Corpora Documentation

Release 1.0

Krzysztof Dorosz

March 18, 2015

Contents

Corpora is a lightweight, fast and scalable corpus library able to store a collection of raw text documents with additional key-value headers. It uses Berkeley DB (bsddb3 module) for index managing what guarantee speed and bullet-proof. Text storage model is based on chunked flat, human readable text files. This architecture can easily scale up to millions documents, hundred of gigabytes collections.

Corpora module provides four main features:

- create a new corpus,
- append documents to a corpus,
- random access to any document in a corpus using it's unique `id`,
- sequential access to document collection (generator over collection).

Key-Value document headers supports storing any kind of objects seriazable with yaml. Corpora supports only append & read-only philosophy, for more information please read section *Motivation for corpora system*.

CHAPTER 1

Quickstart

Installation:

```
$ sudo pip install corpora
```

Basic usage:

```
>>> from corpora import Corpus
>>> Corpus.create('/tmp/test_corpus')
>>> c = Corpus('/tmp/test_corpus')
>>> c.add('First document', 1)
>>> c.add('Second document', 2)
>>> c.save_indexes()
>>> len(c)
2
>>> c[1]
({'id': 1}, u'First document')
>>> c[2]
({'id': 2}, u'Second document')
>>> for t in c:
...     print t
...
({'id': 1}, u'First document')
({'id': 2}, u'Second document')
```

CHAPTER 2

Contents

2.1 Motivation for corpora system

A natural language processing tasks involve using documents collections (for example document retrieval tasks). There are some ready python tools like NLTK but I found them too complicated for the simple purpose of storing a collection of raw text documents (for example gathered by a crawler). I have also needed a flexible way to store some meta information for each document, for example time of crawling, a semantic evaluation, md5 checksum or any other. It was also important to me to deal with a very large corpora, so system should avoid creating to big one flat file.

My typical use of corpora is append & read-only. For example I crawl a subset of webpages, and create a corpus of founded texts. Then I run some semantic evaluation on the corpus and creating next corpus of result set (or just store information about which documents id where matched). In this way I am avoiding a complex problem of dealing with updates, so system architecture can be very simple (new documents are just appended to the end of available chunk file).

2.2 Corpora API

Corpora module is very simple so far and consists only of one class.

class `corpora.`**`Corpus`**(*path*)
Corpus class is responsible for creating new corpus and also represents a corpus as an object

exception **`ExceptionDuplicate`**
Exception raised when appending document with duplicate `id`

exception `Corpus.`**`ExceptionTooBig`**
Exception raised when document is to big to fit chunk file.

`Corpus.`**`add`**(*text*, *ident*, ***headers*)
Appending new document to a corpus.

static `Corpus.`**`create`**(*path*, ***properties*)
Static method for creating new corpus in the given `path`. Additional properties can be given as named arguments.

`Corpus.`**`get`**(*ident*)
Get random document from a corpus.

`Corpus.`**`get_by_idx`**(*idx*)
Get document pointed by `idx` structure which is offset information in chunk file.

`Corpus.`**`get_chunk`**(*number=None*)
Getter for chunk. Default chunk is current_chunk. Method caches opened chunk files.

`Corpus.`**`get_idx`**(*index*)
Return tuple (chunk, offset, header len, text len) for given `index`

`Corpus.`**`get_ridx`**(*key*)
Return index of idx for given `key`

`Corpus.`**`make_new_chunk`**()
Creates new chunk with next sequential chunk number.

`Corpus.`**`save_config`**()
Saving properties of corpora to config file.

`Corpus.`**`save_indexes`**()
Saving all indexes to apropriate files.

`Corpus.`**`test_chunk_size`**(*new_size*)
Tests if `new_size` data will fit into current chunk.

The only argument `path` should be pointed on directory containing corpus.

2.2.1 Creating new corpus

To create new corpus use static method:

static `Corpus.`**`create`**(*path*, ***properties*)
Static method for creating new corpus in the given `path`. Additional properties can be given as named arguments.

Example:

```
Corpus.create('/tmp/test_corpus', name="My test corpus", chunk_size=1024*1024*10)
```

This will create an empty corpus with 10MB chunk size and name *My test corpus* in directory `/tmp/test_corpus`.

2.2.2 Appending document to corpus

`Corpus.`**`add`**(*text*, *ident*, ***headers*)
Appending new document to a corpus.

`text` a document raw text formed as an Unicode; this will be encoded with the `encode` property of corpus before saving to a chunk.

`ident` a unique identifier of an element, this can be a number or any string (ex. hash value); needed for random access.

> **Warning:** you should assume that `ident` will be converted to string, so 1 and '1' are the same `ident` and are not unique.

`**headers` you can add any additional headers as key-value pairs; values can be any serializable by yaml objects; the key "id" is restricted for storing the document ident.

Example:

```
c = Corpus('/tmp/test_corpus')
c.add(u'This is some text', 1, fetched_from_url="http://some.test/place", related_documents=[1,2,3])
c.add(u'Other text', 2, is_interesting=False)
```

Note: documents are saved with order of appending them, this means that if you add 3 documents with `id` like 2, 1, 3 there will be served in the same order while accessed sequentially.

Warning: as you can see you can add any header to document. There is no pre-configuration what can be set as document header. This is very flexible, but in the same time can lead to problem with consistency of headers among all documents collections. Be sure that you append this same headers to every document in corpus or write your code in a way that will deal with `KeyError` from missing headers.

After adding new documents to a corpus you need to sync indexes to a filesystem.

`Corpus.`**`save_indexes`**`()`
Saving all indexes to apropriate files.

```
c.save_indexes()
```

2.2.3 Sequential access to corpus

Typical use of a corpus is to sequentially access all documents one-by-one. Corpora supports operation with generators.

`Corpus.`**`__iter__`**`()`

Example:

```
c = Corpus('/tmp/test_corpus')
for (headers, text) in c:
    ... some processing
```

This will read a file chunks sequentially what should be as fast as possible.

2.2.4 Random access to corpus

There is also a possibility to access a given document pointed by it's `id`.

`Corpus.`**`__getitem__`**`(`*key*`)`
Interface for get method

`Corpus.`**`get`**`(`*ident*`)`
Get random document from a corpus.

Examples:

```
c = Corpus('/tmp/test_corpus')
print c[1]
print c.get(1)
```

Both lines will print the same document tuple (if exists).

2.2.5 Size of corpus

Standard python `len` is used.

`Corpus.`**`__len__`**`()`
Returns size of document collection

Example:

```
c = Corpus('/tmp/test_corpus')
print len(c)
```

2.2.6 Exceptions

exception `corpora.Corpus.`**`ExceptionTooBig`**
Exception raised when document is to big to fit chunk file.

exception `corpora.Corpus.`**`ExceptionDuplicate`**
Exception raised when appending document with duplicate `id`

2.3 Corpus internal format

Single corpus is stored as a directory. In the directory there are several files important for corpus structure.

2.3.1 File `config`

This file stores yaml formatted dict with properties of corpus. A typical `config` file has following properties:

```
chunk_size: 52428800
current_chunk: 0
encoding: utf-8
name: 50k Internet Corpus
```

> **Warning:** you should not modify `config` file by yourself, unless you really know what you do.

`chunk_size` max size (in bytes) of single corpus chunk

Note: single document must be stored within single chunk, so you cannot store documents larger that `chunk_size`.

`current_chunk` number of current chunk that will be used when appending new document;

Note: chunks are numbered from 0.

`encoding` internal chunk encoding; possibly always `utf-8`.

`name` an optional name for corpus

2.3.2 File `chunkN`

Files like `chunk0`, `chunk1`, `chunk2`, ... contains raw texts and headers. Each chunk can have maximum size of `chunk_size` bytes config property.

Chunk file has a very simple internal format. Documents are stored sequentially (one after another). Each document is represented as yamled header dict and raw document text encoded with `encoding` defined in `config` file.

Internal format of chunk is:

```
[yamled header1]\n
[raw document1 text encoded]\n
[yamled header2]\n
[raw document2 text encoded]\n
...
[yamled headern]\n
[raw documentn text encoded]\n
```

Note: chunks are numbered from 0.

Note: single document must be stored within single chunk, so you cannot store documents larger that `chunk_size`.

An example of two documents long chunk:

```
id: 8
Prof. Wojciech Roszkowski jest oficjalnym kandydatem AWS na
prezesa Instytutu Pamięci Narodowej - zdecydowało prezydium
Klubu Parlamentarnego Akcji Wyborczej Solidarność.
Rzecznik klubu Piotr Żak przypomniał, że zgodnie z ustawą o IPN,
Sejm wybiera prezesa Instytutu większością 3/5. Do wyboru
Roszkowskiego konieczne jest zatem uzyskanie poparcia nie tylko
Unii Wolności, ale także Polskiego Stronnictwa Ludowego.
Politycy PSL, UW i SLD odmawiają deklaracji, czy ich ugrupowania
poprą kandydaturę prof. Roszkowskiego.

id: 20
Papieże Pius IX i Jan XXIII zostaną beatyfikowani 3 września -
ogłosił Watykan. Beatyfikacja obu papieży zbiegnie się z
uroczystościami Wielkiego Jubileuszu Roku 2000.
```

2.3.3 File `idx`

This file contains a list of documents descriptors (indexes in chunk file). This is a list, that contains a tuples like:

- chunk number
- offset of document start in chunk file
- length of header section (with additional `\n`)
- length of text section (with additional `\n`)

This file is managed by DB Berkeley Recno structure.

2.3.4 File `ridx`

This files stores a random access index. Basically it is a hashmap containing a mapping of document `id` to the index in `idx` list.

This file is managed by DB Berkeley Hashmap structure.

2.4 Benchmarks for corpora system

Corpora module was tested for size overhead and latency using an experiment of creating 5M documents corpus.

2.4.1 Corpus size overhead

The total size of an initial raw text file was `2 393 050 900` Bytes (`2.2` GB). In the raw text file there where exactly `5 157 200` documents, what give an approximate size of one document about `464` bytes.

Headers of a document have contained only the `id` of the document.

The total size of final corpus was `2 761 723 865` Bytes (`2.57` GB), containing chunks file sized exactly `2 422 882 196` Bytes (`2.26` GB) and indexes sized exactly `338 841 600` Bytes (`323` MB).

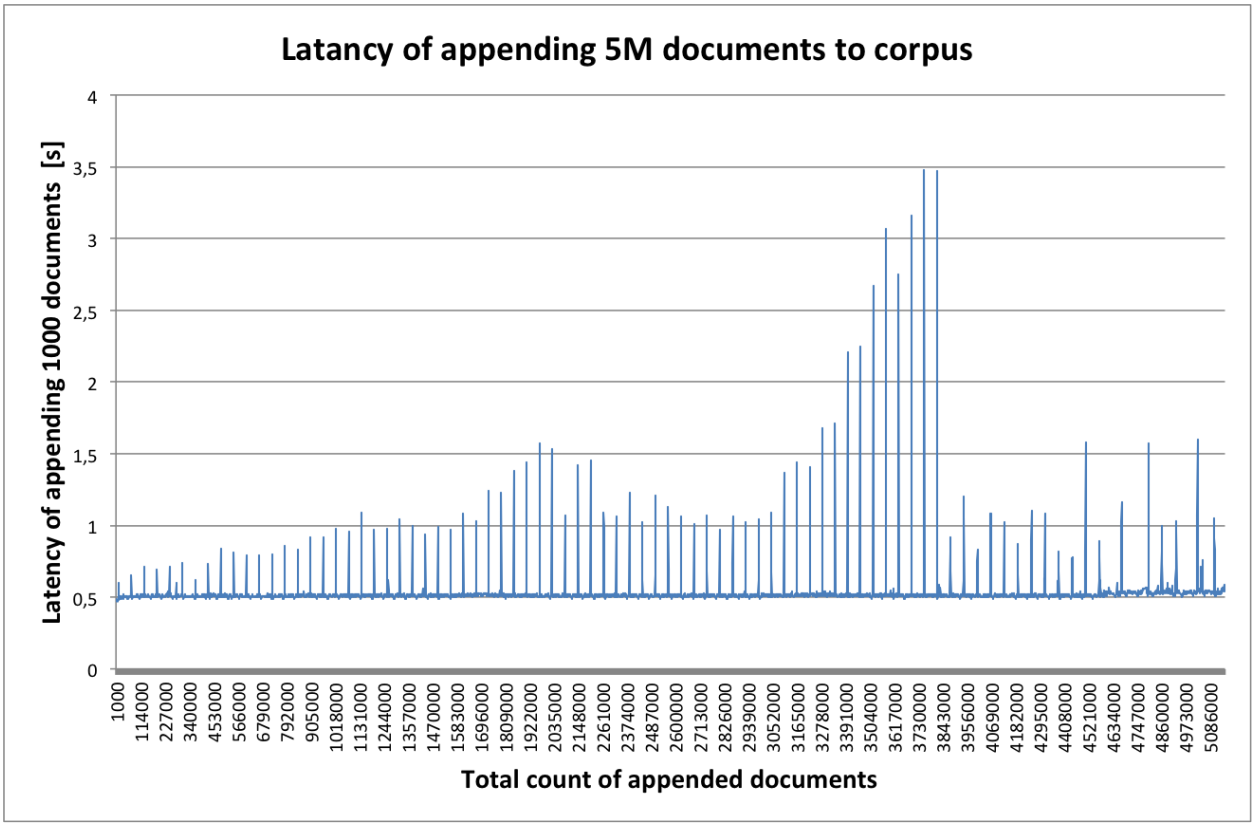
This gives a total overhead of storing raw text in a corpus compared to flat file about `15.4%` and the overhead of chunks with headers compared to flat file about `1.25%`.

Table 2.1: Size and overheads

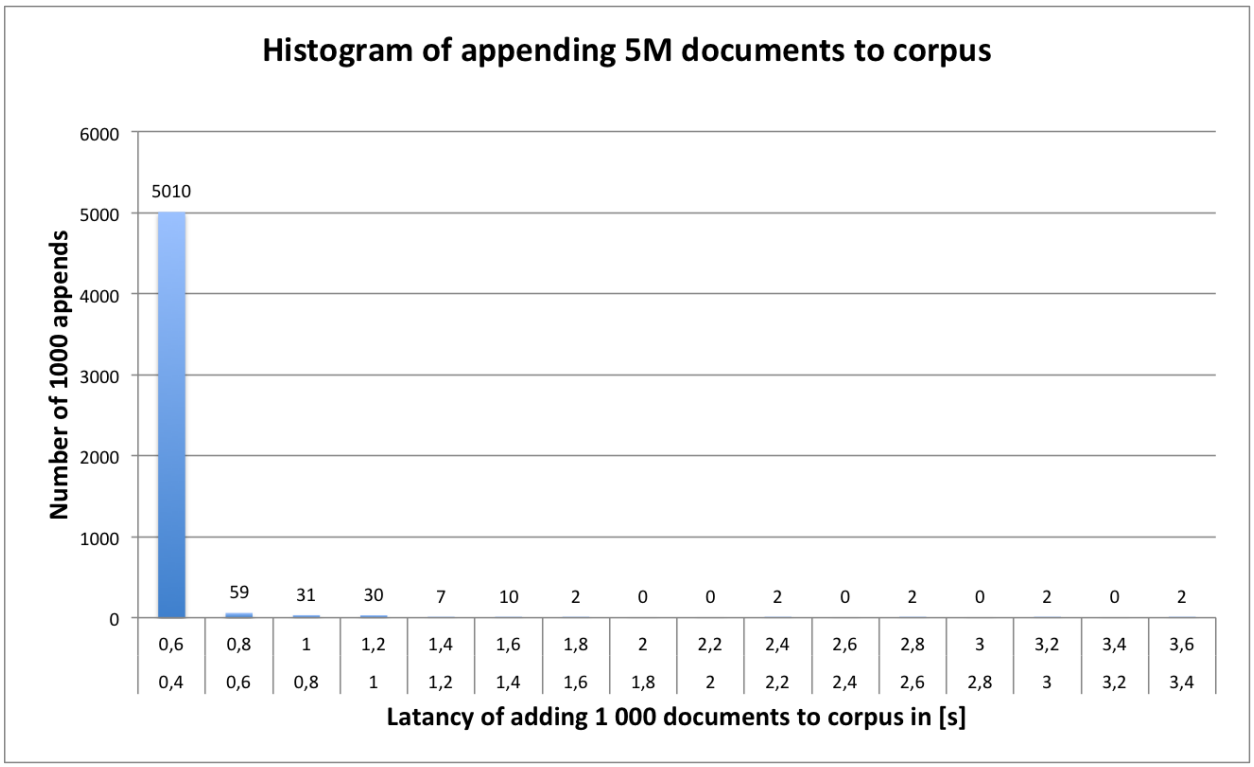
Property	Raw text file	Corpus
Total size	2 393 050 900 B	2 761 723 865 B
Chunk size		2 422 882 196 B
Index size		338 841 600 B
Overhead total		15.4%
Overhead chunks		1.25%

2.4.2 Document append performance

Latency of appending every 1000 documents was measured. The test was performed on `2,8 GHz Intel Core i5`, `12 GB RAM`, Mac OS X Lion `10.7.2` with a HDD drive.



A mean time of appending every 1000 documents was `0.53s` with standard deviation about `0.12s`. On presented graphs latency picks can be observed - possibly caused by DB Berkley indexes synchronizations to disk.



2.5 Credits

2.5.1 Author

Corpora was developed by

Krzysztof Dorosz <cypreess@gmail.com>

Any comments are welcome.

2.5.2 Source code and contribution

Source code is under git version control on github:

https://github.com/cypreess/corpora

License: MIT

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

Symbols

A

C

G

M

S

T