Time-based Compression of SZ and Evaluation

SZ development team
Argonne National Laboratory
June 30th, 2018

Section 1. ECP Milestone Targets

This milestone targets the addition in SZ of the capability to compress datasets in time dimension.

Milestone Execution Plan:

- Explore techniques for compression in time dimension
- Integration of compression in time dimension
- Test and performance evaluation on available CORAL systems

Section 2. Design and Implementation

Development of time-based compression for both compression and decompression has been successfully accomplished and integrated in SZ 1.4.13.

Task 1: Explore techniques for compression in time dimension

During the past three months (April ~ June, 2018), we explored how to design and implement the time-based compression based on SZ compression framework.

The traditional SZ compression framework includes three critical steps: (1) data prediction for each data point based on its neighbors, (2) linear-scaling quantization in order to converting all the floating-point values to integer coding numbers, (3) Entropy-encoding (a customized Huffman encoding algorithm) for significantly reducing the data size.

The new SZ compression framework designed in the past three months mainly improved the step 1 in the traditional SZ framework, while the other two steps are unchanged. Specifically, in addition to the space-based data prediction regarding only the current time step, SZ now supports time-based data prediction that involves a few previous time steps at runtime during the simulation. This creates a new avenue/chance to take advantage of the smoothness of the data along the time dimension, such that the data prediction accuracy could be further improved, leading to higher overall compression ratio. Such a new compression framework also allows compression developers to study how to combine space-based prediction and time-based compression together in the future.

Task 2: Integration of compression in time dimension

In order to allow users to decompress the data efficiently, the new SZ compression framework needs to use space-based compression and time-based compression alternatively. In fact, if the compressor adopts only time-based compression, all the snapshots will depend on the very first snapshot. In this situation, if the user wants to decompress the last snapshot (say, time step 1000), it is still necessary to decompress all the snapshots including the time step 1, because of the dependency in time-based prediction. Hence, we need to call space-based compression periodically (e.g., every k time steps) and the rest time steps will use time-based compression strategy. When decompressing the data, the compressor just needs to decompress a few proceeding steps before the target time step, suffering very limited decompression time.

We have integrated the time-based compression into the SZ release 1.4.13.5. How to switch on this option and use the API is described in Section 4.

Task 3: Testing and performance evaluation on available CORAL system.

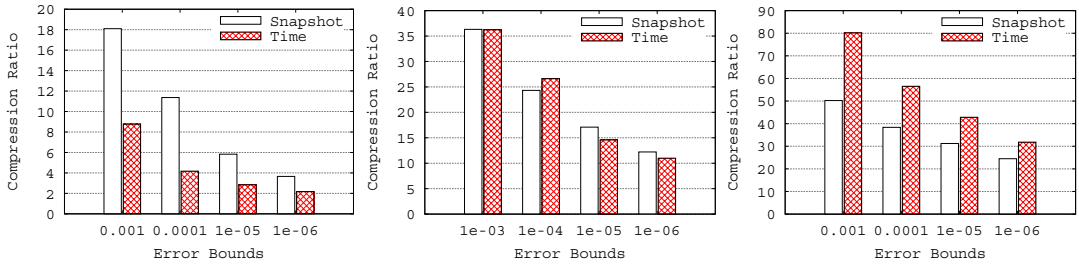
We wrote a couple of examples and scripts to simulate the in-situ data compression/decompression in each data-dumping iteration. Specifically, all the fields that need to be compressed/dumped need to be registered at the beginning (before the key iteration loop). Then, “SZ_compress_ts(&bytes, &outSize)” will be called in each iteration that needs to dump the executed/simulated data. The variable “bytes” refers to the output compressed bytes and the “outsize” is the size of the compressed data (in bytes). Since the fields to compress are registered at the beginning, users don’t need to pass their pointers again inside the key loop of iterations.

We tested the time-based compression vs. space-based compression in SZ on Theta system at Argonne. The entire Theta system has 24 racks, 4,392 nodes with a total of 281,088 cores. 70.272 TB MCDRAM, 843.264 TB DDR4, 562.176 TB SSD, Aries interconnect with Dragonfly configuration, 10 PB Lustre file system, Peak performance of 11.69 petaflops.

The evaluation metrics include rate-distortion, compression ratio, compression rate and decompression rate. Compression ratio is defined as the ratio of the original data size to the compressed data size. Rate-distortion is a figure showing the relationship between the data distortion (evaluated by PSNR) vs. the bit-rate (i.e., the number of bits used to represent one data point on average). In the following, we present the results based on the datasets of three typical fields in the Isabel-hurricane simulation. We compare the results between the snapshot-based compression vs. time-based compression.

Experimental results indicate that neither of the two strategies may always get the better result than the other, depending on the dataset. This observation motivates us to explore a combination of the two strategies in order to optimize the compression quality at runtime.

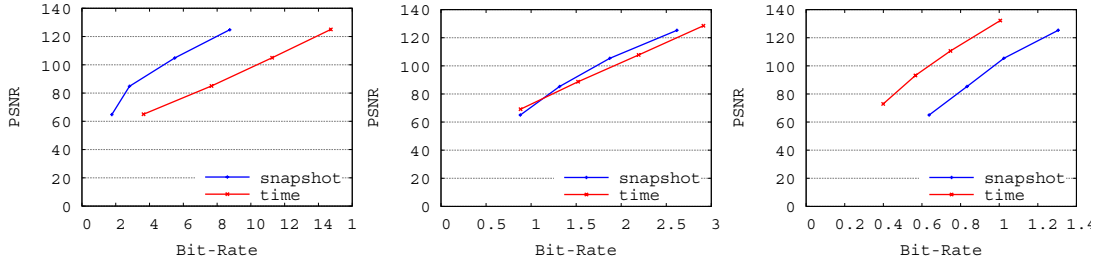
We present the compression ratios in Figure 1. We can clearly observe that the snapshot-based compression has much higher compression ratio than the time-based compression on the Uf field, while it suffers from much lower compression ratio on the QCloudf field. On the Cloudf field, the two strategies lead to similar compression ratios.



(a) Uf (b) Cloudf (c) QCloudf

Figure 1. Compression Ratio of Snapshot-based compression vs. Time-based Compression

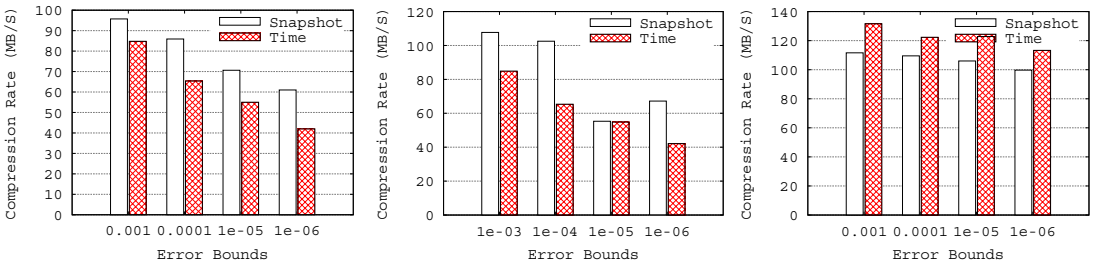
We present the rate-distortion in Figure 2, which also demonstrates that these two strategies are suitable for different fields in particular. For instance, the snapshot-based compression exhibits much better rate-distortion result than the time-based compression, probably because the data exhibit higher smoothness in space than in time-dimension on this field. For the QCloudf field, the time-based compression has better rate-distortion than the snapshot-based compression, because the data of this field is much smoother in the time dimension than in the space.



(a) Uf (b) Cloudf (c) QCloudf

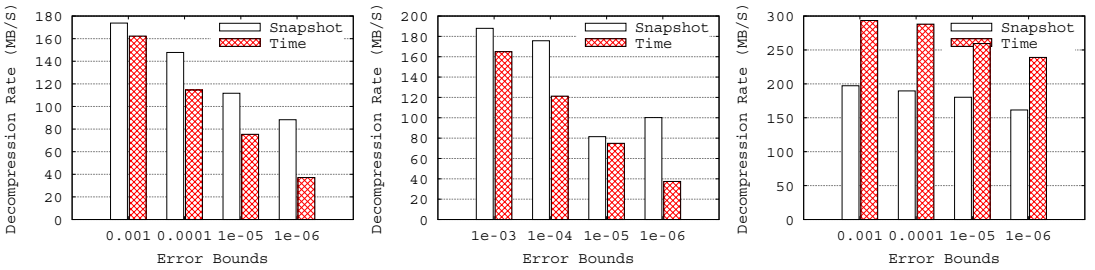
Figure 2. Rate-distortion of Snapshot-based compression vs. Time-based Compression

We present the single-core compression rate and single-core decompression rate in Figure 3 and Figure 4, respectively. It is observed that each of the two different strategies exhibits different processing rates compared to the other one, depending on the datasets. Specifically, the time-based compression works faster than the snapshot-based compression on QCloudf by up to 50% on the decompression and by 20% on the compression.



(a) Uf (b) Cloudf (c) QCloudf

Figure 3. Compression Rate of Snapshot-based compression vs. Time-based Compression



(a) Uf (b) Cloudf (c) QCloudf

Figure 4. Decompression Rate of Snapshot-based compression vs. Time-based Compression

Section 4. Installation and Test

Installation:

./configure --prefix=[Your installation path] **--enable-timecmpr**
make && make install

Test:

You will find testfloat_compress_ts.c and testfloat_decompress_ts.c in the example/ directory.

You can download the testing data from here: http://www.mcs.anl.gov/~shdi/download/consecutive-steps.tar.gz

You can set control the frequency of the snapshot-based compression vs. time-based compression in sz.config. For instance, snapshotCmprStep = 5 means during the simulation, there will be one snapshot-based compression (i.e., individual data compression without dependency to previous time steps) every 5 time steps, and the rest time steps would adopt the time-based compression (i.e., data compression using previous time-step information).

For example, in the datasets you downloaded, you will find there are 20 snapshots. If snapshotCmprStep = 5, then the entire compression will be like this:
step 1: standalone snapshot compression (spatial compression)
step 2: time-step based compression in terms of step 1
step 3: time-step based compression in terms of step 2
step 4: time-step based compression in terms of step 3
step 5: time-step based compression in terms of step 4
step 6: standalone snapshot compression (spatial compression)
step 7: time-step based compression in terms of step 6
step 8: time-step based compression in terms of step 7
step 9: time-step based compression in terms of step 8
step 10: time-step based compression in terms of step 9
step 11: standalone snapshot compression (spatial compression)

We listed two future works to do.

1. Study how to combine the snapshot-based compression and time-based compression. Note that the combination of the two strategies is not simply determining which one to use according to the data features, but also developing a new prediction formula combining them together.
2. Optimization of the moments/frequencies of selecting the two strategies alternatively