# Compressed-memory Mapped-vectors Using Traditional and Deep Neural Network Compression Algorithms

**Themis Palpanas**
***Senior Member of Institut Universitaire de France (IUF)***
***University of Paris***
***themis.palpanas@u-paris.fr***

**Kostas Zoumpatianos**
***Harvard University***
***kostas@seas.harvard.edu***

**[Motivation]** Given the large amounts of data that modern organizations need to store and process, compression has emerged as one of the fundamental ways used to reduce storage overheads. It is a crucial part of fields as diverse as networking, data processing and databases, as well as machine learning. There exist various open source libraries that implement different kinds of algorithms, including **LZW** [1], **Delta** and **Delta-delta encoding** [2], **XOR based encoding** [2], **Snappy** [3], as well as novel **deep neural network** based compression algorithms like **Bit-Swap** [4]. Nevertheless, there is no easy to use **STL vector replacement for C++**, which transparently exposes them to the user.

**[Goal]** To fill this gap, in this project we will develop a **thread-safe, STL compatible vector implementation** for **C++**, backed in tertiary storage through **memory mapped files** [5]. This will allow applications to scale to multi-TB compressed datasets, transparently to the user. It will be implemented as a templated library which will support multiple C++ primitive types and compression methods.

[**Challenges]** The main challenges of this project are around **optimally scheduling compression and decompression operations** of **variable-length compressed segments**. Specifically, each time that an element of the vector is accessed, the relevant compressed chunk(s) should be retrieved from the memory mapped file, decompressed and buffered, given a certain memory budget. Because of the nature of memory mapped files, compressed chunks should have a fixed size, which in the presence of updates, could be required to expand to more than one chunk. This can cause the array to be fragmented. In order to navigate such fragmented arrays, the right meta-data has to be maintained. Since each range of locations can now be spread in different locations of the disk, I/Os should be scheduled accordingly while also utilizing pre-fetching. Specifically, given a vector with locations [0, n], this range can be non-uniformly partitioned in **segments** of non-equal size, which nevertheless are mapped into **compressed chunks** of equal size. Each segment can be compressed using one or more compressed segments, which are connected to each-other through pointers.

**[Experiments and Applications]** In this project we will wrap this STL vector in a **column store** [6] implementation (e.g., Apache Cassandra/HBase, MonetDB), and use it to provide, **put, get, range search** and **sort** operations. We will study the performance of the above methods, and provide optimizations to render them efficient on compressed data. Such optimizations might include **zone-map indexes** (uncompressed synopses for each compressed segment that allow us to skip segments that do not contain our answer), buffering as well as various other techniques that can optimize sorting.

**[Methodology]** In the first part of the project, we will develop a **compressed_vector<T>** interface, which will internally use the STL vector to store compressed chunks in main memory. In the second part, we will extend this to utilize **memory maps** and make sure that we can operate on larger-than-memory datasets. Finally, in the last part, we will implement a basic **column store** using this vector and evaluate its performance under different optimizations.

**[Prerequisites]** Experience with file and data structures, excellent programming skills in C/C++.

**[Internship]** Apply by emailing Prof. Themis Palpanas your detailed CV (including course marks). Accepting this project will make you part of diNo (LIPADE, Paris Descartes University), an enthusiastic team working on real, challenging problems! The internship will last between 3-6 months, and is fully funded.

**References**

**[1]** https://github.com/vapier/liblzw
**[2]** https://github.com/facebookarchive/beringei
**[3]** https://github.com/google/snappy
**[4]** https://github.com/fhkingma/bitswap
**[5]** https://en.wikipedia.org/wiki/Memory-mapped_file
**[6]** https://en.wikipedia.org/wiki/Column-oriented_DBMS