CODAR Design Document

Z-checker: a compression assessment tool

Version of January 22, 2018

Franck Cappello and Sheng Di
with Julie Bessac, Hanqi Guo, and Dingwen Tao

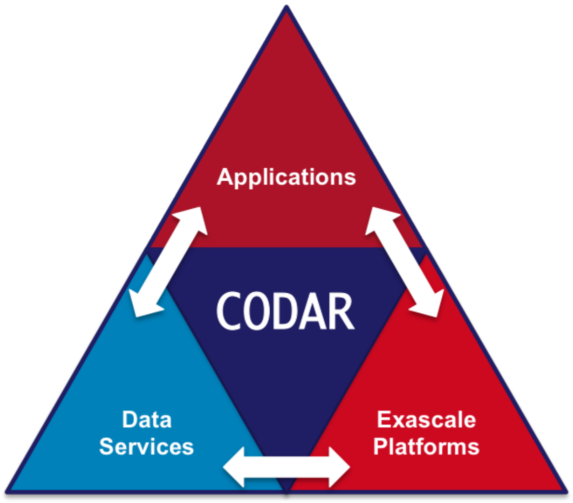

Table of contents

Z-checker: a compression assessment tool

Franck Cappello and Sheng Di
with Julie Bessac, Hanqi Guo, and Dingwen Tao

1 Introduction

Because of the vast volume of data being produced by today's scientific simulations and experiments, lossy data compressors that allow user-controlled loss of accuracy during the compression are relevant solutions for significantly reducing the data size. However, lossy compressor developers and users are missing a tool to explore the features of scientific data sets and understand the data alteration after compression in a systematic and reliable way. To address this gap, we will design and implement a generic framework called Z-checker, such that the users/developers can conveniently select the best-fit, adaptive compressors for different data sets. On the one hand, Z-checker combines a battery of data analysis components for data compression. On the other, Z-checker is implemented as an open-source community tool to which users and developers can contribute and add new analysis components based on their needs. Here we describe the design framework of Z-checker, in which we integrated evaluation metrics proposed in prior work as well as other analysis tools. Specifically, for lossy compressor developers, Z-checker can be used to characterize critical properties of any data set (such as entropy, distribution, power spectrum, principal component analysis, and autocorrelation) to improve compression strategies. For lossy compression users, Z-checker can detect the compression quality (compression ratio, bit rate), provide various global distortion analyses comparing the original data with the decompressed data (PSNR, normalized MSE, rate-distortion, rate-compression error, spectral, distribution, derivatives) and statistical analysis of the compression error (maximum, minimum, and average error, autocorrelation, distribution of errors). Z-checker is also a flexible framework, whose assessment library can be extended with more plugins coded in other programming languages or libraries, such as R and FFTW3. Z-checker provides two alternative execution modes: offline and online. The offline mode allows users to perform the data compression assessment based on generated raw data files or existing decompressed data files. The online mode allows performing an in-situ analysis with running applications. Z-checker also features a visualization interface displaying all analysis results in addition to some basic views of the data sets such as time series.

2 Stakeholders

We list here potential stakeholders for this software and the results of any consultations that we have had with those stakeholders in developing this design document.

2.1 Exascale applications

ECP application (AD) projects demanding lossy compression techniques are a primary source of the problems that Z-checker is designed to address. We have consulted with many, but certainly not all, ECP AD teams to understand their requirement on data compression.

2.2 Software technologies

ECP software technology (ST) projects such as ADIOS, SZ, ZFP, Cinema, ALPINE, VTK-m, and Spack are all potential contributors of software components that may be integrated into the Z-checker assessment tool. Up to now (Jan. 2018), SZ and ZFP have been integrated into Z-checker. Z-checker has been integrated into ADIOS to a certain extent. For other projects/software components, we need to understand any constraints or special requirements that they may impose. We intend to consult with many of those projects in the future.

3 Use cases

Use cases capture the user stories that motivate the work. As mentioned previously, Z-checker is a flexible framework that provides two execution modes based on users' diverse demands for compression assessment and can be extended by adding compressors and analysis function libraries. We first introduce the use cases related to execution modes, then describe the use case of integrating lossy compressors into Z-checker, and finally summarize the ECP applications already assessed using Z-checker.

3.1 Offline analysis

Z-checker provides a series of executables for science users to perform the assessment based on the existing data files that were already produced by simulations. Specifically, given original raw data files, compressed data files, and decompressed data files, the user can perform the analysis by different commands, such as *analyzeDataProperty*, *compareDataSets*, *runOfflineCase*, *generateGNUPlot*, and *generateReport*.

- *analyzeDataProperty* outputs the property information of the target data, such as value range, auto correlation, entropy values, and frequency information.
- *compareDataSets* receives two input files (original raw file and decompressed file) and outputs the data distortion information between them (such as maximum error, distribution of errors, auto correlation of errors, and peak signal-to-noise ratio).
- *runOfflineCase* integrates *analyzeDataProperty* and *compareDataSets*. It allows users to specify a set of data files and decompressed data files to the offline analysis in a batch way and provides multiple options allowing users to do the analysis with different levels/granularities.
- *generateGNUPlot* allows to generate the evaluation figures (in .eps format) based on the analysis.
- *generateReport* allows to generate the assessment report (in .pdf format) based on the analysis.

3.2 Online analysis

The SZ compressor provides a set of easy-to-use APIs for other developers (such as ADIOS developers) to integrate Z-checker's assessment functions. Online analysis is a typical use case allowing users to integrate Z-checker's API functions into their codes and run their applications/tools with Z-checker in parallel.

To activate the MPI version of Z-checker, the user needs to install an mpi library such as MPICH and adopt the "–enable-mpi" option during the installation of Z-checker. We provide examples for parallel data analysis and for parallel compression assessment. The user then needs to switch executionMode from OFFLINE to ONLINE in the configuration file (i.e., zc.config) or set the global variable "executionMode" to be ZC_ONLINE during the initialization (SZ_Init()).

The user needs only to specify the starting point and ending point of the compression and decompression by using Z-checker's interface in order to obtain the online assessment results. The related interfaces are listed below:

- ZC_DataProperty* dataProperty = ZC_startCmpr(propName, ZC_DOUBLE, g, 0, 0, 0, nbLines, M); //start compression
- ZC_CompareData* compareResult = ZC_endCmpr(dataProperty, cmprSize); //end compression
- ZC_startDec(); //start decompression
- ZC_endDec(compareResult, cmprCaseName, decData); //end decompression

An example is shown below:

```
for (i = 0; I < ITER_TIMES; i++) {
    localerror = doWork(nbProcs, rank, M, nbLines, g, h); //perform the simulation
    if(i%50==0) //control the compression frequency over time steps
    {
        //make a name for the current target data property
      sprintf(propName, "%s_%04d", varName, i);
      ZC_DataProperty* dataProperty = ZC_startCmpr(propName, ZC_DOUBLE, \
                    g, 0, 0, 0, nbLines, M); //start compression
      cmprBytes = SZ_compress(SZ_DOUBLE, g, &cmprSize, 0, 0, 0, nbLines, M);
        //end compression
      ZC_CompareData* compareResult = ZC_endCmpr(dataProperty, cmprSize);
      sprintf(cmprCaseName, "%s_%s_%04d(1E-3)", varName, compressorName, i);
      ZC_startDec(); //start decompression
      decData = SZ_decompress(SZ_DOUBLE,cmprBytes,cmprSize,0,0,0,nbLines,M);
      ZC_endDec(compareResult, cmprCaseName, decData); //end decompression

      if(rank==0)
        ZC_writeDataProperty(dataProperty, "dataProperties");

        //free data property generated at current time step
      freeDataProperty(dataProperty);
        //free compression assessment results at current time step
      freeCompareResult(compareResult);
      free(cmprBytes);
      free(decData);
    }
    //Aggregate the residual for checking if the loop meets the convergence
   condition
    if ((i%REDUCE) == 0) {
      MPI_Allreduce(&localerror,&globalerror,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD);
    }
    if(globalerror < PRECISION) {
      break;
    }
}
```

3.3 ECP applications assessed by Z-checker

- ExaSky (HACC cosmology simulation and NYX cosmology simulation)
- QMCPack (quantum chemistry simulation)
- EXAALT (molecular dynamics simulation)
- EXAFEL (Linac Coherent Light Source)
- NWCHEM-X (NWCHEM data sets)
- GAMESS (two-integrals data sets)

4 Requirements

We extract requirements from stakeholder interviews and use cases.

R-1) Z-checker should be able to run experiments on MIRA, Titan, Theta, and more systems.

R-2) Z-checker should be able to run experiments on both Linux workstations and MacOS.

R-3) Z-checker should be able to run the application with alternative reduction algorithms with tunable parameter settings.

R-4) Z-checker should be able to collect and output data distortion results.

R-5) Z-checker should be able to collect and output reduction performance results.

R-6) Z-checker should be able to support different I/O libraries/formats, such as HDF5, NetCDF, Adios.

R-7) Z-checker should be able to provide two execution modes, online and offline.

R-8) Z-checker should be able to support integration with more math libraries such as R and FFTW3.

5 Related work

Here we describe related studies.

Baker et al. [1] investigated the use of data compression techniques on climate simulation data from CESM [2]. They developed an approach for verifying the climate data and used it to evaluate several compression algorithms, including FPZIP [3], ISABELA, APAX, and GRIB2 (with JPEG2000 compression). The verification process included: (1) quantifying the difference between the original and reconstructed data sets via measures of pointwise error, average error (RMSE and NRMSE), and Pearson correlation; and (2) evaluating the reconstructed data in the context of an ensemble of CESM runs with slight perturbations by a CESM port verification tool (CESM-PVT). They determined that the diversity of the climate data requires individual treatment of variables and that the reconstructed data can fall within the natural variability of the system. Laney et al. [4] examined the effects of lossy compression in physics simulations by evaluating two lossy compressors (FPZIP and APAX [5]) in three physics simulation codes. They used physics-based metrics for each simulation to assess the impact of lossy compression. They noted that the characteristics of the compression error must be carefully considered in the context of the underlying physics being modeled. Lakshminarasimhan et al. [6] propose the ISABELA lossy compressor. It performs data compression by B-spline interpolation after sorting the data series. They evaluated the performance of ISABELA with several metrics: compression ratio, maximum compression error (pointwise relative error), average compression error (NRMSE), and compression time. They also evaluated the compressed data by quantitative analysis and visual analysis. In quantitative analysis, they evaluated the Pearson correlation of different data regions and the difference of derived data. In visual analysis, they utilized the visualization tool to present the original data and ISABELA-compressed data. Lindstrom [7] proposed a lossy compression algorithm for floating-point arrays in fixed rate (i.e. ZFP) and evaluated the ZFP results for rate-distortion, rate speed, density spectrum, and derivatives (Morse segmentation of gradients) in several applications using quantitative and visual analysis and visualization Sasaki et al. [8] proposed a lossy compression method (i.e. SSEM) based on wavelet transform and vector quantization. They applied their compression method to a checkpoint/restart technique and evaluate the impact on results of a production climate application (NICAM). Di and Cappello [9] proposed an error-bounded lossy compressor (i.e. SZ) based on curve-fitting and binary representation analysis. They evaluated SZ with the metrics including the maximum compression error (bounded or not bounded), compression ratio, and compression/decompression speed.

6 Design

We introduce the design framework and some implementation details.

6.1 Overview

Z-checker is a novel framework with three important features: (1) it can be used to explore the properties of original data sets for the purpose of data analytics or improvement of lossy compression algorithms; (2) it is integrated with a rich set of evaluation algorithms and assessment functions for selecting best-fit lossy compressors for specific data sets; and (3) it features both static data visualization scripts and an interactive visualization system, which can generate visual results on demand. This interactive mode allows compression algorithm developers and users to dynamically compute analyses on user-selected portions of the data set.

Z-checker is designed to support two processing modes, on-line and off-line, and two display modes, static and interactive. The four modes correspond to four different use cases. The off-line static mode is useful for generating a report on the original data set properties and the compression error, presenting multiple analysis views. The off-line interactive mode is useful for exploring the properties of the original data sets and the compression error. It allows users to dig into some region of interest and reveal local properties that are not present or visible on the whole data set. The on-line static display mode corresponds to in situ data compression and may generate and update the original data set compression analysis as long as the data is presented to Z-checker. The on-line interactive mode allows users to monitor the compression error while the data is produced. Users can zoom in on regions of interest and assess the quality of the data compression. Table 1 summarizes the four use cases of Z-checker, considering the two processing modes and two visualization modes.

The design architecture of Z-checker is presented in Figure 1, which involves three critical parts: *user interface*, *processing module*, and *data module*.

- *User interface* includes three key engines—input engine, output engine, and data visual-

Table 1: Four Use Cases of Z-checker

	Off-line Processing	On-line Processing
Static Display	Load data once and view results by generating local image files.	Load data based on dynamic requests and view results by local image files.
Interactive Display	Load data once and view results on demand via a web page interactively.	Load data dynamically and view results on demand by a web page interactively.

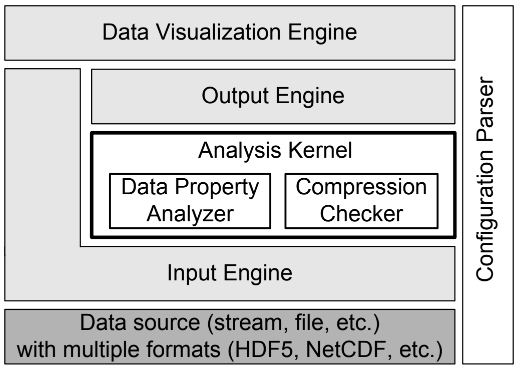

Figure 1: Design Architecture of Z-checker

ization engine—as shown in the light-gray rectangles in Figure 1. They are in charge of reading the floating-point data stream (either original data or compressed bytes), dumping the analyzed data to disks/PFS, and plotting data figures for visualizing analysis results. The Data visualization engine also provides the interactive mode through a web browser interface (details are described later).

- *Processing module* is the core module in the framework, which includes the analysis kernel and configuration parser. The former is responsible for performing the critical analysis, and the latter is in charge of parsing the user's analysis requirements (such as specifying the input file path, the compression command or executable, and the customization for the analysis metrics on demand). Specifically, the analysis kernel is composed of two critical submodules, the data property analyzer and compression checker, which are responsible for exploring data properties based on the original data sets and analyzing the compression results with specified lossy compressors (discussed later in more detail).

- *Data source module*, the dark-gray box in the figure, is the bottom layer in the framework and represents the data source (such as the data stream produced by scientific applications at runtime or the data files stored in the disks).

6.2 Terminology

Application: An executable program with a specific science purpose and a designated input and output format. A given application may have multiple executables for different target architectures and for testing different compiler optimizations, but each such executable must have consistent parameter, input, and output formats.

Application run: A run of an application with a given set of parameters

Lossy compressor: The library that provides the lossy compression capability for scientific data sets. The science data includes different types, such as integer, single-precision and double-precision floating-point data. A lossy compressor should provide both executables and APIs for different users to call the lossy compression functions. A good lossy compressor should also provide multiple options for users to control compression errors and it should respect the compression errors based on the specified error control.

Compression quality: Compression quality includes compression ratio, bit-rate, compression time, decompression time, data distortion, rate-distortion, and so on. Compression ratio is defined as the ratio of the original raw data size to the the size of the compressed data. Bit-rate refers to the number of bits used to represent one data point after the compression. Hence, bit-rate is equal to the ratio of the number of bits used by one data point originally (e.g., 64 for double-precision floating-point data) to the compression ratio. Compression and decompression time are usually evaluated by compression rate and decompression rate respectively. Compression rate is the ratio of the original raw data size (in MB) to the compression time (in seconds). The decompression rate is the ratio of the original raw data size (in MB) to the decompression time (in seconds). Data distortion involves a set of distortion metrics such as maximum compression errors and peak signal-to-noise ratio (PSNR). Rate distortion is a common metric to evaluate the compression quality. Rate, here, means the bit-rate; distortion refers to PSNR.

Offline analysis: Offline analysis means that given specific raw data files, compressed data files, and/or decompressed data files, Z-checker is able to perform the analysis and generate a report.

Online analysis: Online analysis means that Z-checker will be integrated into other tools or applications, performing the analysis while the application is running and generating the assessment report in the end.

6.3 Dependencies

Some functionality in Z-checker depends on third-party libraries, which are listed as follows:

1. Support MPI library: ./configure –prefix=[install_dir] –enable-mpi . (The user needs to install MPI library such as MPICH in advance.)
2. Support NetCDF : usage can be found in the subdirectory NetCDFReader/ (./configure –prefix=[install_dir] –enable-netcdf –with-netcdf-prefix=DIR; NetCDF needs to be installed in advance.)
3. Support HDF5 library: details can be found in HDF5Reader/ (./configure –prefix=[install_dir] –enable-hdf5; the user needs to install HDF5 and set its environment variables according to the HDF5 guide in advance.)
4. Support FFTW3 library: computation of 3D auto correlation requires FFTW3. In particular, 3D auto correlation is computed by using some functions provided by FFTW3. (./configure –prefix=[indall_dir] –enable-fftw3 –with-fftw3-prefix=DIR; the user ndeeds to install FFTW3 in advance.)
5. Support R language/library: the functions coded in R scripts can be executed in the data analysis and compression analysis. In particular, the SSIM function is coded in R and requires the R library in Z-checker. (./configure –prefix=[index_dir] –enable-r –with-r-prefix=DIR)

In addition, the user can also use –with-xxx-include-path and –with-xxx-lib-path to specify the header directory and lib directory, respectively, instead of –with-xxx-prefix. More options can be found by executing ”./configure -help”.

6.4 Z-checker command line format and basic usage

Command: runOfflineCase
Description: This is a command to execute the offline analysis based on given data files and compressed/decompressed data files.

```
Usage: runOfflineCase <options>
Options:
* input information:
    -N <compressor name>: the name of the compressor
    -C <information file> : the file containing the data information
* analysis options:
    -A : perform the full analysis of the compression results
    -a <metric> : perform quick analysis for specific metric
        *metric options: (including all variables)
          cr   : compression ratio (min, avg, max)
          err : compression error (min, avg, max)
          full: complete information listed as above
* metadata options:
    -n : print number of variables
    -m : print the names of variables
    -l : list complete information of all variables
    -p : print all the precisions used in the analysis
* Examples:
    runOfflineCase -C varCmpr.inf -l
    runOfflineCase -C varCmpr.inf -m
    runOfflineCase -C varCmpr.inf -A
    runOfflineCase -C varCmpr.inf -a err
    runOfflineCase -C varCmpr.inf -a cr
```

Command: zccallr
Description: This executable is for users to simply test the execution of R script files. zccallr.c is a good example to show how to call an R script from C.

```
Usage: zccallr <options>
Options:
* Rscript file:
        -s <script_file>: specify the path of the R_script_file
        -c <function_name>: specify the function
* data type:
        -i : integer data (int type)
        -f : single precision (float type)
        -d : double precision (double type)
* input data files:
        -e <endian_type>: endian type of the binary data in input files:
                               0(little-endian); 1(big-endian)
        -A <first data file> : first data file such as original data file
        -B <second data file> : second data file such as decompressed data file
        -C <third data file> : third data file for analysis
        -D <fourth data file> : fourth data file for analysis
        -E <fifth data file> : fifth data file for analysis
        -F <sixth data file> : sixth data file for analysis
* output type of result file:
        -r : print the result on the screen.
        -b : analysis result stored in binary format
        -t : analysis result stored in text format
        -o <output file path> : the path of the output file.
* dimensions:
        -1 <nx> : dimension for 1D data such as data[nx]
        -2 <nx> <ny> : dimensions for 2D data such as data[ny][nx]
        -3 <nx> <ny> <nz> : dimensions for 3D data such as data[nz][ny][nx]
* examples:
        zccallr -s func.R -c add1 -r -f -e 0 \
                -A ../../examples/testdata/x86/testfloat_8_8_128.dat \
                                -3 8 8 128
        zccallr -s func.R -c computeErr -r -f -e 0 \
                -A ../../examples/testdata/x86/testfloat_8_8_128.dat \
                -B ../../examples/testdata/x86/testfloat_8_8_128.dat.sz.out \
                -3 8 8 128
```

6.5 Z-checker configuration files

1. zc.config (specifying the metrics to be used in the assessment)

This configuration file is used by many commands in Z-checker, such as analyzeDataProperty and compareDataSets.

```
#=============================================================
[ENV]
#the path of the R script for special analysis such as KS_test and SSIM
#automatically set during the running of examples/Makefile
RscriptPath = /home/sdi/Z-checker/R/test/data_analysis_script.R

#endianType: either LITTLE_ENDIAN_DATA or BIG_ENDIAN_DATA
#x86, x64 and arm adopt LITTLE_ENDIAN_DATA
#PowerPC (PPC), MAC OS, and KEIL C51 adopt BIG_ENDIAN_DATA
dataEndianType = LITTLE_ENDIAN_DATA

#two statuses: either PROBE (used in detecting/monitoring compression
#results during compression) or ANALYZER (used in gleaning the results
#for plotting and analysis)
#example: checkingStatus = PROBE_COMPRESSOR
#example: checkingStatus = ANALYZE_DATA
#example: chekcingStatus = COMPARE_COMPRESSOR
checkingStatus = PROBE_COMPRESSOR

#two options for execution mode: either ONLINE or OFFLINE;
#ONLINE means running with parallel application such as MPI programs to
#check the compression at runtime (the data are produced by simulations at runtime)
#OFFLINE means running separately from the user application (the data are loaded
#from the files which are already in the disks)
executionMode = ONLINE

[DATA]
#to analyze the properties of the single data set

#compute minimal value of the data set? (1:yes, 0:no)
minValue = 1
#compute maximal value of the data set?
maxValue = 1
#value range of the data set?
valueRange = 1
#average value of the data?
avgValue = 1
#compute entrpy?
entropy = 1
#compute auto correlation of the data (to check smoothness)?
autocorr = 1
```

```
#compute 3D auto correlation of the data (to check smoothness)
autocorr3D = 1
#generate coefficients of the FFT transform?
fft = 1
#generate analysis for laplace
lap = 0

[COMPARE]
#To compare two data sets (e.g., original data vs. decompressed data)

#compression time & compression rate
compressTime = 1
#decompression time & decompression rate
decompressTime = 1
#compression size
compressSize = 1

#compute minimal absolute error between the two data sets
minAbsErr = 1
#compute average absolute error between the two data sets
avgAbsErr = 1
#compute maximal absolute error between the two data sets
maxAbsErr = 1
#compute the auto correlation of the absolute errors (white noises?)
autoCorrAbsErr = 0
#compute the 3D auto correlation of the absolute errors
autoCorrAbsErr3D = 0
#compute the PDF of the absolute errors
absErrPDF = 1
#compute the PDF of the pwr errors
pwrErrPDF = 0

#compute the value-range based minimal relative error
minRelErr = 1
#compute the value-range based average relative error
avgRelErr = 1
#compute the value-range based maximal relative error
maxRelErr = 1

#compute root mean squared error
rmse = 1
#compute normalized root mean squared error (NRMSE)
nrmse = 1
#compute peak signal-to-noise ratio (PSNR)
psnr = 1
#compute signal-to-noise ratio (SNR)
snr = 1

#compute the pearson correlation between the original data values and the
#compression errors
valErrCorr = 1

#compute the pearson correlation coefficient between the two data sets
#(to check five "nine"s?)
pearsonCorr = 1

[PLOT]
#plot the figures based on the data across different compressors or variables
```

```
#extension of property_files, which are under compressors_dir/dataProperties
propertyExtension = prop

plotAutoCorr = 1
plotFFTAmp = 1
plotEntropy = 1

plotCompressionResults = 1

plotAbsErrPDF = 1
```

2. varCmpr.info (specifying the data files to be involved in the assessment)

This configuration file is used by runOfflineCase command. As follows, we present an example configuration file to specify a set of existing data files for assessment.

- ori_data: original data file
- prec: precision of the compression (such as error bound)
- cpr_time: compression time (manually set by users)
- dec_time: decompression time (manually set by users)
- cpr_data: the path of the compressed data file
- dec_data: the path of the decompressed data file

```
#information about the variables and compression demands
ori_data=CLDHGH_1_1800_3600:LITTLE_ENDIAN:FLOAT:1800x3600:../runOfflineCase_testdata/CLDHGH_1_18
prec=1E-3
cpr_time=0.2
dec_time=0.1
cpr_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-3.dat.sz
dec_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-3.dat.sz.out
prec=1E-4
cpr_time=0.4
dec_time=0.2
cpr_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-4.dat.sz
dec_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-4.dat.sz.out
prec=1E-5
cpr_time=0.8
dec_time=0.4
cpr_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-5.dat.sz
dec_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-5.dat.sz.out
prec=1E-6
cpr_time=1.2
dec_time=0.6
cpr_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-6.dat.sz
dec_data=../runOfflineCase_testdata/CLDHGH_1_1800_3600_1E-6.dat.sz.out

ori_data=CLDLOW_1_1800_3600:LITTLE_ENDIAN:FLOAT:1800x3600:../runOfflineCase_testdata/CLDLOW_1_18
prec=1E-3
cpr_time=0.2
dec_time=0.1
cpr_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-3.dat.sz
dec_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-3.dat.sz.out
prec=1E-4
```

```
cpr_time=0.4
dec_time=0.2
cpr_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-4.dat.sz
dec_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-4.dat.sz.out
prec=1E-5
cpr_time=0.8
dec_time=0.4
cpr_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-5.dat.sz
dec_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-5.dat.sz.out
prec=1E-6
cpr_time=1.2
dec_time=0.6
cpr_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-6.dat.sz
dec_data=../runOfflineCase_testdata/CLDLOW_1_1800_3600_1E-6.dat.sz.out
```

3. errBounds.cfg (specifying the error bounds used in the assessment)

This configuration is used only by the One-command assessment of Z-checker-installer. Before executing the key assessment command (runZCCase.sh), the users can modify the errBounds.cfg to specify the error bounds they want to use when doing the compression/decompression test in the assessment. The runZCCase.sh will perform the compression based on the error bounds set in the errBounds.cfg and plot the compression results correspondingly. For example, the following two lines indicate that SZ and ZFP will be used to compress the target data with four error bounds (1E-1, 1E-2, 1E-3, and 1E-4) during the assessment.

SZ_ERR_BOUNDS = “1E-1 1E-2 1E-3 1E-4”

ZFP_ERR_BOUNDS = “1E-1 1E-2 1E-3 1E-4”

The setting comparisonCases = “sz_f(1E-2),sz_d(1E-2),zfp(1E-2) sz_f(1E-4),sz_d(1E-4),zfp(1E-4)” indicates that under which compressors and what error bounds the user wants to do the comparison. Specifically, with the above setting, the users can see the comparison results of compression ratio among the three compressors (sz_f, sz_d and zfp) with error bound = 1E-2 and 1E-4.

```
#Compression error bounds for SZ
SZ_ERR_BOUNDS="1E-1 1E-2 1E-3 1E-4"

#Compression error bounds for ZFP
ZFP_ERR_BOUNDS="1E-1 1E-2 1E-3 1E-4"

#Compression cases used to compare two compressors
comparisonCases="sz_f(1E-2),sz_d(1E-2),zfp(1E-2) sz_f(1E-4),sz_d(1E-4),zfp(1E-4)"

#Number of evaluation results to be shown in the report
numOfErrorBoundCases="2"
```

6.6 output directory layout

```
dataProperties
    variable_1.prop          #basic property analysis results of variable 1
    variable_1.autocorr      #auto correlation of variable 1
    variable_1.lap           #lapalace results of variable 1
    variable_1.fft           #FFT transform coefficients (real and imag)
    variable_1.fft.amp       #amplitude of FFT transform
    variable_1.autocorr3d    #3D auto correlation of variable 1
    .....
    variable_N.prop         #basic property analysis results of variable N
    variable_N.autocorr     #auto correlation of variable N
    variable_N.lap          #lapalace results of variable N
    variable_N.fft          #FFT transform coefficients (real and imag)
    variable_N.fft.amp      #amplitude of FFT transform
    variable_N.autocorr3d   #3D auto correlation of variable N

compressionResults
    compressor:variable.cmp          #basic compression results
    compressor:variable.dis          #distribution of compression errors
    compressor:variable.fft          #FFT transform results of decompressed data
    compressor:variable.fft.amp      #FFT transform amplitudes of decompressed data
    compressor:variable.autocorr     #auto correlation of compression errors
    compressor:variable.autocorr3d   #3D auto correlation of compression errors
```

6.7 One-command installation and usage

We provide an easy method for users to install and use Z-checker by leveraging Z-checker-installer. In Z-checker-installer, one command (i.e., z-checker-install.sh) installs every related dependency, such as SZ, ZFP, gnuplot, and latex. After the simple installation, the user executes two commands to finish the complete assessment procedure for a given data set: (1) execute *createZCCase.sh* to generate an individual workspace based on the user's specified test case name; and (2) execute *runZCCase.sh* to perform the compression automatically by different compressors such as SZ and ZFP, and generate the assessment report with narratives and figures. In the future, if the user wants to update their package based on the already installed Z-checker-installer, they can execute z-checker-update.sh instead of rerunning z-checker-install.sh.

Both Z-checker and Z-checker-installer will be added to Spack.

6.8 A typical test plan

This example describes how to generate data property analysis results and compression results, given original data files (in binary), compressed files, and decompressed files.

```
#Use createOfflineCase.sh to create a use-case directory, which will contain
#all the executables. Then, do the following steps.
#Preparing the testing data
.1. Download the testing data from here:
http://www.mcs.anl.gov/~shdi/download/runOfflineCase_testdata.tar.gz
#Tips: the testing data package contains two original data files and their
#corresponding SZ-compressed files and decompressed files based on different
#error bounds
.2. tar -xzvf runOfflineCase_testdata.tar.gz

#Generate the property analysis results and compression results
.3. ./createOfflineCase.sh testcase1
.4. cd testcase1
.5. ./runOfflineCase -C varCmpr.inf -A -N sz
#Tips: More options can be shown by executing ./runOfflineCase without any input
#options; in the above example, we set the compress name as "sz" because the
#compressed/decompressed data files were generated by sz.

#Generate figures based on the property/compression results in form of GNUPlot
.6. Edit zc.config as follows:
    compressors = sz:[the absolute path of the directory of the test case]
#Tips: sz here refers to the compressor name. You can replace it by your
```

```
#own compressor
    comparisonCases = sz(1E-3)
.7. ./generateGNUPlot zc.config
#Then, you can find rate-distortion eps files generated in the current directory.
#More information can be found in the doc/userguide.pdf
#Tips: You can run the executables or scripts without any inputs to see
#the help information
```

6.9 Release Goals

Release 0.2 for December 2018 will consist of the following functionalities.

- Support both online and offline analysis
- Support reading of HDF5, NetCDF, ADIOS types
- One-command installation and assessment
- Support integration of 3rd-party libraries
- Z-checker web visualization
- Evaluation reports generated by Z-checker

7 Metrics

The purpose of Z-checker is to perform all the compression-related assessment on real science applications to allow the developers to select either the best-fit compressors for specific applications or improve the compression quality, particularly on exascale architectures. The metrics will be the number of applications assessable by Z-checker and the number of I/O libraries Z-checker supports. Our goal is to increase our understanding of the compression quality based on Z-checker and use that knowledge to help improve compressors.

8 Work plan

Date	Milestone
3/31/2017	Final design document for Z-checker v0.1
6/30/2017	Z-checker v0.1.0 (support offline analysis)
12/15/2017	Z-checker v0.1.1 (support offline analysis)
1/31/2018	Final design document for Z-checker v0.2
1/31/2018	Z-checker v0.1.2 (support online analysis partially)
6/30/2018	Z-checker v0.1.3 (support full online analysis)
12/14/2018	Z-checker v0.2

9 Open questions

- How do various compressors set parameters?
- How to implement parallel versions for complicated assessment algorithms, such as FFT and auto-correlation of errors?

- How to implement parallel algorithms (online analysis) based on external calls to third-party libraries such as R?

References

[1] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein, and A. Wegener, “A methodology for evaluating the impact of data compression on climate simulation data,” in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pp. 203–214, ACM, 2014.

[2] Community Earth Simulation Model (CESM). `http://www.cesm.ucar.edu/`, 2016. Online; accessed: 2016-12-30.

[3] P. Lindstrom and M. Isenburg, “Fast and efficient compression of floating-point data,” *TVCG*, vol. 12, no. 5, pp. 1245–1250, 2006.

[4] D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener, “Assessing the effects of data compression in simulations using physically motivated metrics,” *Scientific Programming*, vol. 22, no. 2, pp. 141–155, 2014.

[5] A. Wegener, “Universal numerical encoder and profiler reduces computing’s memory wall with software, FPGA, and SoC implementations,” in *DCC 2013*, p. 528, 2013.

[6] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Ku, C. Chang, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova, “ISABELA for effective in situ compression of scientific data,” *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 524–540, 2013.

[7] P. Lindstrom, “Fixed-rate compressed floating-point arrays,” *TVCG*, vol. 20, no. 12, pp. 2674–2683, 2014.

[8] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, “Exploration of lossy compression for application-level checkpoint/restart,” in *IPDPS 2015*, pp. 914–922, 2015.

[9] S. Di and F. Cappello, “Fast error-bounded lossy HPC data compression with SZ,” in *IPDPS 2016*, pp. 730–739, 2016.