aduana Documentation

Release 1.0

Pedro López-Adeva Fernández-Layos

June 23, 2015

Contents

Contents:

CHAPTER 1

Introduction

Aduana is a component to be used with a web crawler. It contains the logic to decide which page to crawl next. It accepts as inputs crawled pages and it outputs the next pages to be crawled (requests).

Fig. 1.1: Aduana input/output

The main objectives of Aduana are:

- Speed: it must be able to output thousands of requests per second.
- Scalability: it must be able to consider billions of crawled pages.
- Intelligence: it must be able to direct the crawl to interesting pages.

1.1 Components

There are two main components right now: a C library and Python bindings.

The C library does the heavy lifting. In addition it also ships with several command line tools. It's portable ANSI C99 code and all the necessary dependencies are bundled with the library. Ideally you should not concern yourself with this library unless you plan to extend Aduana.

The Python bindings contain low-level bindings to the C library and also:

- Frontera backends. Frontera is an extension to Scrapy which allows to plug different crawl frontier backends. Aduana can be used as a Frontera backend.
- An Aduana server, to be used when crawling using multiple spiders.

1.2 Installation

Use pip:

```
pip install aduana
```

CHAPTER 2

Python library

2.1 Installation

To install just make:

```
pip install aduana
```

It will automatically compile the C library and wrap it using CFFI. Not all parts of the C library are accessible from python, only the necessary ones for the Frontera backends.

Apart from the python module it will also install two scripts:

- aduana-server.py
- aduana-server-cert.py

These scripts are to be used when using the *Distributed spider backend*.

2.2 Using Scrapy/Frontera with Aduana

Check the Frontera documentation, for general instructions about setting up Scrapy, Frontera and custom backends. The workflow specific for Aduana is:

1. Set the backend, either as:

   ```
   BACKEND = 'aduana.frontera.Backend'
   ```

   or if you want to make a distributed crawl with multiple spiders as:

   ```
   BACKEND = 'aduana.frontera.WebBackend'
   ```

2. Set additional options, for example:

   ```
   PAGE_DB_PATH = 'test-crawl'
   SCORER = 'HitsScorer'
   USE_SCORES = True
   ```

3. Run your spider:

   scrapy crawl your_spider_here

2.3 Single spider backend

This backend is the easiest one to run and works by calling directly the wrapped C library. To use it set the backend as:

```
BACKEND = 'aduana.frontera.Backend'
```

Additionally, the following setting are also used by this backend

- `PAGE_DB_PATH`

  String with the path where you want the PageDB to be stored. Note that Aduana will actually make two directories. One will be the one specified by `PAGE_DB_PATH` and the other will add the suffix `_bfs`. This second directory contains the database necessary for the operation of the (best first) scheduler. If this settings is not specified, of it is set to None, the directory will be generated randomly, with suffix `frontera_` and it will be automatically deleted when the spider is closed.

- `SCORER`

  Strategy to use to compute final page scores. Can be one of the following:

  - `None`
  - `'HitsScorer'`
  - `'PageRankScorer'`

- `USE_SCORES`

  Set to `True` if you want that the scorer, in case that it was HITS or PageRank based merges the content scores with link based scores. Default is `False`.

- `SOFT_CRAWL_LIMIT`

  When a domain reaches this limit of crawls per second Aduana will try to make requests to other domains. Default is 0.25.

- `HARD_CRAWL_LIMIT`

  When a domain reaches this limit of crawls per second Aduana will stop making new requests for this domain. Default is 100.

- `PAGE_RANK_DAMPING`

  If the scorer is PageRank then set the damping to this value. Default is 0.85.

2.4 Distributed spider backend

This backend allows to use several spiders simultaneously, maybe at different computers to improve CPU and network performance. It works by having a central server and several spiders connecting to it through a REST api.

The first thing you need to do is launch the server:

```
aduana-server.py --help

usage: aduana-server.py [-h] [--seeds [SEEDS]] [settings]

Start Aduana server.

positional arguments:
  settings         Path to python module containing server settings
```

```
optional arguments:
  -h, --help       show this help message and exit
  --seeds [SEEDS]  Path to seeds file
```

Once the server is launched press Ctrl-C to exit.

The server settings are specified in a separate file that is passed as a positional argument to the `aduana-server.py` script. The reason is that they are settings that will be shared by all spiders that connect to the server.

The following server settings have the same meaning as the ones in the *Single spider backend*.

- `PAGE_DB_PATH`
- `SCORER`
- `USE_SCORES`
- `SOFT_CRAWL_LIMIT`
- `HARD_CRAWL_LIMIT`
- `PAGE_RANK_DAMPING`

Additionally the following settings are available:

- `SEEDS`

  Path to the seeds file, where each line is a different URL. This setting has no default and is mandatory. It can be specified/overriden with the `--seeds` option when launching the server.

- `DEFAULT_REQS`

  If the client does not specify the desired number of requests serve this number. Default number is 10.

- `ADDRESS`

  Server will listen on this address. Default `'0.0.0.0'`.

- `PORT`

  Server will listen on this port. Default 8000.

- `PASSWDS`

  A dictionary mapping login name to password. If `None` then all connections will be accepted. Notice that it uses BasicAuth which sends login data in plain text. If security is of concern then it is adviced to use this option along with `SSL_KEY` and `SSL_CERT`. Default value for this setting is `None`.

- `SSL_KEY`

  Path to SSL keyfile. If this setting is used then `SSL_CERT` must be set too and all communications will be encrypted between server and clients using HTTPS. Default `None`.

- `SSL_CERT`

  Path to SSL certificate. Default `None`.

The Frontera settings to use this backend are:

```
BACKEND = 'aduana.frontera.WebBackend'
```

Additionally, the following setting are also used by this backend

- `SERVER_NAME`

  Address of the server. Default `'localhost'`

- `SERVER_PORT`

Server port number. Default 8000.

- `SERVER_CERT`

  Path to server certificate. If this option is set it will try to connecto to the server using HTTPS. Default `None`.

2.4.1 WebBackend REST API

There are two messages exchanged between the spiders and the server.

- Crawled

  When a spider crawls a page it sends a POST message to `/crawled`. The body is a json dictionary with the following fields:

  - url: The URL of the crawled page, ASCII encoded. This is the only mandatory field.
  - score: a floating point number. If omited defaults to zero.
  - links: a list links. Each element of the links is a pair made from link URL and link score.

  En example message:

```
{ "url"   : "http://scrapinghub.com",
  "score": 0.5,
  "links": [["http://scrapinghub.com/professional-services/", 1.0],
            ["http://scrapinghub.com/platform/", 0.5],
            ["http://scrapinghub.com/pricing/", 0.8],
            ["http://scrapinghub.com/clients/", 0.9]] }
```

- Request

  When the spider needs to know which pages to crawl next it sends a GET message to `/request`. The query strings accepts an optional parameter `n` with the maximum number of URLs. If not specified the default value specified in the server settings will be used. The response will be a json encoded list of URLs. Example (`pip install httpie`):

```
$ http --auth test:123 --verify=no https://localhost:8000/request n==3

HTTP/1.1 200 OK
Date: Tue, 23 Jun 2015 08:40:46 GMT
content-length: 120
content-type: application/json

[
    "http://www.reddit.com/r/MachineLearning/",
    "http://www.datanami.com/",
    "http://venturebeat.com/tag/machine-learning/"
]
```

2.5 Running the examples

To run the single spider example just go to the example directory, install the requirements and run the crawl:

```
cd example
pip install -r requirements.txt
scrapy crawl example
```

To run the distributed spider example we need to dance a little more:

1. Go to the example directory:

```
cd example
```

2. Generate a server certificate:

```
aduana-server-cert.py
```

3. Launch the server:

```
aduana-server.py server-config.py
```

4. Go to the example directory in another terminal and then:

```
scrapy crawl -s FRONTERA_SETTINGS=example.frontera.web_settings example
```

CHAPTER 3

C Library

This section is aimed at developers that want to understand the architecture of the library, in order to extend it.

The library can be compiled and installed independently of the python bindings. To build and install:

```
cd lib
mkdir debug
cd debug
cmake .. -DCMAKE_BUILD_TYPE=Debug
make && sudo make install
```

When trying to understand some code I like to start with the data structures that make the inputs and the outputs of the code. *CrawledPage* is the input of Aduana and the best place to start.

3.1 CrawledPage

3.1.1 Data structures

struct `CrawledPage`
The information that comes with a crawled page.

Public Members

char ***`url`**
ASCII, null terminated string for the page URL

PageLinks ***`links`**
List of links inside this page

double **`time`**
Number of seconds since epoch

float **`score`**
A number giving an idea of the page content's value

char ***`content_hash`**
A hash to detect content change since last crawl. Arbitrary byte sequence

size_t **`content_hash_length`**
Number of byes of the content_hash

The utility of *CrawledPage::url* and *CrawledPage::links* are quite obvious, the others need an explanation:

- *CrawledPage::time*: this is used to compute how often a page changes and also it would be useful for a revisiting schedule to know how much time has passed since the page was crawled.
- *CrawledPage::score*: one of the objectives of Aduana is to guide the crawl to interesting pages. Since the definition of interesting is application dependent each crawler can give a measure of how interesting they found the page to be. How this number will be exactly used depends on which scorer are we going to use. This field is not mandatory and actually Aduana can be configured to ignore it.
- *CrawledPage::content_hash*: in order to detect if a page has changed this hash is compared with the hash previously stored for this same page. If the hash has changed we consider that the page has changed. Notice that the content hash is also application dependent: some applications may consider that the page has changed only if there are new links, others will consider a page has changed if the body text, after stripping HTML tags, has changed, etc... This field can be ignored too, in which case the pages will be considered as unchanging.

C is not known for its powerful and flexible data structures. In order to store a variable number of links per crawled page we implement this resizable array. Each time we run out of allocated memory the size of the reserved memory is doubled.

struct PageLinks

A (resizable) array of page links.

Initially: n_links = 0 m_links = PAGE_LINKS_MIN_LINKS

Always: 0 <= n_links <= m_links

Public Members

LinkInfo ***link_info**

Array of *LinkInfo*

size_t **n_links**

Number of items inside link_info

size_t **m_links**

Maximum number of items that can be stored inside link_info

Initially we reserve this number of links

PAGE_LINKS_MIN_LINKS

Allocate at least this amount of memory for link info

Finally, each link not only carries an URL, but also a score. The score gives an idea of how good the (maybe uncrawled) link is, according to the web crawler. Think of the link score as an approximation to *CrawledPage::score* when we have not crawled the link yet.

struct LinkInfo

The information that comes with a link inside a crawled page.

The link score is used to decide which links should be crawled next. It is application dependent and tipically computed by looking at the link surrounding text.

Public Members

char ***url**

ASCII, null terminated string for the page URL

float **score**
An estimated value of the link score

3.1.2 Constructor/Destructor

CrawledPage ***crawled_page_new** (**const** char **url*)
Create a new *CrawledPage*

url is a new copy

The following defaults are used for the different fields:

- links: no links initially. Use *crawled_page_add_link* to add some.
- time: current time
- score: 0. It can be setted directly.
- content_hash: NULL. Use *crawled_page_set_hash* to change

Return

NULL if failure, otherwise a newly allocated *CrawledPage*

void **crawled_page_delete** (*CrawledPage* **cp*)
Delete a Crawled Page created with *crawled_page_new*

3.1.3 Manipulate links

int **crawled_page_add_link** (*CrawledPage* **cp*, **const** char **url*, float *score*)
Add a new link to the crawled page

const *LinkInfo* ***crawled_page_get_link** (**const** *CrawledPage* **cp*, size_t *i*)
Get a pointer to the link

size_t **crawled_page_n_links** (**const** *CrawledPage* **cp*)
Get number of links inside page

3.1.4 Set content hash

int **crawled_page_set_hash** (*CrawledPage* **cp*, **const** char **hash*, size_t *hash_length*)
Set content hash

The hash is a new copy

int **crawled_page_set_hash128** (*CrawledPage* **cp*, char **hash*)
Set content hash from a 128bit hash

int **crawled_page_set_hash64** (*CrawledPage* **cp*, uint64_t *hash*)
Set content hash from a 64bit hash

int **crawled_page_set_hash32** (*CrawledPage* **cp*, uint32_t *hash*)
Set content hash from a 32bit hash

3.2 PageInfo

3.2.1 Data structures

This structure contains all we know about a given page, and it's changed as new `CrawledPage` arrive.

And here it is:

struct `PageInfo`
The information we keep about crawled and uncrawled pages

PageInfo are created at the PageDB, that's why there are no public constructors/destructors available.

Public Members

char ***`url`**
A copy of either *CrawledPage::url* or *CrawledPage::links*[i]

uint64_t **`linked_from`**
The page that first linked this one

double **`first_crawl`**
First time this page was crawled

double **`last_crawl`**
Last time this page was crawled

size_t **`n_changes`**
Number of content changes detected between first and last crawl

size_t **`n_crawls`**
Number of times this page has been crawled. Can be zero if it has been observed just as a link

float **`score`**
A copy of the same field at the last crawl

size_t **`content_hash_length`**
Number of bytes in *PageInfo::content_hash*

char ***`content_hash`**
Byte sequence with the hash of the last crawl

3.2.2 Constructor/Destructor

There is no constructor available for this structure. The reason is that they are automatically created from the info inside `CrawledPage` when `page_db_add()` is called.

void **`page_info_delete`**(*PageInfo* **pi*)
Destroy *PageInfo* if not NULL, otherwise does nothing

3.2.3 Functions

int **`page_info_print`**(**const** *PageInfo* **pi*, char **out*)
Write printed representation of *PageInfo*.

This function is intended mainly for debugging and development. The representation is: first_crawl last_crawl n_crawls n_changes url

Each field is separated with an space. The string is null terminated. We use the following format for each field:

- first_crawl: the standard fixed size (24 bytes) as output by ctime. For example: Mon Jan 1 08:01:59 2015
- last_crawl: the same as first_crawl
- n_crawls: To ensure fixed size representation this value is converted to double and represented in exponential notation with two digits. It has therefore always 8 bytes length: 1.21e+01
- n_changes: The same as n_crawls
- url: This is the only variable length field. However, it is truncated at 512 bytes length.

Return

size of representation or -1 if error

Parameters

- `pi` - The PageInfo to be printed
- `out` - The output buffer, which must be at least 580 bytes long

float **page_info_rate**(**const** *PageInfo* *pi)

Estimate change rate of the given page. If no valid rate can be computed return -1.0, otherwise a valid nonnegative change rate.

3.3 PageDB

This is one of the main components of the library. Here we store all the `PageInfo` and how pages are linked between them.

The first thing to understand is that there are two different ways to refer to a given page, using either the URL hash or the *index*. Both ways of addressing the page are linked in the *hash2idx* database.

3.3.1 URL hash

The URL hash is computed using the following function:

uint64_t **page_db_hash**(**const** char *url)

Hash function used to convert from URL to hash.

The hash is a 64 bit number where the first 32 bits are a hash of the domain and the last 32 bits are a hash of the full URL. In this way all URLs whith the same domain get grouped together in the database. This has some good consequences:

1. We can access all pages inside a domain by accessing the first of them in the database and moving sequentially.
2. When streaming links this improves locality since pages in the same domain tend to have similar links.

When a new `CrawledPage` arrives we compute the hash of `CrawledPage::url` and use this as the key inside the *hash2info* database, to retrieve the associated `PageInfo`. If no entry is found inside the database a new one is created. We do the same with each one of the links inside `CrawledPage::links`.

The following two functions are useful to extract the different parts of the hash.

uint32_t **page_db_hash_get_domain**(uint64_t *hash*)
Extract the domain hash from the full hash

uint32_t **page_db_hash_get_url**(uint64_t *hash*)
Extract the URL hash from the full hash

3.3.2 Index

We could store links between pages using their URL hash, for example, as a list of pairs of the form:

```
004619df1e9191ff 004619df1eb839e2
004619df1e9191ff 004619df1f1a5477
004619df01e223ae 00115773f1ea355c
...
```

However the hashing would spoil one interesting property of links: locality. Locality means that pages usually link to pages inside their same domain. For example, here are the first links extracted from the front page of Wikipedia:

```
https://en.wikipedia.org/wiki/Main_Page#mw-head
https://en.wikipedia.org/wiki/Main_Page#p-search
https://en.wikipedia.org/wiki/Wikipedia
https://en.wikipedia.org/wiki/Free_content
https://en.wikipedia.org/wiki/Encyclopedia
https://en.wikipedia.org/wiki/Wikipedia:Introduction
https://en.wikipedia.org/wiki/Special:Statistics
https://en.wikipedia.org/wiki/English_language
```

Locality can also happen when there are several links outgoing to the same domain, but a different one of the originating page. For example, from among the 135 links at the front page of Hacker News more than 100 remained on the same domain but there were also the following groups:

```
http://www.ycombinator.com/
http://www.ycombinator.com/apply/

https://github.com/blog/2024-read-only-deploy-keys
https://github.com/whamtet/Excel-REPL
https://github.com/tadast/switching-to-contracting-uk/blob/master/README.md
https://github.com/HackerNews/API
```

Instead of storing links using the URL hash we instead assign each page an integer, that starts at zero with the first page and it's automatically incremented when a new page is added to the database. Links are stored then as lists where the first element is the originating page index and the rest of the elements are the indices of the outoging links. For example, taken from a real crawl:

```
7 1243 1245 1251 1254 1260 1262 1263
  1264 1267 1269 1271 1274 1275 1276
  1277 1280 1283 1286 1289 1291 1295
  1309 1311 ...
```

Since we want be able to perform big crawls with billions of pages we use 64 bit integers for the indices, which means they still take as much space as the URL hashes. However, these links are delta-encoded: starting at the second element of the list we substract the previous one:

```
7 2 6 3 6 2 1 1 3 2 2 3 1 1 1 3 3 3 3 2 4 14 2 ...
```

Finally we use varint encoding for each integer. As you can see in the above example each link requires just 8 bits, instead of the 64 bits (or 32 bits if somehow we could reuse the domain part of the hash) URL hashing would.

Having indices instead of hashes is also convenient for the PageRank and HITS algorithms. They can store the pages scores using arrays where the position of each page inside those arrays are just their index. Having fast O(1) access time greatly improves the speed of the computation when using billions of pages. Besides, locality also helps access speed, even when working in-memory.

The *index* for a given page is automatically created when `page_db_add()` is called.

3.3.3 Data structures

struct `PageDB`

Page database.

We are really talking about 4 diferent key/value databases:

- info: contains fixed size information about the whole database. Right now it just contains the number of pages stored.
- hash2idx: maps URL hash to index. Indices are consecutive identifier for every page. This allows to map pages to elements inside arrays.
- hash2info: maps URL hash to a PageInfo structure.
- links: maps URL index to links indices. This allows us to make a fast streaming of all links inside a database.

Public Members

char ***`path`**

Path to the database directory

TxnManager ***`txn_manager`**

The transaction manager counts the number of read and write transactions active and is capable of safely performing a database resize

DomainTemp ***`domain_temp`**

Track the most crawled domains

int **`persist`**

If true, do not delete files after deleting object

enum `PageDBError`

Values:

`page_db_error_ok` = 0

No error

`page_db_error_memory`

Error allocating memory

`page_db_error_invalid_path`

File system error

`page_db_error_internal`

Unexpected error

`page_db_error_no_page`

A page was requested but could not be found

3.3.4 Constructor/Destructor

PageDBError **page_db_new** (*PageDB* ***db*, **const** char **path*)
Creates a new database and stores data inside path

Return

0 if success, otherwise the error code

Parameters

- `db` - In case of *page_db_error_memory* *db could be NULL. In case of other failures it is nevertheles allocated memory so that the error code and message can be accessed.
- `path` - Path to directory. In case it doesn't exist it will created. If it exists and a database is already present operations will resume with the existing database. Note that you must have read, write and execute permissions for the directory.

PageDBError **page_db_delete** (*PageDB* **db*)
Close database

Close database, delete files if it should not be persisted, and free memory

3.3.5 Add page

PageDBError **page_db_add** (*PageDB* **db*, **const** *CrawledPage* **page*, *PageInfoList* ***page_info_list*)
Update PageDB with a new crawled page

It performs the following actions:

•Compute page hash

•If the page is not already into the database:

–It generates a new ID and stores it in hash2idx

–It creates a new *PageInfo* and stores it in hash2info

•If already present if updates the *PageInfo* inside hash2info

•For each link:

–Compute hash

–If already present in the database just retrieves the ID

–If not present:

∗Generate new ID and store it in hash2idx

∗Creates a new *PageInfo* and stores it in hash2info

•Create or overwrite list of Page ID -> Links ID mapping inside links database

Return

0 if success, otherwise the error code

Parameters

- `db` - The database to update
- `page` - The information of the crawled page

- `page_info_list` - If not NULL this function will allocate and populate a new *PageInfoList* which contains the PageInfo of the updated pages. It is your responsability to call when you no longer need this structure.

3.3.6 Get info from database

PageDBError **page_db_get_info** (*PageDB* *db, uint64_t *hash*, *PageInfo* **pi*)
Retrieve the *PageInfo* stored inside the database.

Beware that if not found it will signal success but the *PageInfo* will be NULL

PageDBError **page_db_get_idx** (*PageDB* *db, uint64_t *hash*, uint64_t **idx*)
Get index for the given URL

PageDBError **page_db_get_scores** (*PageDB* *db, *MMapArray* **scores*)
Build a *MMapArray* with all the scores

float **page_db_get_domain_crawl_rate** (*PageDB* *db, uint32_t *domain_hash*)
Get crawl rate for the given domain

3.3.7 Database settings

void **page_db_set_persist** (*PageDB* *db, int *value*)
Set persist option for database

PageDBError **page_db_set_domain_temp** (*PageDB* *db, size_t *n_domains*, float *window*)
Set domain temperature tracking options

3.3.8 Export database

This functions are used by the *page_db_dump* command line utility.

PageDBError **page_db_info_dump** (*PageDB* *db, FILE **output*)
Dump database to file in human readable format

PageDBError **page_db_links_dump** (*PageDB* *db, FILE **output*)
Dump database to file in human readable format

3.4 PageInfoList

This structure exists just because `page_db_add()` needs a way of returning which pages had their info created/modified. This information is necessary for schedulers. It's just a linked list so we are not going to make more comments about it.

3.4.1 Data structures

struct PageInfoList
A linked list of PageInfo (and hash), to be returned by *page_db_add*

Public Members

uint64_t **hash**
Hash inside the hash2info database

PageInfo ***page_info**
Info inside the hash2info database

struct *PageInfoList* ***next**
A pointer to the next element, or NULL

3.4.2 Constructor/Destructor

PageInfoList ***page_info_list_new** (*PageInfo* **pi*, uint64_t *hash*)
Create a new *PageInfoList*, with just one element.

Return

A pointer to the first element of the list, or NULL if failure

Parameters

- `pi` - The PageInfo to add. From this point it is the property of the list, so deleting the list deletes this element.
- `hash` -

void **page_info_list_delete** (*PageInfoList* **pil*)
Deletes the list and all its contents

3.4.3 Functions

PageInfoList ***page_info_list_cons** (*PageInfoList* **pil*, *PageInfo* **pi*, uint64_t *hash*)
Add a new element to the head of the list.

Return

A pointer to the first element of the list, or NULL if failure

Parameters

- `pi` - The PageInfo to add. From this point it is the property of the list, so deleting the list deletes this element.
- `hash` -

3.5 LinkStream

Maybe the most interesting stream going out of `PageDB` is the link stream, because it's the main interface between `PageDB` and the different scorers like PageRank and HITS. This stream outputs a list of `Link`, which are just pairs of *from* and *to* indices. Right now, because of the way links are stored inside the database the stream groups together all the links with the same *from* index, however this could change in the future and it's actually not necessary for the current PageRank or HITS implementations.

The reason for using a link stream is that when billions of pages are crawled the size of the links database can grow to several hundreds of megabytes.

3.5.1 Data structures

struct `PageDBLinkStream`

Public Members

MDB_cursor ***`cur`**
PageDB where links database is stored Cursor to the links database

uint64_t **`from`**
Current page

uint64_t ***`to`**
A list of links

size_t **`n_to`**
Number of links

size_t **`i_to`**
Current position inside *to*

size_t **`m_to`**
Allocated memory for *to*. It must be that *n_to* <= *m_to*.

size_t **`n_diff`**
Number of out domain links

int **`only_diff_domain`**
If true only links that go to a different domain will be streamed

struct `Link`

3.5.2 Constructor/Destructor

void **`page_db_link_stream_delete`**(*PageDBLinkStream* **es*)
Delete link stream and free any transaction hold inside the database.

3.5.3 Functions

The signature of these functions use *void* because they must agree with the following interfaces:

typedef `StreamState( LinkStreamNextFunc)(void *state, Link *link)`

for

StreamState **`page_db_link_stream_next`**(void **es*, *Link* **link*)
Get next element inside stream.

Return

::link_stream_state_next if success

and

typedef `StreamState( LinkStreamResetFunc)(void *state)`

for

StreamState **page_db_link_stream_reset** (void *es)
Rewind stream to the beginning

3.6 HashInfoStream

3.6.1 Data structures

This is used by the command line utility *page_db_find*, which iterates over all the pages and returns which ones have their URL matching some regexp.

struct HashInfoStream
Stream over HashInfo inside *PageDB*

Public Members

MDB_cursor ***cur**
Cursor to info database

3.6.2 Constructor/Destructor

PageDBError **hashinfo_stream_new** (*HashInfoStream* **st*, *PageDB* **db*)
Create a new stream

void **hashinfo_stream_delete** (*HashInfoStream* **st*)
Free stream

3.6.3 Functions

StreamState **hashinfo_stream_next** (*HashInfoStream* **st*, uint64_t **hash*, *PageInfo* ***pi*)
Get next element in stream

3.7 HashIdxStream

This is used in two different places. The first one is the command line utility *page_db_links* which returns which pages link or are linked from other page.

The other more important use case is inside schedulers, which after pages scores are updated, need to iterate over all of them to see which ones have changed enough to be rescheduled.

3.7.1 Data structures

struct HashIdxStream
Stream over hash/index pairs inside *PageDB*

Public Members

MDB_cursor ***cur**
Cursor to the hash2idx database

3.7.2 Constructor/Destructor

PageDBError **hashidx_stream_new** (*HashIdxStream* ***st*, *PageDB* **db*)
Create a new stream

void **hashidx_stream_delete** (*HashIdxStream* **st*)
Free stream

3.7.3 Functions

StreamState **hashidx_stream_next** (*HashIdxStream* **st*, uint64_t **hash*, size_t **idx*)
Get next element in stream

3.8 DomainTemp

This is used inside `PageDB` to track how many times the most often domains are crawled. This information will in turn be used by the scheduler, which will try to not serve requests for the most crawled domains.

Ideally, for each domain we would store a (growing) list of timestamps when some page in the domain has been crawled. With this list in hand we could answer questions like *How many times the domain has been crawled in the last 60 seconds?*. Instead of that we make the following approximation: imagine that we store only how many times the domain has been crawled in the last T seconds. We don't know how the crawls have been distributed in that time, it could be that thay are distributed all at the beginning:

or maybe following some strange pattern:

Instead we will assume they are evenly distributed:

Now, if some time t is elapsed without any more crawled, how many crawls remain in the time window?

The answer is that since there are n crawls evenly distributed then there are n/T crawls per second, and then $n\frac{t}{T}$ have moved out of the time window.

$$n(t_0 + t) - n(t_0) = n(t_0)\frac{t}{T}$$

If $t \to dt$ then we have the following differential equation:

$$\frac{dn}{dt} = -\frac{1}{T}n$$

The solution of the above equation is obviously:

$$n(t) = n(0)e^{-\frac{t}{T}}$$

And n would evolve following some similar shape to:

The above figure has a time window of just 2 seconds and there are crawls at instants 1, 2.5, 2.6, 2.7, 4 and 5.

3.8.1 Data structures

struct `DomainTemp`
Tracks how "hot" are the most crawled domains.

We want to avoid crawling the same domain repeatedly. For this purpose this structure tracks how many times a domain has been crawled in the specified time window. For performance reasons an approximation of the actual number of crawls is maintained. Under certain assumptions it can be shown that if 'n' is the number of crawled for a domain it follows the following (cool down) differential equation:

$$\frac{dn}{dt} = -\frac{1}{T}n$$

where T is the time window.

Public Members

DomainTempEntry ***`table`**
An array of domain/temperature pairs

size_t **`length`**
Length of *DomainTemp::table*

float **`time`**
Last time temperatures were updated

float **`window`**
Time window to consider in the cooldown

struct `DomainTempEntry`
Associate a domain hash with a temperature

Public Members

uint32_t **`hash`**
Domain hash

float **`temp`**
Domain temperature: an estimation of how many times the domain has been crawled in the time window

3.8.2 Constructor/Destructor

DomainTemp ***`domain_temp_new`**(size_t *length*, float *window*)
Create a new domain temp tracking structure

Return

A pointer to the new struct of NULL if failure

Parameters

- `length` - Maximum number of domains to track
- `window` - Time window

void **domain_temp_delete**(*DomainTemp* *dh)
Free memory

3.8.3 Functions

void **domain_temp_update**(*DomainTemp* *dh, float *t*)
Updates temp up to current time t

void **domain_temp_heat**(*DomainTemp* *dh, uint32_t *hash*)
Adds another count to domain.

If the domain already in already tracked its counter is incremented. If the domain is not present then we try to initialize it in an empty slot. If not empty slot is available then the domain with fewest crawls is replaced with the new domain if its counter is below 1.

float **domain_temp_get**(*DomainTemp* *dh, uint32_t *hash*)
Gets domain temp

3.9 Error handling

Errors are signaled in the following ways:

- For functions not returning pointers 0 means success and any other value some kind of failure. Usually an enumeration of error codes is defined, otherwise -1 is used as failure code.
- For functions returning pointers failure is signaled returning a null pointer.
- If the causes of error are varied enough the structures inside this library have an *Error* structure, which contains the error code and an error message. The error message usually resembles an stack trace to aid debugging the problem.

3.9.1 Data structures

MAX_ERROR_LENGTH
Maximum length of error message

struct Error

Public Members

pthread_mutex_t **mtx**
Make operations on errors atomic.

If an error is produced dealing with this mutex it will be silently ignored

int **code**
Error code, depends on the application but 0 always signals no error

char **message**[MAX_ERROR_LENGTH+1]
A descriptive message associated with the error code. If no error then it contains "NO ERROR"

3.9.2 Constructor/Destructor

void **error_init** (*Error* *error)
Initialize structure.

Error code is set to 0 and message to "NO ERROR".

void **error_destroy** (*Error* *error)
Clean up. Will NOT free `error`

Error ***error_new** (void)
Allocate and initialize a new error structure

void **error_delete** (*Error* *error)
Destroy and free an error structure

3.9.3 Functions

void **error_set** (*Error* *error, int *code*, **const** char *msg)
Set error.

If an error is already present then do nothing. If you want to overwrite an already existing error then first call *error_clean*

void **error_clean** (*Error* *error)
Clean error.

Error code is set to 0 and the message to NO ERROR.

void **error_add** (*Error* *error, **const** char *msg)
Add a description message to the existing message and leaves as is the error code

const char ***error_message** (**const** *Error* *error)
Return error message if error, otherwise NULL

int **error_code** (**const** *Error* *error)
Return error code

3.10 TxnManager

3.10.1 Data structures

struct TxnManager
Transaction Manager.

LMDB has several restrictions in the operations it allows in multiple threads, but some of these restrictions must be imposed in the application code. In particular:

1. Some operations require that no transactions in the same process are active, for example mdb_env_set_mapsize
2. Some operations require that no write transactions are active. For example it is not documented, but it seems to happen that, mdb_env_info crashes if write transactions are active.

This structure tracks the number of read and write transactions active inside the process and allows blocking until all of them are aborted or commited.

Public Members

MDB_env ***env**
LMDB environment where transactions happen

InvSemaphore **txn_counter_read**
Counter of read transactions

InvSemaphore **txn_counter_write**
Counter of write transactions

struct InvSemaphore
Inverse Semaphore.

An inverse semaphore blocks when the count is greater than zero (a regular semaphore blocks when the count is at zero).

enum TxnManagerError
Values:

txn_manager_error_ok = 0
No error

txn_manager_error_internal
Unexpected error

txn_manager_error_memory
Error allocating new memory

txn_manager_error_thread
Error inside pthreads

txn_manager_error_mdb
Error inside LMDB

3.10.2 Constructor/Destructor

TxnManagerError **txn_manager_new**(*TxnManager* ***tm*, MDB_env **env*)
Allocate a new *TxnManager*

Return

0 if success, otherwise error code.

Parameters

- `tm` - The new transaction manager.
- `env` - The LMDB environment where transactions will be opened, aborted or commited.

TxnManagerError **txn_manager_delete**(*TxnManager* **tm*)
Destroy and free manager

3.10.3 Functions

The following functions are wrappers around the corresponding ones in LMDB. They will increment/decrement automatically the read and write transactions counters.

TxnManagerError **txn_manager_begin** (*TxnManager* **tm*, int *flags*, MDB_txn ***txn*)
Begin a new transaction.

Return

0 if success, otherwise error code.

Parameters

- `tm` -
- `flags` - The flags that you pass to LMDB's mdb_txn_begin. These flags will be checked for MDB_RDONLY to decide which transaction counter to increment. This operation will block if an environment resize is in progress.
- `txn` - New transaction.

TxnManagerError **txn_manager_commit** (*TxnManager* **tm*, MDB_txn **txn*)
Commit transaction.

The corresponding counter will be decremented

TxnManagerError **txn_manager_abort** (*TxnManager* **tm*, MDB_txn **txn*)
Abort transaction.

The corresponding counter will be decremented

The following function is the main reason for the existence of *TxnManager*.

TxnManagerError **txn_manager_expand** (*TxnManager* **tm*)
Check if the environment must be resized. If this is the case then resize it.

This call will block for sure until there are no write transactions active. This call may block until there are no read transactions active, only if a resize is necessary.

If a resize happens then creation of new read and write transactions will be blocked until it finishes.

MDB_MINIMUM_FREE_PAGES
Parameter associated to *txn_manager_expand*.

The mmap is resized when the remaining free space is less than this amount.

3.11 BFScheduler

3.11.1 Data structures

BF_SCHEDULER_DEFAULT_SIZE
Size of the mmap to store the schedule

BF_SCHEDULER_DEFAULT_PERSIST
Default value for *BFScheduler::persist*

struct BFScheduler
BestFirst scheduler.

As it name implies this scheduler follows a greedy strategy to decide which page is going to crawl next. It mains an ordered list of uncrawled pages. To decide the next page to be crawled this scheduler picks the highest score page and removes it from the top of the list.

The key is then to assign valid scores to the pages. If no scorer is selected this scheduler will use the score provided when the page is crawled. Additionally an alternative scorer can be set up, see for example *page_rank_scorer_setup* or *hits_scorer_setup*.

Public Members

PageDB ***page_db**
Page database

The page database is neither created nor destroyed by the scheduler. The rationale is that the scheduler can be changed while using the same PageDB. The schedule is "attached" to the PageDB.

Scorer ***scorer**
The scorer use to get page score.

If not set up, the *PageInfo.score* will be used

TxnManager ***txn_manager**
The scheduler state is maintained inside am LMDB environment

char ***path**
Path to the env

It is built by appending `_bfs` to the *PageDB::path*

int **persist**
If true, do not delete files after deleting object

float **max_soft_domain_crawl_rate**
Maximum crawls per second per domain

float **max_hard_domain_crawl_rate**
Maximum crawls per second per domain

enum BFSchedulerError
Values:

bf_scheduler_error_ok = 0
No error

bf_scheduler_error_memory
Error allocating memory

bf_scheduler_error_invalid_path
File system error

bf_scheduler_error_internal
Unexpected error

bf_scheduler_error_thread
Error inside the threading library

3.11.2 Constructor/Destructor

BFSchedulerError **bf_scheduler_new**(*BFScheduler* ***sch*, *PageDB* **db*)
Allocate memory and create a new scheduler

Return

0 if success, otherwise the error code

Parameters

- `sch` - Where to create it. `*sch` can be NULL in case of memory error
- `db` - *PageDB* to attach. Remember it will not be created nor destroyed by the scheduler

void **bf_scheduler_delete** (*BFScheduler* *sch)

Delete scheduler.

It may or may not delete associated disk files depending on the *BFScheduler::persist* flag

3.11.3 Input/Output

BFSchedulerError **bf_scheduler_add** (*BFScheduler* *sch, **const** *CrawledPage* *page)

Add a new crawled page

It will add the page also to the *PageDB*.

Return

0 if success, otherwise the error code

Parameters

- `sch` -
- `page` -

BFSchedulerError **bf_scheduler_request** (*BFScheduler* *sch, size_t *n_pages*, PageRequest **request)

Add a new crawled page

It will add the page also to the *PageDB*.

Return

0 if success, otherwise the error code

Parameters

- `sch` -
- `page` -

3.11.4 Update scores

BF_SCHEDULER_UPDATE_BATCH_SIZE

Size of the batch used in updating the schedule.

Updating the schedule involves starting a write transaction. However write transactions coming from multiple threads are serialized. Since adding new pages to the schedule and returning requests also start write transactions it means that the update thread could block this more critical operations. To avoid this we avoid long write transactions and split them in batches.

BF_SCHEDULER_UPDATE_NUM_PAGES

Don't update scores until this amount of new pages has arrived

BF_SCHEDULER_UPDATE_PER_PAGES

Don't update scores until this percentage of new pages has arrived

BFSchedulerError **bf_scheduler_update_start** (*BFScheduler* **sch*)
Start the update thread.

The update thread will run periodically the scorer, in case there is one, to recompute page scores.

BFSchedulerError **bf_scheduler_update_stop** (*BFScheduler* **sch*)
Stop the update thread

3.11.5 Settings

void **bf_scheduler_set_persist** (*BFScheduler* **sch*, int *value*)
Set persist option for scheduler

BF_SCHEDULER_CRAWL_RATE_STEPS
Number of steps to take between soft and hard crawl rate limit

BFSchedulerError **bf_scheduler_set_max_domain_crawl_rate** (*BFScheduler* **sch*, float *max_soft_crawl_rate*, float *max_hard_crawl_rate*)
Set *BFScheduler::max_soft_domain_crawl_rate* and *BFScheduler::max_hard_domain_crawl_rate*

3.12 Scorer

struct Scorer
Scorers are responsible of computing a measure between 0 and 1 of the relevance of a given page.

In order to be used in different schedulers they must obey the following interface.

Public Members

void ***state**
Scorer specific state

ScorerUpdateFunc ***update**
Update scorer

ScorerAddFunc ***add**
Add new page to scorer

ScorerGetFunc ***get**
Get a page score

typedef int(ScorerUpdateFunc)(void *state)
Scorer update function interface

typedef int(ScorerAddFunc)(void *state, const PageInfo *page_info, float *score)
Scorer add page function interface

typedef int(ScorerGetFunc)(void *state, size_t idx, float *score_old, float *score_new)
Scorer get page score function

To see concrete implementations have a look at `PageRankScorer` and `HitsScorer`.

3.13 PageRankScorer

3.13.1 Data structures

PAGE_RANK_SCORER_USE_CONTENT_SCORES
Default value for *PageRankScorer::use_content_scores*

PAGE_RANK_SCORER_PERSIST
Default value for *PageRankScorer::persist*

struct PageRankScorer

Public Members

PageRank ***page_rank**
Implementation of the *PageRank* algorithm

PageDB ***page_db**
Database with crawl information

Error ***error**
Error status

int **persist**
If true files will not be removed by *page_rank_scorer_delete*

int **use_content_scores**
If true use content scores inside PageRank algorithm

enum PageRankScorerError
Values:

page_rank_scorer_error_ok = 0
No error

page_rank_scorer_error_memory
Error allocating memory

page_rank_scorer_error_internal
Unexpected error

page_rank_scorer_error_precision
Could not achieve precision in maximum number of loops

3.13.2 Constructor/Destructor

PageRankScorerError **page_rank_scorer_new**(*PageRankScorer* ***prs*, *PageDB* **db*)
Create new scorer

PageRankScorerError **page_rank_scorer_delete**(*PageRankScorer* **prs*)
Delete scorer.

Files will be deleted unles *PageRankScorer::persist* is true

3.13.3 Functions

int **page_rank_scorer_add** (void **state*, **const** *PageInfo* **page_info*, float **score*)
Add new page to scorer.

Function signature complies with *Scorer::add*

int **page_rank_scorer_get** (void **state*, size_t *idx*, float **score_old*, float **score_new*)
Access *PageRank* scorer as with *page_rank_get*.

Function signature complies with *Scorer::get*

int **page_rank_scorer_update** (void **state*)
Update scores.

Function signature complies with *Scorer::update*

void **page_rank_scorer_setup** (*PageRankScorer* **prs*, *Scorer* **scorer*)
Given a Scorer fill its fields with the necessary info

3.14 Settings

void **page_rank_scorer_set_persist** (*PageRankScorer* **prs*, int *value*)
Sets *PageRankScorer::persist*

void **page_rank_scorer_set_use_content_scores** (*PageRankScorer* **prs*, int *value*)
Sets *PageRankScorer::use_content_scores*

void **page_rank_scorer_set_damping** (*PageRankScorer* **prs*, float *value*)
Sets PageRankScorer::page_rank::damping

3.15 HitsScorer

3.15.1 Data structures

HITS_SCORER_USE_CONTENT_SCORES
Default value for *HitsScorer::use_content_scores*

HITS_SCORER_PERSIST
Default value for *HitsScorer::persist*

struct HitsScorer

Public Members

Hits ***hits**
Implementation of the HITS algorithm

PageDB ***page_db**
Database with crawl information

Error ***error**
Error status

int **persist**
If true files will not be removed by *page_rank_scorer_delete*

int **use_content_scores**
If true use content scores inside PageRank algorithm

enum HitsScorerError
Values:

hits_scorer_error_ok = 0
No error

hits_scorer_error_memory
Error allocating memory

hits_scorer_error_internal
Unexpected error

hits_scorer_error_precision
Could not achieve precision in maximum number of loops

3.15.2 Constructor/Destructor

HitsScorerError **hits_scorer_new** (*HitsScorer* ***hs*, *PageDB* **db*)
Create new scorer

HitsScorerError **hits_scorer_delete** (*HitsScorer* **hs*)
Delete scorer.

Files will be deleted unles *HitsScorer::persist* is true

3.15.3 Functions

int **hits_scorer_add** (void **state*, **const** *PageInfo* **page_info*, float **score*)
Add new page to scorer.

Function signature complies with *Scorer::add*

int **hits_scorer_get** (void **state*, size_t *idx*, float **score_old*, float **score_new*)
Access HITS scorer as with *hits_get_authority*.

Function signature complies with *Scorer::get*

int **hits_scorer_update** (void **state*)
Update scores.

Function signature complies with *Scorer::update*

void **hits_scorer_setup** (*HitsScorer* **hs*, *Scorer* **scorer*)
Given a Scorer fill its fields with the necessary info

3.16 Settings

void **hits_scorer_set_persist** (*HitsScorer* **hs*, int *value*)
Sets *HitsScorer::persist*

void **hits_scorer_set_use_content_scores** (*HitsScorer* **hs*, int *value*)
Sets *HitsScorer::use_content_scores*

3.17 PageRank

3.17.1 Data structures

PAGE_RANK_DEFAULT_DAMPING
Default *PageRank::damping*

PAGE_RANK_DEFAULT_MAX_LOOPS
Default *PageRank::max_loops*

PAGE_RANK_DEFAULT_PRECISION
Default *PageRank::precision*

PAGE_RANK_DEFAULT_PERSIST
Default *PageRank::persist*

struct PageRank
Implementation of the *PageRank* algorithm.

See for example Wikipedia.

Additionally, it allows to merge the pure link based original algorithm with page content scores.

Public Members

MMapArray ***out_degree**
Number of outgoing links.

If page content scores are used then this array is actually the aggregated scores of all the outgoing links.

MMapArray ***value1**
PageRank value, old iteration

MMapArray ***value2**
PageRank value, new iteration

size_t **n_pages**
Number of pages

char ***path_out_degree**
Path to the out degree mmap array file

char ***path_pr**
Path to page rank mmap array file

Error ***error**
Error status

float **damping**
Probability of making a random page jump: 1.0 - damping

MMapArray ***scores**
External computed scores associated with the pages

float **total_score**
Total score

size_t **max_loops**
If greater than 0 stop computation even if precision was not achieved

float **precision**
Stop iteration when the the largest change in any page score is below this threshold

int **persist**
If true, do not delete files after deleting

enum PageRankError
Values:

page_rank_error_ok = 0
No error

page_rank_error_memory
Error allocating memory

page_rank_error_internal
Unexpected error

page_rank_error_precision
Could not achieve precision in maximum number of loops

3.17.2 Constructor/Destructor

PageRankError **page_rank_new** (*PageRank* ***pr*, **const** char **path*, size_t *max_vertices*)
Create a new structure.

Return

0 if success, otherwise an error code.

Parameters

- `pr` - The new structure is returned here. NULL if memory error.
- `path` - Directory where all files will be stored.
- `max_vertices` - Initial hint of the number of pages.

PageRankError **page_rank_delete** (*PageRank* **pr*)
Free memory and close associated resources.

Files will be deleted or not depending on the value of *PageRank::persist*.

3.17.3 Functions

PageRankError **page_rank_set_n_pages** (*PageRank* **pr*, size_t *n_pages*)
Reserve memory for the specified number of pages

PageRankError **page_rank_compute** (*PageRank* **pr*, void **link_stream_state*, LinkStreamNextFunc **link_stream_next*, LinkStreamResetFunc **link_stream_reset*)
Compute *PageRank* score for all pages.

The algorithm makes random access of pages scores and sequential access of the links.

Return

0 if success, otherwise an error code.

Parameters

- `pr` -

- `link_stream_state` - For example PageDBLinkStream
- `link_stream_next` - For example *page_db_link_stream_next*
- `link_stream_reset` - For example *page_db_link_stream_reset*

PageRankError **page_rank_get** (**const** *PageRank* *pr, size_t *idx*, float *score_old*, float *score_new*)
Get *PageRank* score associated to a given page.

Return

0 if success, otherwise an error code.

Parameters

- `pr` -
- `idx` - Page index.
- `score_old` - Score on the previous call to *page_rank_compute*.
- `score_new` - Score on the last call to *page_rank_compute*.

void **page_rank_set_persist** (*PageRank* *pr, int *value*)
Set value of *PageRank::persist*

3.18 Hits

3.18.1 Data structures

HITS_DEFAULT_MAX_LOOPS
Default *Hits::max_loops*

HITS_DEFAULT_PRECISION
Default *Hits::precision*

HITS_DEFAULT_PERSIST
Default *Hits::persist*

struct Hits
Implementation of the HITS algorithm.

See for example Wikipedia.

Additionally, it allows to merge the pure link based original algorithm with page content scores. The idea is that the authority scores are distributed back to the hub according to the content score. For example imagine that page A links to B, C and D and the content/authority scores are:

-B: 0.5 / 0.1 -C: 0.1 / 1.0 -D: 0.9 / 0.5

Then the hub score of A would be computed as:

Hub(A) = 0.5*0.1 + 0.1*1.0 + 0.9*0.5

Public Members

MMapArray ***h1**
Hub score, previous iteration

MMapArray ***h2**
Hub score, current iteration

MMapArray ***a1**
Authority score, previous iteration

MMapArray ***a2**
Authority score, current iteration

char ***path_h1**
Path to mmap file of *Hits::h1*

char ***path_h2**
Path to mmap file of *Hits::h2*

size_t **n_pages**
Number of pages

Error ***error**
Error status

MMapArray ***scores**
External computed scores associated with the pages

size_t **max_loops**
If greater than 0 stop computation even if precision was not achieved

float **precision**
Stop iteration when the the largest change in any page score is below this threshold

int **persist**
If true, do not delete files after deleting object

enum HitsError
Values:

hits_error_ok = 0
No error

hits_error_memory
Error allocating memory

hits_error_internal
Unexpected error

hits_error_precision
Could not achieve precision in maximum number of loops

3.18.2 Constructor/Destructor

HitsError **hits_new** (*Hits* ***hits*, **const** char **path*, size_t *max_vertices*)
Create a new structure.

Return

0 if success, otherwise an error code.

Parameters

- `pr` - The new structure is returned here. NULL if memory error.
- `path` - Directory where all files will be stored.

- `max_vertices` - Initial hint of the number of pages.

HitsError **hits_delete** (*Hits *hits*)
Free memory and close associated resources.

Files will be deleted or not depending on the value of *Hits::persist*.

3.18.3 Functions

HitsError **hits_set_n_pages** (*Hits *hits*, size_t *n_pages*)
Reserve memory for the specified number of pages

HitsError **hits_compute** (*Hits *hits*, void *link_stream_state*, LinkStreamNextFunc *link_stream_next*, LinkStreamResetFunc *link_stream_reset*)
Compute HITS score for all pages.

The algorithm makes random access of pages scores and sequential access of the links.

Return

0 if success, otherwise an error code.

Parameters

- `pr` -
- `link_stream_state` - For example PageDBLinkStream
- `link_stream_next` - For example *page_db_link_stream_next*
- `link_stream_reset` - For example *page_db_link_stream_reset*

HitsError **hits_get_hub** (**const** *Hits* *pr*, size_t *idx*, float *score_old*, float *score_new*)
Get hub score associated to a given page.

Return

0 if success, otherwise an error code.

Parameters

- `pr` -
- `idx` - Page index.
- `score_old` - Score on the previous call to *hits_compute*.
- `score_new` - Score on the last call to *hits_compute*.

HitsError **hits_get_authority** (**const** *Hits* *pr*, size_t *idx*, float *score_old*, float *score_new*)
Get authority score associated to a given page.

Return

0 if success, otherwise an error code.

Parameters

- `pr` -
- `idx` - Page index.
- `score_old` - Score on the previous call to *hits_compute*.

- `score_new` - Score on the last call to *hits_compute*.

void **hits_set_persist** (*Hits* *hits, int *value*)
Set value of *Hits::persist*

3.19 MMapArray

3.19.1 Data structures

struct MMapArray
A memory mapped array

Public Members

char ***mem**
Pointer to data

int **fd**
File descriptor for data

char ***path**
Path to data file

size_t **n_elements**
Number of elements

size_t **element_size**
Size of each element

int **persist**
If true, do not delete files after deleting object

enum MMapArrayError
Values:

mmap_array_error_ok = 0
No error

mmap_array_error_memory
Error allocation memory

mmap_array_error_internal
Unexpected error

mmap_array_error_mmap
Error with a mmap operation (creation, unmapping, advise...)

mmap_array_error_file
Error manipulating the file system

mmap_array_error_out_of_bounds
Tried to access array past boundaries

3.19.2 Constructor/Destructor

MMapArrayError **mmap_array_new** (*MMapArray* ***marr*, **const** char **path*, size_t *n_elements*, size_t *element_size*)

Create a new *MMapArray*

Return

0 if success, otherwise the error code (also available in marr if not NULL)

Parameters

- `marr` - Will be changed to point to the newly allocated structure, or NULL if failure
- `path` - Path to the associated file. Can be NULL in which case the mapping is made anonymous.
- `n_elements` - Number of elements (can be changed later with *mmap_array_resize*)
- `element_size` - Number of bytes of each element

MMapArrayError **mmap_array_delete** (*MMapArray* **marr*)

Delete *MMapArray*

If the structure cannot be deleted, the memory will not be freed

Return

0 if success, otherwise the error code (also available in marr)

3.19.3 Functions

MMapArrayError **mmap_array_advise** (*MMapArray* **marr*, int *flag*)

Advise memory use pattern

It accepts any flag that madvise accepts

Return

0 if success, otherwise the error code (also available in marr)

MMapArrayError **mmap_array_sync** (*MMapArray* **marr*, int *flag*)

Force memory-disk syncronization

It accepts any flag that msync accepts

Return

0 if success, otherwise the error code (also available in marr)

void ***mmap_array_idx** (*MMapArray* **marr*, size_t *n*)

Returns pointer to the array element

Return

In case of failure it will return NULL. The error code is available in marr

MMapArrayError **mmap_array_set** (*MMapArray* **marr*, size_t *n*, **const** void **x*)

Set array element value

Return

0 if success, otherwise the error code (also available in marr)

void **mmap_array_zero** (*MMapArray* *marr)
Set all elements of array to zero

MMapArrayError **mmap_array_resize** (*MMapArray* *marr, size_t n_elements)
Change number of elements

The new memort is initialized to 0

Return

0 if success, otherwise the error code (also available in marr)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

B

C

D

E

H

I

L

M

P

S

T