Smart Pipe Documentation

Release 0.1

Max Lapan

November 28, 2016

Contents

Small and fast storage container for key-value entries, optimized for sequential data access, but allowing fast individual entry lookup.

Contents:

CHAPTER 1

Overview

Quite frequently it is required to save large amount of objects for later sequential process. For example: list of downloaded files, entries queried from DB, log entries, etc.

Such storage can be organized in various ways, like:

1. dump all entries in one file,
2. create tar archive,
3. save in sqlite db, etc.

Unfortunately, if your access to data is expected to be sequential, those options are not exactly what is required for fast access and efficient on-disk storage.

1.1 Quick example

But let's look at small example.

```
import smart_pipe as sp

# save some data
writer = sp.SmartPipeWriter("my_data", compress=True)
writer.checkpoint(b'block-1')
writer.append(b'info', b'value')
writer.append(b'other-info', b'value2')
writer.checkpoint(b'block-2')
writer.append(b'info', b'value for block 2')
writer.append(b'error', b'failed, dude')
writer.close()

# read data back
reader = sp.SmartPipeReader("my_data")
while True:
    block_key = reader.get_next_block_key()
    if block_key is None:
        break
    print("Processing block: %s" % (str(block_key)))
    for key, value in reader.pull_block():
        print("  key: %s, value len %d" % (str(key), len(value)))
reader.close()
```

Result of this code will be two files **my_data.datz** with actual data and **my_data.idx** with block index. Code output:

```
Processing block: block-1
  key: info, value len 5
  key: other-info, value len 6
Processing block: block-2
  key: info, value len 17
  key: error, value len 12
```

CHAPTER 2

Architecture of SmartPipe

It's very simple and straightforward. There are two angles architecture can be described: logical and physical.

- Logical is more about **what** you can store and do with this storage class,
- physical is about **how** data is stored and what to expect from it.

2.1 Logical model

2.1.1 Blocks

Logically, smart pipe is a sequence of blocks. Every block block has key, which has to be uniq byte sequence. Like this:

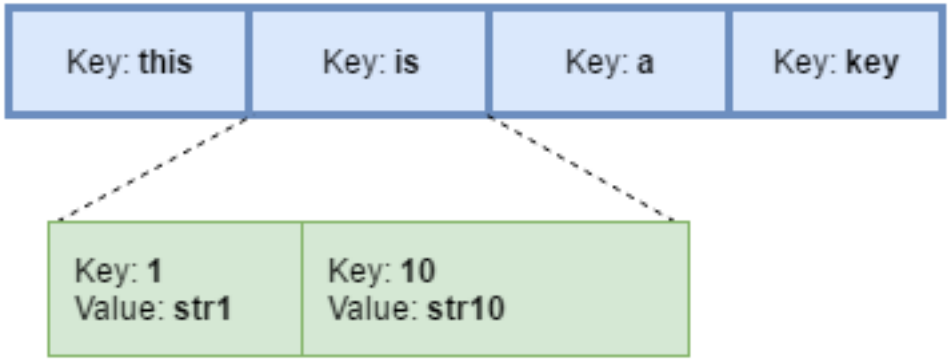
We don't require keys to be numbers or strings of some form or ordered, etc. They just need to be uniq, that's it.

We remember position of every block in separate index, so, every individual block can be accessed by it's key. Or you can read all blocks sequentially, like file.

Every block should have reasonable size – during reading, all it's contents are read into memory, so, several megabytes will be fine, but couple of gigabytes can be too much. But library never keeps several blocks in memory at the same time, so, with modern memory prices it's not too strict (otherwise, why are you using Python?).

2.1.2 Block structure: key-values

Inside every block can contain any amount of key-value pairs which both are just byte sequences.



Key and values have even less limitations than block keys – they can be uniq, empty, etc. So, technically, only one limitation: they have to fit in your memory.

Access pattern is simple: you need to iterate all key-value pairs of the block to go to next block's data. You can't efficiently seek inside the block, so, keep this in mind designing your app.

2.1.3 Usage example

This library was designed to efficiently save and re-process results obtained from the web, but can be applied to other areas. Our particular example is:

1. Every block is a site we've processed, addressed by our uniq internal id (or url)
2. Inside the block we can have various key-value pairs:

1. raw html result from site (with 'raw' key),
2. server's response text ('response' key),
3. processing results in json form ('result' key)
4. log information from processing ('log' key)

After data was downloaded and processed, smart pipe file are almost immutable (in theory, it won't be too hard to append but currently it's not implemented), but can be efficiently read sequentially or you can efficiently read individual blocks, which is mostly useful for debugging specific problematic cases.

2.2 Physical model

There are two files per smart pipe:

1. with data blocks, can have extension .dat (for uncompressed smart pipes) or .datz (for compressed)
2. with index, containing index of every blocks' key and offset withing data file.

Index is just binary file with raw keys of blocks and offsets and this file is read completely into memory on smart pipe reader creation. So, please, don't try to store trillions of blocks inside single pipe!

Data blocks have slightly more complicated structure, but also simple: every block has keys and values in serialized form, and whole block's data is optionally compressed by zlib library.

CHAPTER 3

Tutorial

TODO

CHAPTER 4

Tools

There are special command-line tools, which can help access smart pipe data files, look inside them, analize their performance etc.

All of them are inside single python program called **sp.py**, which is added to path automatically on installation of package (if you're installing smart_pipe package inside VE, you need to activate VE first).

There are four tools available at the moment:

1. **ls** – list block indices and block's contents
2. **cat** – retrieve individual block's key/value pair
3. **cat_all** – list all entries with optional regexp applied to key
4. **check** – check smart pipe consistency and performance by reading data sequentially and randomly

All tools are available as sp.py subcommand, for example:

```
$ sp.py ls ~/work/data/2016_11_17_231501/visual_annotations/ve
1
  1:room_container_Tab2_after_click 266937
  1:room_container_Tab3_before_click 158908
  1:whole_page 1157479
  1:room_container_Tab1_before_click 304945
  1:room_container_Tab0_before_click 125256
  1:room_container_Tab2_after_reprocess 254864
  1:room_container_Tab0_after_reprocess 125723
  1:room_container_Tab2_before_click 253828
  1:room_container_Tab3_after_click 167145
  1:room_container_Tab1_after_click 318049
  1:room_container_Tab0_after_click 131816
  1:shop_summary 75227
  1:room_container_Tab3_after_reprocess 159497
  1:room_container_Tab1_after_reprocess 305962
2
  2:whole_page 262741
  2:room_container_before_click 16203
  2:shop_summary 6724
--- output truncated ---
```

CHAPTER 5

API Docs

class smart_pipe.**SmartPipeWriter**(*path_prefix*, *compress=False*)
Class which can create smart pipe file

__init__(*path_prefix*, *compress=False*)
Constructs smart pipe writer

Parameters

- **path_prefix** – path prefix to be used. Extension for data and index files will be appended to this prefix
- **compress** – boolean flag (False by default) indicating compression of data chunks

append(*key*, *value*)
Appends given key-value pair to current chunk

Parameters

- **key** – byte string with key
- **value** – byte string with value

checkpoint(*key*)
Flushes current chunk on disk and starts new with the given key

Parameters **key** – byte string with binary key

close()
Closes smart pipe writer and flushes data to disk

class smart_pipe.**SmartPipeReader**(*path_prefix*)
Reader for smart pipe data

__init__(*path_prefix*)
Constructs smart pipe reader

Parameters **path_prefix** – path prefix for smart pipe data and index files

close()
Closes smart pipe reader

get_next_block_key()
Return index key for the next block

Returns key of the next block or None if we reached end of pipe

pull_block(*index_key=None*)
Get list of key,value pairs from next block

Parameters **index_key** – optional key of block to seek

Returns yield block records

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Symbols

A

C

G

P

S