Computer Communications 61 (2015) 48–57

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Optimal chunking and partial caching in information-centric networks

Liang Wang [*], Suzan Bayhan, Jussi Kangasharju

Department of Computer Science, University of Helsinki, Finland

ARTICLE INFO

Article history:
Received 6 July 2014
Received in revised form 17 December 2014
Accepted 20 December 2014
Available online 30 December 2014

Keywords:
Information-centric network
Chunking
In-network caching
Performance analysis
Performance modeling

ABSTRACT

Caching is widely used to reduce network traffic and improve user experience. Traditionally caches store complete objects, but video files and the recent emergence of information-centric networking have highlighted a need for understanding how partial caching could be beneficial. In partial caching, objects are divided into chunks which are cached either independently or by exploiting common properties of chunks of the same file. In this paper, we identify why partial caching is beneficial, and propose a way to quantify the benefit. We develop an optimal n-Chunking algorithm with complexity $O(ns^2)$ for an s-byte file, and compare it with ϵ-optimal homogeneous chunking, where ϵ is bounded by $\mathcal{O}(n^{-2})$. Our analytical results and comparison lead to the surprising conclusion that neither sophisticated partial caching algorithm nor high complexity optimal chunking are needed in information-centric networks. Instead, simple utility-based in-network caching algorithm and low complexity homogeneous chunking are sufficient to achieve the most benefits of partial caching.

1. Introduction

Network caching reduces network traffic by exploiting redundancy in traffic [1] and popularity of content [2]. Different caching strategies have been proposed for different cases, such as web and media caching. An important, yet not widely studied question in caching relates to whether objects should be cached in their entirety (*integral caching*) or if only parts of objects should be cached (*partial caching*). Traditionally, web caching has favored the integral approach, whereas media caching has also considered partial caching.

Information-centric networking (ICN) [3–6] uses caching extensively for content delivery and some ICN approaches by default divide objects into chunks, leading effectively to partial caching. This is suitable for large video files, since they are often only partially accessed [7]. Integral caching may waste cache space by storing parts of objects that are less popular than the most popular parts of the same objects. It appears intuitive to develop efficient partial caching algorithms for optimal handling of such differing popularities inside objects.

However, the reality is somewhat more nuanced, as we show in this paper. Our focus is on understanding how different ways of dividing the object (*chunking*) reflect on performance of caching and how to optimally chunk an object. We show that, while partial caching is useful for partially accessed objects, similar results can be achieved via chunking the object into enough many chunks and using simple caching algorithms. In other words, there is only a very small range of system parameters where sophisticated partial caching algorithms are needed, even with erratic object access patterns.

Even though there is a large body of literature on ICN, chunking analysis has long been overlooked. In addition, many ICN proposals [3–6] adopted various chunking schemes in their architecture design, but how fast the benefit from chunking vanishes is still poorly understood. Consequently, the optimal chunk size also remains as an open research question. To the best of our knowledge, there has not been any thorough theoretical analysis or empirical evaluation of the effects of different chunking schemes on caching. Especially in ICN, as content items are digitally signed and managed at chunk level, finding a good tradeoff point between caching efficiency and maintenance overhead is vital. These unsolved questions have posed a significant challenge to system architects, ISPs and IETF (Internet Engineering Task Force) when they define the specifications of the content management and transportation for the future Internet. We hope our work can fill this gap by providing a deep understanding on the optimal chunking and partial caching on ICN networks.

Specifically, our contributions are as follows:

- We analyze the effects of chunking and develop the concept of *popularity distribution distance* to measure the effectiveness of a chunking scheme. We derive bounds on performance of partial caching and compare the optimal chunking with naive homogeneous chunking analytically.

* Corresponding author.
E-mail address: liang.wang@cs.helsinki.fi (L. Wang).

http://dx.doi.org/10.1016/j.comcom.2014.12.009
0140-3664/© 2014 Elsevier B.V. All rights reserved.

- We demonstrate the performance of partial caching algorithms with different chunking schemes and the experiments confirm our analysis that after a moderate number of chunks, partial caching yields no further benefits.
- Both analytical and experimental results show neither sophisticated partial caching algorithm nor optimal chunking are needed in practice. Instead, a simple utility-based caching algorithm with naive homogeneous chunking is sufficient to achieve most benefits of partial caching.

The rest of the paper is organized as follows. Section 2 describes our system model and Section 3 presents formal analysis on chunking schemes. Section 4 formalizes optimal caching algorithms and evaluates their performance under different chunking schemes. Section 5 introduces a utility-based heuristic and compares its performance with the optimal caching algorithms. Section 6 reviews related work and Section 7 concludes the paper.

2. System model

Consider a network of M routers organized in a general topology. Let R_i denote the router i with cache storage capacity of C_i bytes. This network serves the users that generate requests for files in the set $\mathcal{I}$ with $|\mathcal{I}| = N$. We denote a file by f_i and its size by s_i. All files are stored permanently at the Content Provider (CP) which is represented as the $(M+1)$th router (R_{M+1}). Users interact only with the L *edge routers* – routers connect to the users – also referred to as leaf nodes.[1] A file f_i is divided into n_i smaller units referred to as *chunks*, and jth chunk is denoted as $f_{i,j}$. Denote the probability of request for a file by this file's *popularity* p_i, and similarly denote the popularity of chunk $f_{i,j}$ by $p_{i,j}$. We refer to the popularity vector in both cases by $\mathbf{p} = [p_i]$ (or $\mathbf{p} = [p_{i,j}]$). If an edge router has the requested item in its cache, we call this a *hit* and this item is transmitted to the user directly from this router. In case of a *miss* – the case where the router does not have the item, the request is retrieved from the closest router storing this item. If the item is not stored in the network, it is retrieved from CP.

3. Analysis on chunking

Cutting a file into smaller *chunks* improves caching performance since more fine-grained caching decisions can be made, especially when different parts of the file have different popularities. However, quantifying the effects of chunking and the resulting benefits in partial caching have largely been unexplored. We now present the relationship between chunking and performance, then quantify the benefits of partial caching, and outline the steps of an optimal chunking algorithm.

Although the common understanding is that smaller chunk size can capture user behavior (e.g., frequently-accessed parts of a video) more accurately, the smallest indivisible unit in practice is determined by many other factors, e.g., application configuration, hardware limit, packet size, etc. For the simplicity of presentation, we refer one byte as the smallest unit for the discrete case, and a continuous real function for the continuous case to derive analytical results in closed form. Note that our choice of the word *byte* is only for distinguishing the smallest unit from chunk, rather than indicating byte-granularity chunking in realistic settings.

3.1. Chunking effect: origin of partial caching benefit

Assume that the smallest indivisible unit of a file is one byte and the smallest unit requested by the user is a chunk. Fig. 1 gives an example where a six-byte file is divided into two chunks A and B. Users can access individual bytes in an arbitrary manner and we denote this "real user access pattern" as $\mathcal{M}$. Because of chunking, the real access pattern has to be translated into coarser granularity chunk access pattern, from $\mathcal{M}$ to $\widetilde{\mathcal{M}}$ as in Fig. 1. This distorts the popularity distribution of the bytes since unpopular bytes could be in the same chunk as popular bytes (e.g., bytes 1 and 2 in the figure), thus inflating their observed popularity. Due to this distortion, although the original popularity distribution is $\mathbf{p} = \{\frac{2}{3}, 0, \frac{1}{6}, 0, \frac{1}{6}, 0\}$, the translated popularity distribution becomes $\tilde{\mathbf{p}} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$. We call this distortion from the real $\mathbf{p}$ to the translated $\tilde{\mathbf{p}}$ *chunking effect.*

Chunking effect leads to the popularity d the instruction booklet "Pof the bytes in the same chunk to be equal which often also means over- or under-estimation of popularity. Popularity estimate has direct impact on caching performance:

1. Overestimated popularity increases the chance of caching unpopular content.
2. Underestimated popularity decreases the chance of caching popular content.

Both cases lead to a failure to use cache efficiently because of wasting cache space on unpopular content.

The effect can happen anywhere in the network: at the client, at the server, or at a router whenever the chunk size is bigger than the smallest unit. Nonetheless, the effect is the same since the bytes in the same chunk will be given the same popularity and we lose the information from the real sequence. Vanichpun et al. [8] show that for most demand-driven caching algorithms (e.g., LRU, LFU), it is reasonable to assume that the closer $\tilde{\mathbf{p}}$ is to $\mathbf{p}$, the better caching decision a caching algorithm can make, therefore achieve higher caching performance.

3.2. Popularity distribution distance: quantifying the benefits

Multiple metrics could be used to measure information loss due to the chunking effect. However, simple difference of two distributions has very direct connection to the performance.

Vanichpun et al. [8] show that probability of an object being cached is a function of its translated probability. Let $\mathcal{C} : \mathcal{F}(\widetilde{\mathcal{M}}, C) \to \mathbf{x}$ be a caching algorithm which maps a translated request pattern $\widetilde{\mathcal{M}}$ and cache capacity C to a caching decision vector $\mathbf{x} = \{x_1, x_2, x_3 \ldots\}$, where x_i specifies the probability that item f_i should be kept in the cache. Let $\mathcal{I}$ denote the set of the whole content items with N elements and $\mathcal{I}_\mathcal{C}$ the items cached by $\mathcal{C}$. Assuming unit size items, the total size of the items in the cache sums to the cache capacity: $\sum_{i=1}^{N} x_i = C$. Let us introduce $\mathbf{v} = \{\frac{x_1}{C}, \ldots, \frac{x_i}{C}, \ldots, \frac{x_N}{C}\}$, which is simply normalized version of $\mathbf{x}$.

The performance of $\mathcal{C}$ can be evaluated by its (byte) hit rate $\mathcal{H}$ which is simply the joint probability of an incoming request being for a specific content f_i and this item f_i being stored in the cache. We calculate $\mathcal{H}$ as follows:

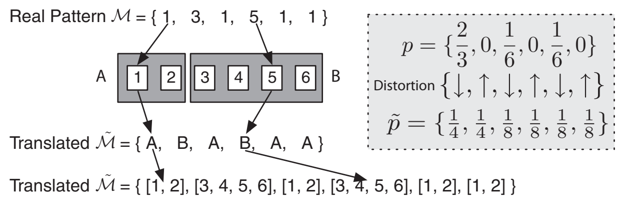
Real Pattern $\mathcal{M}$ = { 1, 3, 1, 5, 1, 1 }
A
1
2
3
4
5
6
B
$p = \{\frac{2}{3}, 0, \frac{1}{6}, 0, \frac{1}{6}, 0\}$
Distortion $\{\downarrow, \uparrow, \downarrow, \uparrow, \downarrow, \uparrow\}$
$\tilde{p} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$
Translated $\tilde{\mathcal{M}}$ = { A, B, A, B, A, A }
Translated $\tilde{\mathcal{M}}$ = { [1, 2], [3, 4, 5, 6], [1, 2], [3, 4, 5, 6], [1, 2], [1, 2] }

Fig. 1. Illustration of chunking effect for a chunking scheme with two chunks A and B.

[1] We use node, router, and cache interchangeably.