Partially-Decompressible Dictionary Based Compression Format for All Flash Array

Yosuke Oyama
Tokyo Institute of Technology
oyama.y.aa@m.titech.ac.jp

Hiroki Ohtsuji, Jun Kato, Kosuke Suzuki, Mitsuru Sato, Eiji Yoshida
FUJITSU LABORATORIES LTD.

I. Introduction

The demand for managing a huge amount of data, so-called "big data", has been increasing in all kinds of fields. All Flash Array (AFA) is a possible solution to face this difficulty, since it provides high-bandwidth and low latency data management with several SSDs and computational resources. The most troublesome problem of exploiting AFA is that each SSD has upper limit of writing times, which shortens the lifespan of the flash memory cells. Data compression techniques can reduce the amount of data to write and solve the lifetime problem.

In order to improve the compression ratio, data blocks should be compressed in bulk to leverage inter-block common features. However, partial read requests are problematic because existing compression methods do not support the partial decompression feature. Without this feature, storage systems are required to decompress entire compressed blocks to read any fraction of data. We propose a dictionary coding based compression technique, which can leverage the benefit of the bulk compression method and support the partial decompression feature. This characteristic provides competitive decompression speed with the case that each block is individually compressed.

II. Partially-Decompressible Dictionary Based Compression Format

Our proposed format is based on LZ4 [1], a fast LZ77-like compression format. LZ4 finds duplicated data literals in a sliding window and keeps only the unique data for data compression. The partial decompression feature is achieved by modifying the sliding window mechanism (Fig. 1). With this mechanism, repeated data occurrence can refer either occurrence on the present block or that on the head block, therefore inter-node occurrences is expected to be compressed for the former case, locally occurrences for the latter case. In addition, when referring the front block, the length of the sandwitched blocks is subtracted from the original offset, hence all offsets of the proposed format can be fitted into 64 KB, which is equals to the size of the original LZ4 offset format.

The proposed format requires the decompression process to read at least two blocks, however it takes less computational cost than the naive bulk compression format.

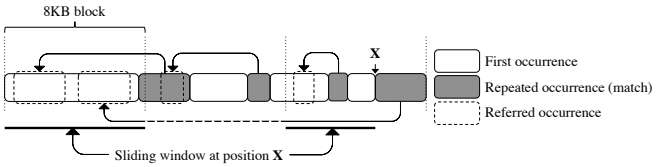

Fig. 1. Proposed Partially-Decompressible Sliding Window Technique

III. Evaluation

We compared the compressed block size and the decompression speed of the proposed format with those of the case that each block is compressed with LZ4 individually (Fig. 2). The evaluated machine has one Intel Xeon CPU E5-2697@2.70GHz, and all blocks are placed on the memory. We arbitrarily set up eight CentOS 7.2 with various applications, and combined VDI files to create one testing data. Note that the blocks of the testing data are deduplicated, consequently the entire size is reduced from 4 GB to 770 MB.

The average compressed size of the proposed format gets the minimum size of 3.73 KB, which is 0.88x of individually compressed size, 4.25 KB. On the other hand, if blocks are sorted randomly average compressed size is 4.29 KB, which is worse than the individual case.

The average decompression speed of one block in four blocks group is 0.85x of the individual case. This performance degradation is expected to be smaller than the case of the naive bulk compression.

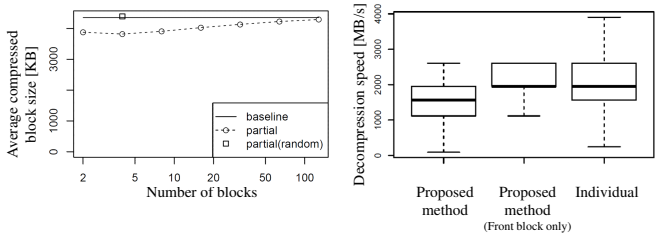

Fig. 2. Average Compressed Block Size (Left) and Decompression Speed (Right)

Reference

[1] Y. Collet. Lz4 - extremely fast compression. https://github.com/Cyan4973/lz4.