Technical Disclosure Commons

Defensive Publications Series

January 2020

Data compression using pre-generated dictionaries

Simon Cooke

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Cooke, Simon, "Data compression using pre-generated dictionaries", Technical Disclosure Commons, (January 16, 2020)
https://www.tdcommons.org/dpubs_series/2876

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Data compression using pre-generated dictionaries

ABSTRACT

A file is compressed by replacing its characters by codes that are dependent on the statistics of the characters. The character-to-code table, known as the dictionary, is typically incorporated into the compressed file. Popular compression schemes reach theoretical compression limit only asymptotically. Small files or files without much intra-file redundancy, either compress poorly or not at all. This disclosure describes techniques that achieve superior compression, even for small files or files without much intra-file redundancy, by independently maintaining the dictionary at the transmitting and receiving ends of a file transmission, such that the dictionary does not need to be incorporated into the compressed file.

KEYWORDS

- Data compression
- Codec dictionary
- Compression dictionary
- Compression ratio
- Page load speed
- Webpage download
- Shannon limit
- Webpage compression
- JavaScript compression
- Web browser

BACKGROUND

A file is compressed by replacing its characters by codes that are dependent on the statistics of the characters. For example, characters that appear in the file with a high frequency are encoded with codes of short length, and vice-versa. As another example, a long string of repeated characters is encoded as a single character followed by the number of repetitions (run-length coding). The character-to-code table, known as dictionary or codec-dictionary, is typically incorporated into the compressed file to make decompression self-bootstrapping.

Popular compression schemes, e.g., lossless entropy-minimization schemes such as Lempel-Ziv, reach theoretical compression limit only asymptotically. Small files or files without much intra-file redundancy, either compress poorly or not at all, since self-bootstrapping compression is performed without taking into consideration the statistics of similar files or documents. Similarly, information theory limits the amount of compression that is possible within a closed, self-bootstrapping compression scheme.

DESCRIPTION

When a webpage that includes content components such as HTML, JavaScript, CSS, etc. or other file is to be transmitted from a server to a client, it is compressed to save bandwidth. Per the techniques of this disclosure, a dictionary is independently maintained at the transmitting and receiving ends of a transmission, such that the dictionary does not need to be incorporated into the transmitted file. The absence of the dictionary in the file transmission enables superior compression.

In the context of this disclosure, codec refers to a device or program that compresses and decompresses data, e.g., maps an uncompressed stream of data (such as text) to a compressed

stream of data, and vice-versa. The codec is used to reduce transmission bandwidth or storage requirements.

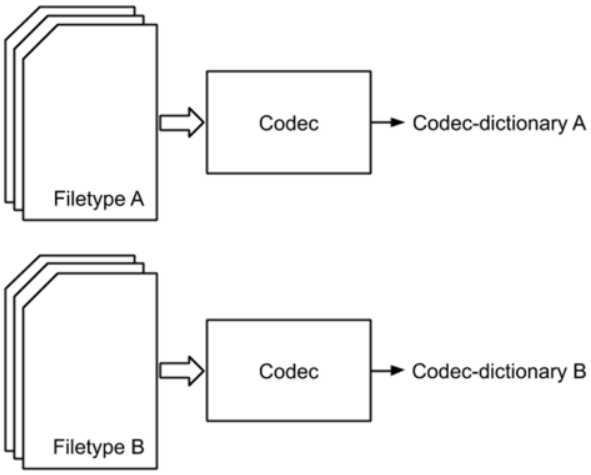

Fig. 1: Pre-generation of dictionaries

Per the techniques of this disclosure, dictionaries are pre-generated for each filetype, as illustrated in Fig. 1. A large number of benchmark files of type A, e.g., HTML files, are analyzed for redundancy by a codec or data compression program to pre-generate a dictionary for files of type A. Similarly, a large number of benchmark files of type B, e.g., CSS files, are analyzed for redundancy to pre-generate a dictionary for file-type B. Effectively, the state of a codec is initialized as if the aggregate of all its benchmark files had been stably compressed, and the most common features and redundancies captured in the startup state of the codec from a stored memo for that codec.

The pre-generated codec dictionaries are provided by a central authority or a chain of trusted authorities. Once generated, dictionaries for each file type are propagated to clients, e.g., web browsers or other client software. Updates to the dictionary by the pre-generating authority are propagated to the population of clients.

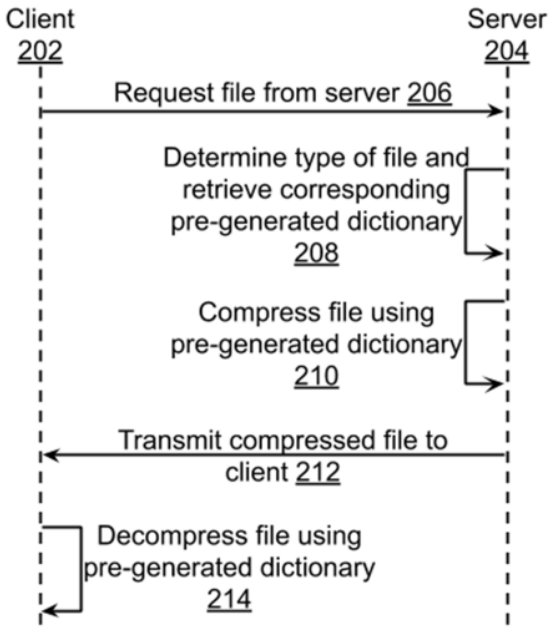

Fig. 2: Compressing and decompressing a file using pre-generated dictionaries

Fig. 2 illustrates compressing and decompressing a file using pre-generated dictionaries, per techniques of this disclosure. A client (202) requests a file (206) from a server (204). For example, the client may be a web browser and the server may be a web server, and the file may be a webpage. The client and server each maintain pre-generated dictionaries for a variety of filetypes.

The server retrieves the requested file, analyzes it, and determines its filetype, e.g., the server assigns the file an identifier as belonging to a class of similar documents. The server finds and caches the best-match pre-generated codec-dictionary class identity (CDCID) that is applicable to the requested file. The server, having access to the pre-generated codec state, compresses (and for static documents, caches) the file using the matching pre-generated dictionary (210). The server transmits the compressed file, or a cached rendition thereof, to the client (212).

Upon receipt of the compressed file, the client retrieves its corresponding pre-generated codec-dictionary class identity (CDCID) and uses the CDCID state to decompress the file (214). The file is presented to the rest of the client software in decompressed form. If the client does not already have the matching pre-generated codec state, the client fetches the appropriate pre-generated codec state, e.g., from a dictionary pre-generating authority, and caches it locally. To reduce storage requirements, a most-recently-used (MRU) cache can be used to store the pre-generated codec state.

The pre-generated codec dictionaries are not fixed or static for a single compression or decompression operation; rather, they form a template enabling (de)compression to occur from a known starting point with a pre-primed dictionary. The dictionary evolves as the document is compressed, allowing redundancies throughout the document to be exploited.

Example: pre-generated dictionary that evolves during compression

For compressing HTML documents, a pre-generated HTML dictionary might include the normal HTML opening and closing tags, but not any common English language words. Consider an HTML string such as:

```
<html><body><p><b>the the the</b><i>the the the</i></p></body><html>
```

During compression, the codec exploits redundancies within the string to cause the dictionary to evolve, and the string might transform into the following:

```
<common opening token><b token>the the the<b close token><i token><3x
the token><i close token><common close token>
```

In this manner, the techniques of this disclosure can deliver better compression than a bootstrapping entropy codec by moving redundant information, e.g., the dictionary, which is derived from the transmitted signal (the server-to-client transmission), to a locally-cached side-

channel. Thus cached, the dictionary, e.g., the pre-generated codec state, can be applied to many client-server transmissions.

Hashes of the pre-generated dictionaries are made available from the chain of trusted dictionary-generating authorities. This enables both servers and central authorities to distribute dictionaries, and enables clients to validate and authenticate dictionaries before use. In this manner, the potential attack vector that is available if bad dictionary states are provided by an attacker, is eliminated. If the hash of the dictionary at the server does not match the hash when the client validated it, the server is viewed as compromised, and the client retrieves a known, good master copy from a trusted authority.

As mentioned before, dictionaries are generated from sets of benchmark files. Each set is assigned a unique CDCID. The benchmark files used to generate a dictionary state are made publicly available for verification.

The MIME type of the file determines which subset of CDCIDs are to be used to encode or decode the files. By doing so, it is possible for the techniques of this disclosure to be applied to any kind of binary (or other) file, e.g., C++ files, JavaScript files, files from particular websites, files from particular locales, icons, metadata, PNG files, image headers, etc. that are to be transferred over the server-client link. CDCIDs can be updated or published at periodic intervals, e.g., for small dictionaries, new sets of dictionaries can be published yearly.

Identifying the CDCID of the dictionary that best compresses a file is performed by simply compressing the same data against each of the CDCID in turn, and finding the one that produces the best results. Keeping a small list of the top *N* dictionaries that produce good results for a given file type (or query generated by the server in a given context) enables quick selection of a dictionary based on the typical content served. The selection and ranking of the top *N*

dictionaries for a given file type can be tweaked and refined as a background process after the data is served, thereby optimizing the next document or file published by the server.

As programming style and usage changes over time, the age of the dictionary itself can be a good gauge of the compression efficiency. In most circumstances, it is likely that there exists one reasonably good all-purpose choice published by a specific server that can be used for the majority of files of a specific type.

CONCLUSION

This disclosure describes techniques that achieve superior compression, even for small files or files without much intra-file redundancy, by independently maintaining the dictionary at the transmitting and receiving ends of a file transmission, such that the dictionary does not need to be incorporated into the compressed file.

REFERENCES

1. "GNU Gzip" available online at https://www.gnu.org/software/gzip/
2. "Oodle Data Compression" available online at http://www.radgametools.com/oodlecompressors.htm