UE Software Engineering

Software Engineering Project 2025

compressing for speed up transmission

Prepared By:

Abdelbaki Kacem

2025

Chapter 1

Integer Array Compression by Bit Packing

Plan

1 Introduction 2

2 Project Context 2

3 Target Problem 2

4 Proposed Solution 2

5 Methodology 3

1.1 Introduction

This report introduces the general context of the project, including an analysis of the current state, the problem statement, the proposed solution, and the chosen methodology to plan and execute the project.

1.2 Project Context

The project consists of designing and implementing a compression system for integer arrays. The main objective is to reduce the size of integer arrays for more efficient transmission over a network, while preserving the ability to quickly access any element of the compressed array.

1.3 Target Problem

The raw transmission of integer arrays (typically 32 bits per integer) can be very inefficient, especially when the actual values contained in the array can be represented with far fewer bits. The challenge is to find a compression method that:

- Reduces the total number of bits to be transmitted.
- Allows fast direct access (get(i) function) to any element without having to decompress the entire array.
- Efficiently handles arrays containing integers of variable bit widths, particularly when a few large values might impose an inefficient compression width for all others.

1.4 Proposed Solution

The implemented solution includes three distinct compression strategies based on bit packing:

- **No Spill :** Packs integers into 32-bit blocks without allowing a single integer to cross a block boundary.
- **Spill :** Packs integers sequentially into a binary stream, allowing a single integer to cross 32-bit block boundaries for better density.
- **Overflow :** Identifies a bit threshold. Integers exceeding this threshold are stored in a separate list. The main array contains compact indicators pointing to this overflow list.

Each strategy is accompanied by functions to compress, decompress, and directly access elements. A dynamic thresholding method is implemented to optimize the overflow strategy. Performance is measured, and a bandwidth threshold is calculated to determine when compression becomes beneficial.

1.5 Methodology

The project was designed and implemented in Java. The code structure follows a modular approach, with distinct functions for each compression strategy and its associated operations (compress, decompress, get). An enumeration (CompressionType) is used to simulate a factory pattern and select the method to execute. Performance is evaluated by measuring the execution times of each main function using System.nanoTime() over multiple iterations.

Conclusion

In this project, we established the context of a bit-packing compression project. We presented the problem being addressed and proposed a solution that uses three distinct strategies. In addition, we detailed the implementation methodology and how performance will be evaluated.