–184.

[12] Wang Z, Lin X, Zhong L, Chishtie M. How far can client-only solutions go for mobile browser speed? In: Proc. of the 21st Int'l Conf. on World Wide Web (WWW 2012). 2012. 31–40.

[13] Qian F, Quah KS, Huang J, Erman J, Gerber A, Mao Z, Sen S, Spatscheck O. Web caching on smartphones: Ideal vs. reality. In: Proc. of the 10th Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys 2012). 2012. 127–140.

[14] Qian F, Sen S, Spatscheck O. Characterizing resource usage for mobile Web browsing. In: Proc. of the 12th Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys 2014). 2014. 218–231.

[15] Wang X, Krishnamurthy A, Wetherall D. How much can we micro-cache Web pages? In: Proc. of the 2014 Conf. on Internet Measurement (IMC 2014). 2014. 249–256.

[16] Zhang Y, Tan C, Qun L. CacheKeeper: A system-wide Web caching service for smartphones. In: Proc. of the 2013 ACM Int'l Joint Conf. on Pervasive and Ubiquitous Computing (UbiComp 2013). 2013. 265–274.

[17] Lymberopoulos D, Riva O, Strauss K, Mittal A, Ntoulas A. PocketWeb: Instant Web browsing for mobile devices. In: Proc. of the 17th Int'l Conf. on Architectural Support for Programming Languages and Operating System (ASPLOS 2012). 2012. 1–12.

[18] Butkiewicz M, Wang D, Wu Z, Madhyastha HV, Sekar V. KLOTSKI: Reprioritizing Web content to improve user experience on mobile devices. In: Proc. of the 12th USENIX Symp. on Networked Systems Design and Implementation (NSDI 2015). 2015. 439–453.

[19] Baumann P, Santini S. Every byte counts: Selective prefetching for mobile applications. In: Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT). 2017,1(2):6:1–6:29.

[20] Pande N, Somani A, Samal SP, Kakkirala V. Enhanced Web application and browsing performance through service-worker infusion framework. In: Proc. of the 2018 IEEE Int'l Conf. on Web Services (ICWS 2018). 2018. 195–202.

[21] Netravali R, Mickens J. Remote-control caching: Proxy-based URL rewriting to decrease mobile browsing bandwidth. In: Proc. of the 19th Int'l Workshop on Mobile Computing Systems & Applications (HotMobile 2018). 2018. 63–68.

附中文参考文献:

[2] 黄罡,刘譞哲,张颖.面向云-端融合的移动互联网应用运行平台.中国科学:信息科学,2013,(1):24–44.

马郓(1989－),男,博士,助理研究员,CCF 专业会员,主要研究领域为系统软件,Web 系统,移动计算.

梅宏(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为系统软件,软件工程.

刘譞哲(1980－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为服务计算,移动计算.