Enhancing SWH Object Storage with Compressed and Dynamic Solutions

P. Ferragina, F. Tosoni

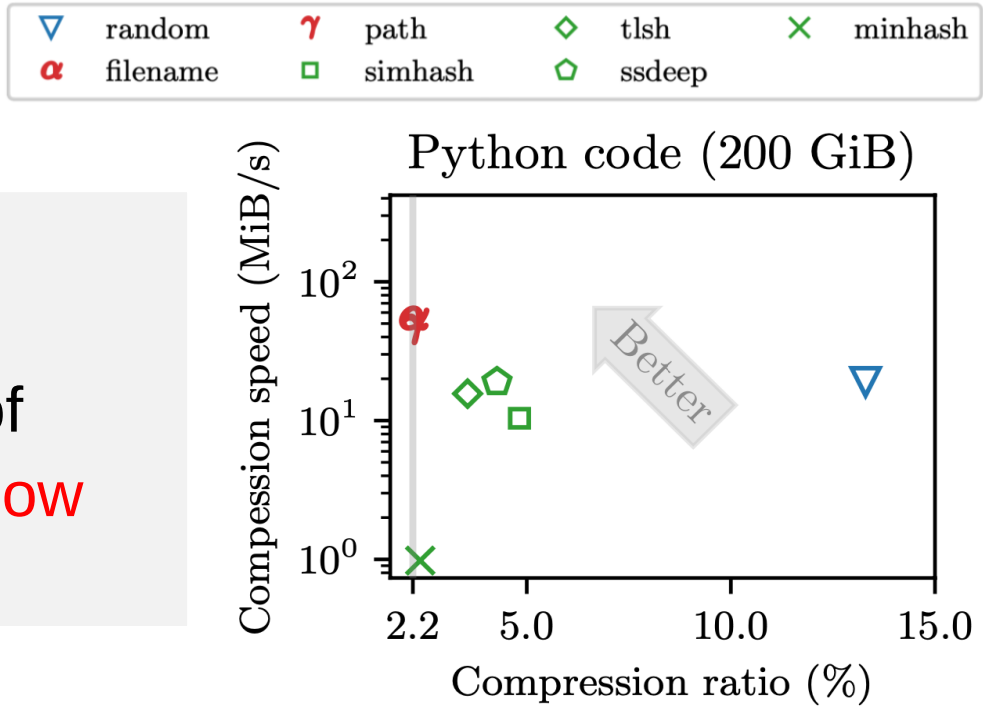
A3Lab
Department of Excellence 2023 - 2027
LEMbeDS
Economics, Management and Law in the era of Data Science
IN SUPREMÆ DIGNITATIS 1343

Previous work (1/2)

Permute-Partition-Compress (PPC) paradigm

Backup scenario

- **Permute** files by similarity
- **Partition** data into blocks of a specific size
- **Compress** the blocks



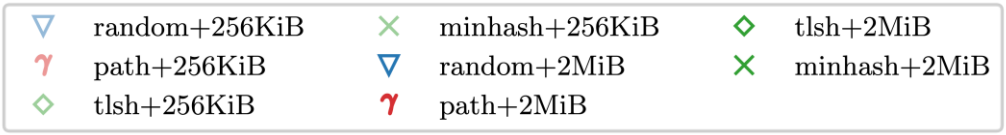
⇒ **Outcome**: very good compression ratios (up to 2.2% on 200 GBs of Python code) but static, slow construction

Previous work (2/2)

Permute-Partition-Compress (PPC) paradigm

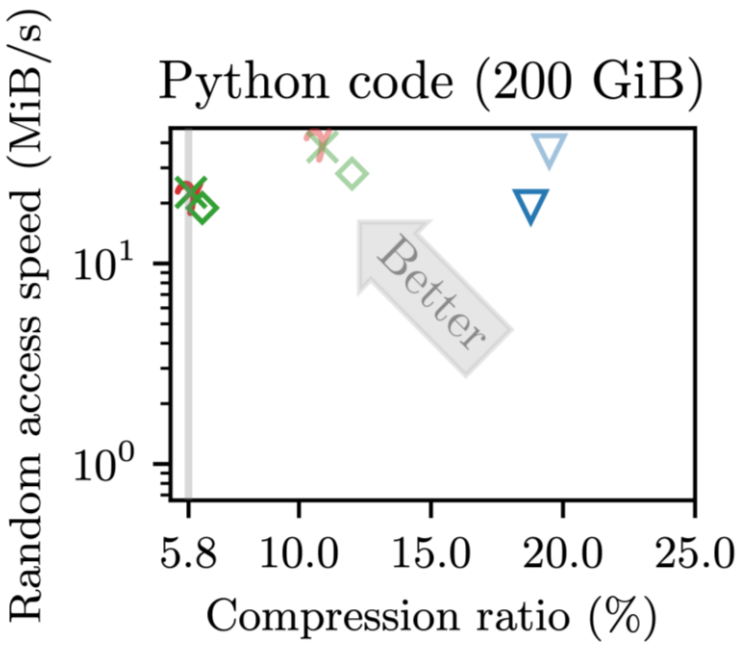
Access scenario

- Backup capabilities
- Manage blocks with RocksDB?



RocksDB Meta

⇒ **Outcome**: slightly worse compression ratios (up to 5.8%) and moderate access speed (~10^1 MBs^{-1})



RocksDB

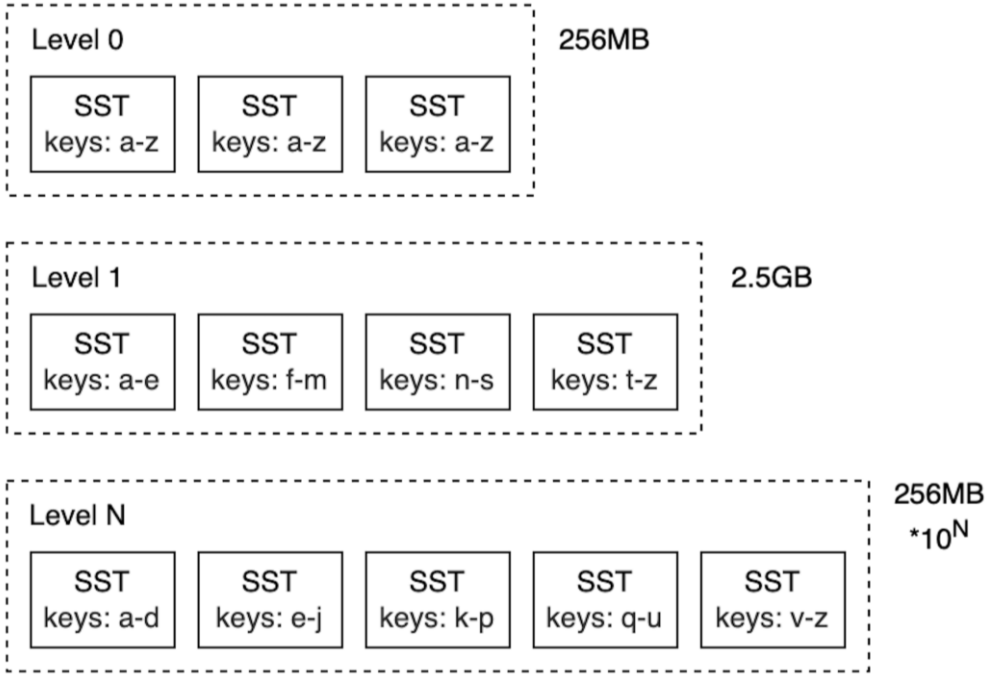
RocksDB

Meta

A **key-value store** based upon the **LSM-Tree**

PPC approach:

- Ordered by key
  → permute
- Divided into blocks
  → partition
- Compressed blocks
  → compress



Why RocksDB?

- **Good compression** ratio
- **Very fast** in insertion and retrieval
- Multiple retrieval options:
  - Single-get
  - **Multi-get**
- Fine-tunable with many options

Officially distributed in **C++** & **Java**; wrappers in Python3, Rust & other languages

Tuning RocksDB

Federico Ramacciotti's MSc thesis

Pairs = <fingerprint* of the file, file>

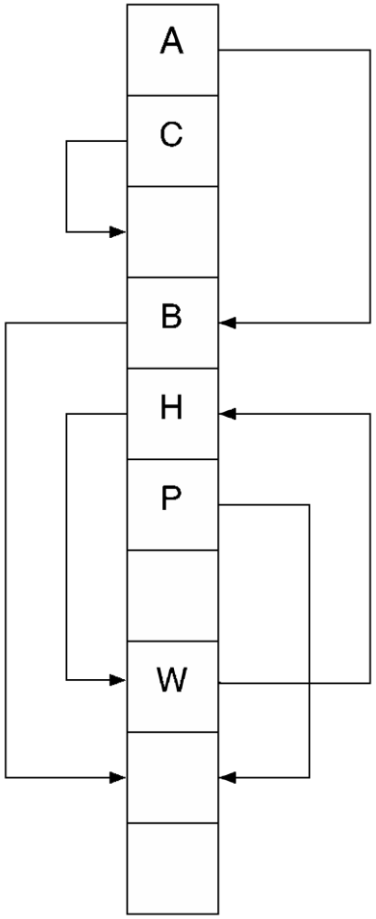
(*) Fingerprints = filename, tlsh, min_hash, …

Three scenarios:

- **Backup**: just storage and full decompression (sort the pairs, create a `Parquet` file and compress with `zstd-22`)
- **Access**: support random access to files (sort the pairs, insert them in RocksDB, and compress with `zlib-6`) → multiget
- **Dynamic**: key-value pairs arriving in streaming (inserted as is in RocksDB and compressed with `zlib-6`)

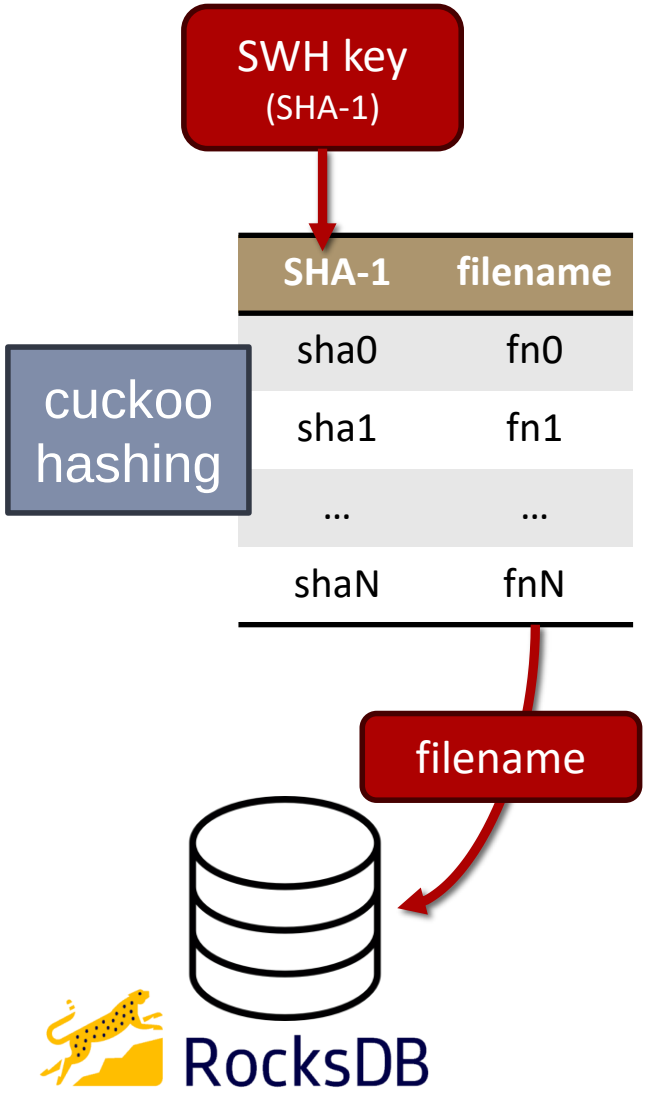
→ multiget: ~10^1 MB s^{-1}

RocksDB for SWH's Object Storage (1/2)



Cuckoo hashing (`libcuckoo`) for a *space-efficient* and *dynamic* mapping SHA-1→filenames.

We might consider replacing cuckoo hashing with an additional RocksDB instance as the number of entries grows.



RocksDB for SWH's Object Storage (2/2)

- **Python Integration** via `pybind11` wrappers **of C++** components (RocksDB, libcuckoo).
- **Apache Arrow** to improve data serialisation and processing (support for **Parquet** file format).

Outcome: A new fast and dynamic object-storage layer for files ≤5KB (which make up most of SWH's space)

Conclusions

Simple, modular and general framework

How to design an **object-storage backend**?
RocksDB offers...

- Dynamic and effective solution
- Good compression (~20%)
- Fast insertion (400 MB s^{-1}, 10^5 files s^{-1}, zlib-6) and deletion
- Fast access (10 MB s^{-1}, 2500 files s^{-1}, zlib-6)

How to distribute **datasets for AI**?
PPC approach offers...

- Output based on the parquet format
- Content- and context-based compression
- Open-source codebase → reproducible

Efficient storage for **Code2Code search engines**