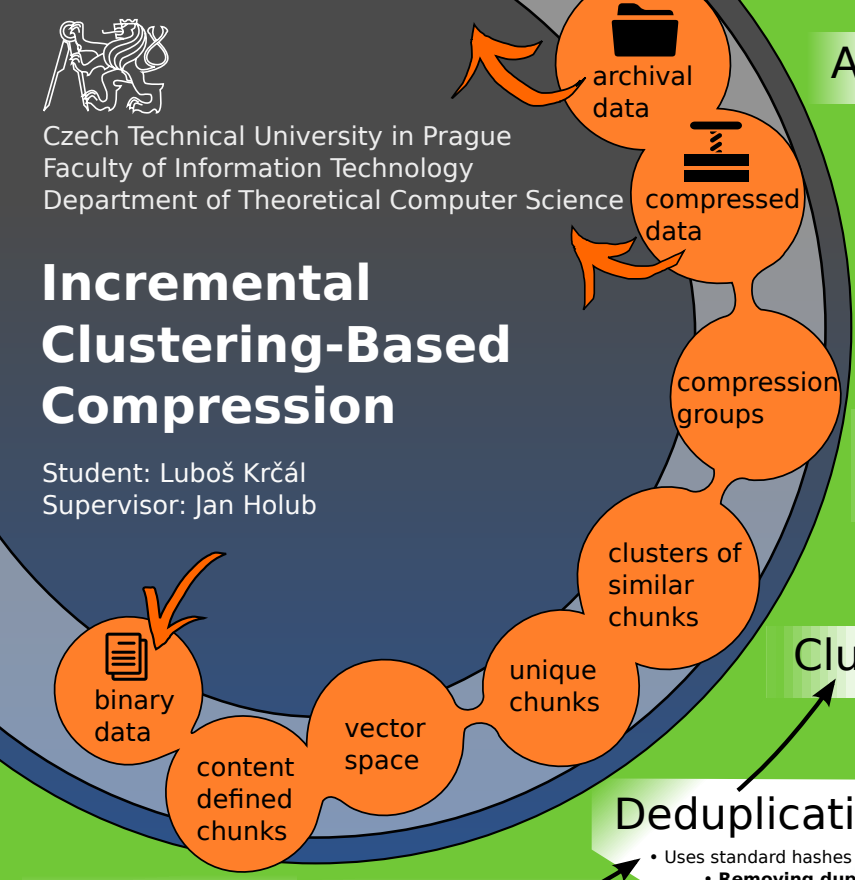
Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Incremental Clustering-Based Compression

Student: Luboš Krčál
Supervisor: Jan Holub



Archiver

- It is possible to store more information about the deduplication, simhashing, clustering and grouping - the **archivedata**
- This allows for conveniet **CRUD** operations over files in the archive

Example of metadata implementation:

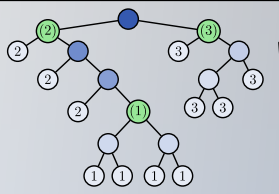
```
// magic number
0000: 0000 0123

// number of compression
// groups and chunks
0004: 0000 1C28 002D 40AC

// original data size
0014: 0000 0000 06C1 229F

// offests of the 1. group to
// compressed and orig data
001C: 0003 DBB2 0013 6002
      .... .... .... ....
// offset of the 1. chunk
1C3C: 00E3 2421
      .... ....
```

Compression

- Compression groups are **compressed using standard compressors**, such as Deflate, BZip2, etc.
- **Metadata** is then added, representing all the chunks and compression groups

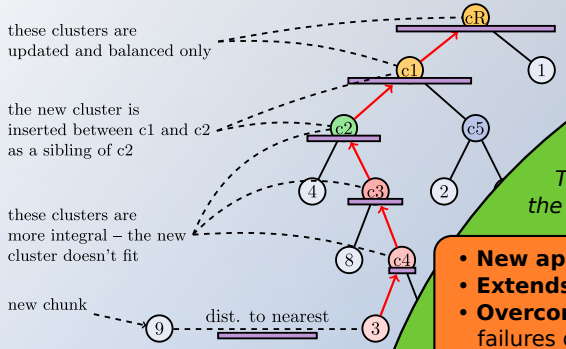
Grouping & Reordering

- Clustering is gradually processed and **compression groups** are determined
- Compression groups are represented with a single cluster



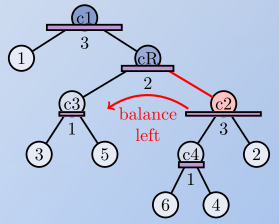
Clustering

- **Hierarchical clustering** – binary tree, leaves represent chunks
- **Bottom-up** – new node inserted from the bottom
- **Randomized KD-trees** used to find the nearest neighbor
- **Simhash calculated for parent nodes** – as linear combination of children
- Preserves heap property on inter-cluster distances (with simhash distances)



- Optional extra balancing (in scenarios where heap property is not inherent)
- Optional deep distance (use multiple levels of descendants)
- Optional representatives (use only selected descendants)

Disbalanced clusutering:

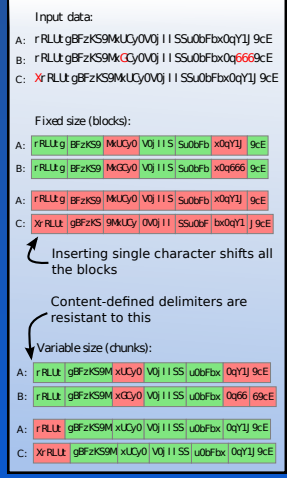


Deduplication

- Uses standard hashes to find duplicate chunks
- **Removing duplicate chunks** is more effective than compressing then together
- Most effective for small chunks

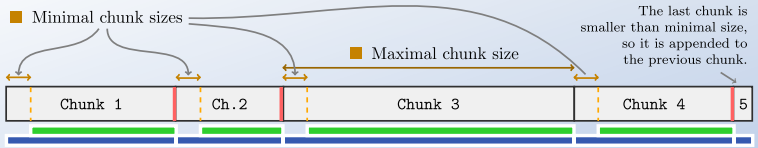
Chunking

- Binary data is split into **content-defined chunks**
- Resistant to character inserts or shifts
- Uses **partial pattern matches** to mark delimiters



Similarity Hashing

- **Features**: **n-gram**, compression features, etc. Every feature is hashed
- Standard simhash: merges feature hashes based on majority value per hash position (vector over Z_2 , same length as feature hashes)
- **Extended simhash**: feature hashes are combined together into a vector and normalized (vector over $\mathbf{Z_N}$ of arbitrary length)
- Vector norms are used as distance measures



- Content breakpoint defined by rabin fingerprint
- Rabin fingerprint computation range
- Chunk hash computation range

Individual 3-gram hashes are computed. These are then combined into a vector.

Example of binary simhash for the string "SWISS MISS".

Note that the length of the simhash determines the dimension of the vector space and the width determines the cardinality of the space.

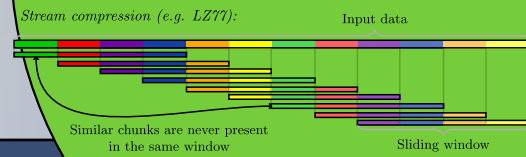
```
hash("ISS") = 11101000110110100000101111011101
hash("SWI") = 10000101000111111001001100010111
hash("WIS") = 11000011101000011100100100101110
hash("SS ") = 10000101000111111001001100010111
hash("S M") = 10000101000111111001001100010111
hash(" MI") = 00001011111011101001011101000111
hash("MIS") = 01011000101000111011111000001011

simhash("SWISS MISS) = 10000001000111111000101100010111
```

Problem

What if the following solutions are not good enough?

- **Solid-compression (tar.*)**
  How do you decompress a single file from the archive?
- **File-by-file compression (zip)**
  What happens if you compress similar files?
- **Deduplication**
  Only finds exact duplicates
  Several near-deduplication systems exist (only very similar blocks are then compressed)
- **Compression file systems**
  Combination of deduplication and file-by-file compression

And, the standard compression techniques fail on data with distant similarities?



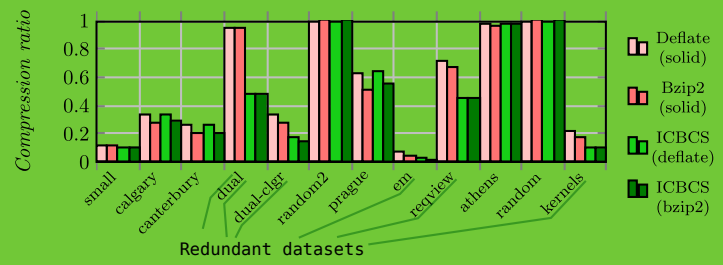
We want to achieve an excellent and efficient compression over a large dataset. With the possibility to read, add, modify and remove data conveniently.

Results

One of the major contributions was to design a compression and archival system based on clustering.

This was successfully completed with the following results:

- **New approach** to compression
- **Extends** current near-deduplication systems
- **Overcomes locality-based** redundancy removal failures of standard compression algorithms
- Has the potential of being an **ultimate archiver**



- Compression ratio significantly improved for very redundant datasets
- For non-redundant dataset, the ratio was almost the same. This is a success, since the files were now compressed in very small batches
- Using weaker compression on appropriately preprocessed data results in both faster and better compression than using a strong compressor as it is