Z-checker (ZC)

User Guide (Version 0.1.0)

Mathematics and Computer Science (MCS)
Argonne National Laboratory, Exascale Computing Project (ECP)
Contact: Sheng Di (sdi1@anl.gov)
Developers: Sheng Di, Dingwen Tao, Hanqi Guo
Advisor: Franck Cappello
Oct, 2017

Table of Contents

1. Brief description

Due to vast volume of data being produced by today's scientific simulations and experiments, lossy data compressor allowing user-controlled loss of accuracy during the compression is a relevant solution 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 design and implement a generic framework, called Z-checker. On the one hand, Z-checker combines a battery of data analysis components relevant for data compression. On the other, Z-checker is implemented as an open-source community tool for which users and developers can contribute and add new analysis components based on their additional analysis demand.

In this user guide, we present the design framework of Z-checker, in which we integrated evaluation metrics proposed in prior work as well as other analysis required by lossy compressor developers and users. For lossy compressor developers, Z-checker can be used to characterize critical properties (such as entropy, distribution, power spectrum, principle component analysis, auto-correlation) of any data set to improve compression strategies. For lossy compression users, Z-checker can obtain the compression quality (compression ratio, bit-rate), provide various global distortion analysis comparing the original data with the decompressed data (PSNR, SNR, normalized MSE, rate-distortion, rate-compression error, spectral, distribution, derivatives) and statistical analysis of the compression error (maximum/minimum/average error, autocorrelation, distribution of errors). Z-Checker can perform the analysis with either course (throughout the whole data set) or fine granularity (by user defined blocks), such that the users/developers can select the best-fit, adaptive compressors for different parts of the data set. Z-checker features a visualization interface displaying all analysis results in addition to some basic views of the data sets such as time series.

2. Design Framework

Z-checker has three important features. (1) Z-checker can be used to explore the properties of original data sets for the purpose of data analytics or improvement of lossy compression algorithms. (2) Z-checker is integrated with a rich set of evaluation algorithms and assessment functions for selecting best-fit lossy compressors for specific data sets. (3) Zchecker features both static data visualization scripts and an interactive visualization system, which can generate visual results on demand.

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

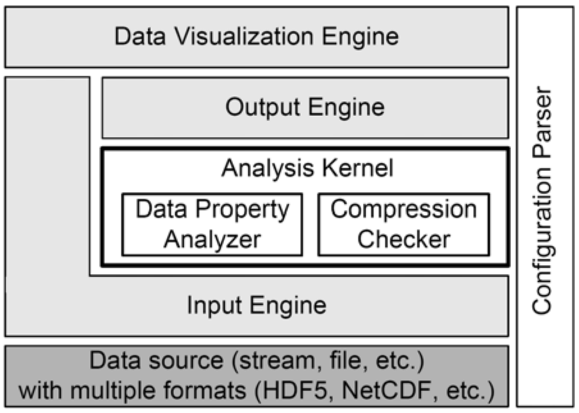

Fig. 1 Design Framework of Z-checker

- **User interface** includes three key engines, namely input engine, output engine, and data visualization 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 the visualization of analysis results. The Data Visualization Engine also provides the interactive mode through a web-browser interface.

- **Processing module**, in the whole framework, is the core module, which includes Analysis Kernel and Configuration Parser. The former is responsible for performing the critical analysis, and the latter is in charge of parsing user's analysis requirement (such as specifying input file path, specifying compression command/executable, and customizing the analysis metrics on demand). Specifically, the analysis kernel is composed of two critical sub-modules, data property analyzer and compression checker, which are responsible for exploring data properties based on original data sets and analyzing the compression results with specified lossy compressors, to be discussed later in more details.
- **Data source module** (shown as dark-gray box in the figure) is the bottom layer in the whole framework, and it represents the data source (such as data stream produced by scientific applications at runtime or the data files stored in the disks).

2. Installation of Z-checker

We provide two alternative ways to install Z-checker, stand-alone installation or z-checker-installer.

The former is installing Z-checker individually, without any third-party dependencies or compression libraries. The z-checker-installer is recommended for generic users, because it will check if the required third-party libraries are already installed. If not, it will download and install the dependencies automatically before installing Z-checker. It also includes some state-of-the-art lossy compressors such as SZ and ZFP, and also provides a set of scripts allowing users to get the compression report just by running one command or so.

2.1 Stand-alone installation

The Z-checker library/tool can be cloned by the following command:
git clone https://github.com/CODARcode/z-checker
or downloaded from http://www.mcs.anl.gov/~shdi/download/z-checker-0.1.0.tar.gz
Perform the following three simple steps to finish the installation:

./configure --prefix=[INSTALL_DIR]
make
make install

You'll find all the executables in [INSTALL_DIR]/bin or [DOWNLOAD_PACKAGE]/examples.
The static library (.a file) and shared library (.so file) can be found in [INSTALL_DIR]/lib.
We describe the key executables as follows:

2.2 Quick start (using Z-checker-installer)

1. git clone https://github.com/CODARcode/z-checker-installer
2. Run z-checker-install.sh will download latexmk, gnuplot, Z-checker, ZFP, and SZ and install them one by one automatically, and then add the patches to let ZFP and SZ fit for Z-checker.
(Note: in the future, for updating the package based on your already installed z-checker-installer, you can execute z-checker-update.sh for simplicity)

3. Testing

The testing cases can be found in **[ZC_Package]/examples**
You can use "make clean;make" to recompile all the example codes, or compile them by the customized Makefile.bk as follows:
make -f Makefile.bk
(Makefile.bk allows you to compile your customized source codes.)

For simplicity, you can use [ZC_Package]/example/*test.sh* to test some examples (including data property analysis code and the wrapper code of calling R script from Z-checker).
More tests related to compression analysis need to be performed with some data compressor such as SZ (https://collab.cels.anl.gov/display/ESR/SZ). More details will be described later.

3.1 Testing based on stand-alone installation

The executables are in [INSTALL_DIR]/bin or [DOWNLOAD_PACKAGE]/examples.

- analyzeDataProperty (or analyzeDataProperty.sh):

  Usage: ./analyzeDataProperty.sh [datatype (-f or -d)] [data directory] [dimension sizes....]
  Example: ./analyzeDataProperty.sh -f /home/shdi/CESM-testdata/1800x3600 3600 1800
  Description: The analysis results will be put in the directory called dataProperties.
  In particular,
  ▫ [variable_name].fft keeps the spectrum information (generated by FFT), including real part and imaginary part.
  ▫ [variable_name].fft.amp contains the amplitude of the spectrum data.

- compareDataSets (or compareDataSet.sh)

  Usage: compareDataSets [dataType -f or -d] [config_file] [compressionCase] [varName] [oriDataFilePath] [decDataFilePath] [dimension sizes...]
  Example: compareDataSets -f zc.config SZ 8_8_128 testfloat_8_8_128.dat

testfloat_8_8_128.dat.out 8 8 128

Description:

If you just want to compare the reconstructed data with the original data, you can use executable compareDataSets, as shown below.

compareDataSets [config_file] [oriDataFilePath] [decDataFilePath] [dimension sizes...]

Three output files (.cmp, .autocorr, and .dis) will be stored in the directory compareResults/:

- .cmp file keeps the general information about the compression results.
- .autocorr file keeps the auto correlation of the compression errors with different lags.
- .dis is about the PDF of compression errors.

- generateReport (or generateReport.sh):

Usage: ./generateReport [config_file] [title of dataset]

Example: ./generateReport zc.config CESM-ATM-tylor-data

Description: generateReport assumes that the data properties and compression results are already generated in the three key directories, dataProperties, compressionResults and compareCompressors, and it will generate the figures and the pdf report based on them.

- generateGNUPlot:

Usage: ./generateGNUPlot [config_file]

Example: ./generateGNUPlot zc.config

Description: generateGNUPlot will find compression results in the three directories: dataProperties, compressionResults and compareCompressors, and plot figures accordingly.

3.2 Automatic testing with Z-checker-installer

After the simple one-command installation described in Section 2.2, you can download the testing data sets, CESM-ATM and MD-simulation (exaalt). The two testing data sets are available only for the purpose of research of compression.

- CESM-ATM: http://www.mcs.anl.gov/~shdi/download/CESM-ATM-tylor.tar.gz
- MD-simulation (exaalt): http://www.mcs.anl.gov/~shdi/download/exaalt-dataset.tar.gz

Then, you are ready to perform the compression quality checking by Z-checker.
You can generate compression results with SZ and ZFP using the following simple steps:
(Note: you have to run z-checker-install.sh to install the software before doing the following tests)

(1) Configure the error bound setting and comparison cases in errBounds.cfg.
(Example settings are already in the errBounds.cfg. You can change it based on your demand, such as the list of compression errors)
(2) Create a new test-case, by executing "createNewZCCase.sh [test-case-name]". You need to replace [test-case-name] by a meaningful name.
For example:

[user@localhost z-checker-installer] ./createNewZCCase.sh CESM-ATM-tylor-data

(3) Perform the checking by running the command "runZCCase.sh": runZCCase.sh [data_type] [error-bound-mode] [test-case-name] [data dir] [dimensions....].

Example:

[user@localhost z-checker-installer] ./runZCCase.sh -f REL CESM-ATM-tylor-data /home/shdi/CESM-testdata/1800x3600 3600 1800

Description:

- -f means that the data set is single-precision floating-point data.
- REL means that the error bounds used in the compression are based on value_range.
- CESM-ATM-tylor-data is the name of the testing case.
- [dimensions….] follows the column-major style. In the above example, the matrix in C programming code style is 1800x3600, so the dimensions sizes should be 3600 1800.

(4) Then, the report are generated in z-checker-installer/Z-checker/[test-case-name]/report. All the figures can be found in z-checker-installer/Z-checker/[test-case-name]/report/figs.

For more information, you can locate the specific results as follows:

- In z-checker-installer/Z-checker/[test-case-name]/dataProperties: The *.prop files contain the property analysis results about the data set, such that min value, max value and entropy value. The *.fft* files are the FFT analysis result about the data set. The *.autocorr* files contain the auto-correlation coefficient results with different lags.
- In z-checker-installer/Z-checker/[test-case-name]/compressionResults: The *.cmp files contain the results about compression quality, such as compression ratio, maximum compression error, PSNR, SNR, MSE, compression time, and decompression time. The *.dis files are about the distribution of compression errors. The *.autocorr files contain the auto-correlation coefficients of the compression error with different lags.
- In z-checker-installer/Z-checker/[test-case-name]/compareCompressors: This directory contains the results about the comparison among different compressors. For example, rate-distortion_psnr_FREQSH_1_1800_3600.dat.txt presents rate-distortion figure (PSNR vs. bit-rate) for the variable FREQSH with the dimension size 1800x3600; the file crate_sz_f(1E-2)_sz_d(1E-2)_zfp(1E-2).txt compares the compression rate (i.e., compression speed) across three compression solutions with the same error bound 10^{-2}. cratio_sz_f(1E-2)_sz_d(1E-2)_zfp(1E-2).txt refers to compression ratio and drate_sz_f(1E-2)_sz_d(1E-2)_zfp(1E-2).txt indicates the decompression rate.

3.3 Manual testing without Z-checker-installer

Testing procedure:

Download a compression library, such as SZ (https://collab.cels.anl.gov/display/ESR/SZ), and then put the data compression monitoring interfaces before and after the compression/decompression function. Recompile the compression library. And then, perform the compression using some data set. The analysis results (such as compression rate, compression ratio, compression errors) will be stored in the directory called compareData/.

Example (with SZ):

1. Download SZ-1.4.9 from https://collab.cels.anl.gov/display/ESR/SZ

2. Install SZ package by the following steps:
(tar –xzvf sz-1.4.9-beta.tar.gz;cd sz-1.4.9.1-beta;./configure --prefix=[INSTALL_DIR]; make;make install)

3. Copy the testing code testfloat_CompDecomp.c from Z-checker's examples/ directory to SZ's example/ directory, and compile it by the following command (you need to replace the bolded parts by the installation paths on your own machine)

#compile_CompDecomp.sh
#!/bin/bash
SZPATH=**[SZ_INSTALL_DIR]**
ZCPATH=**[Z-Checker_INSTALL_DIR]**
SZFLAG="-I$SZPATH/include -I$ZCPATH/include $SZPATH/lib/libsz.a $SZPATH/lib/libzlib.a $ZCPATH/lib/libzc.a"
gcc -lm -g -o testfloat_CompDecomp testfloat_CompDecomp.c $SZFLAG

4. Run testfloat_CompDecomp:
./Testfloat_CompDecomp [sz_config_file] [zc_config_file] [compressor_name] [testcase] [absErrBound] [input_data_file_path] [dimension_sizes…..]

Example:
Set the compression mode in configuration file sz.config to *SZ_BEST_SPEED*, then:

absErrBound=1E-3
compressor_case=**sz1.4_f**($absErrBound) #sz1.4_f refers to sz1.4(best_speed_mode)
testcase=varName
datapath=testdata/x86/testfloat_8_8_128.dat

testfloat_CompDecomp sz.config zc.config "${compressor_case}" ${testcase} ${absErrBound} ${datapath} 8 8 128

(***Note***: You need to copy zc.config from Z-checker's examples/ directory to SZ's example/ before running the above command.)

5. After running testfloat_compDecomp, the compression results will be kept in compareData/ directory.

"sz(1E-3):varName.cmp" keeps the compression results, including compression time, decompression time, compression rate, decompression rate, PSNR, SNR, compression factor, maximum compression error, etc.

"sz(1E-3):varName.autocorr" keeps the auto-correlation values of the compression errors with different lags.

“sz(1E-3):varName.dis” keeps the distribution (PDF) of the compression errors.

6. Change the compression mode in sz.config to be *SZ_BEST_COMPRESSION* , and then rewind the above steps 4 with the compressor_case tagged as **sz1.4_c**, as shown below.

absErrBound=1E-3

compressor_case=**sz1.4_c**($absErrBound) #sz1.4_c is sz1.4(best_compression_mode)

testcase=varName

datapath=testdata/x86/testfloat_8_8_128.dat

testfloat_CompDecomp sz.config zc.config “${compressor_case}” ${testcase} ${absErrBound} ${datapath} 8 8 128

7. Now, we are ready to compare the compression results between the two “compressors”: sz1.4_f vs. sz1.4_c, as follows:

Go back to Z-checker’s examples/ directory, and then set the compressors as follows:

compressors = sz1.4_f:sz-1.4.9-beta/example sz1.4_c:sz-1.4.9-beta/example

comparisonCases = sz1.4_f(1E-3),sz1.4_c(1E-3)

Note: the format of the compressors string is [compressor_name]:[directory of running command]. The comparisonCases follows such as format: [compressor_case1],[compressor_case2] [compressor_case3],[compressor_case4] ……

8. Run the following command to generate the comparison figure files (in .eps format):

./generateGNUPlot zc.config

(Note: the eps files will be generated and put in the current directory).

4. Application Programming Interface (API)

Programming interfaces are provided in C, and we provide a wrapper to call R script in the C code.

4.1 Initialization and finalization of the Z-checker environment

4.1.1 ZC_Init

int ZC_Init(char *configFilePath);

4.1.2 ZC_Finalize

void ZC_Finalize();

4.2 Generic I/O

4.2.1 ZC_readDoubleData in bytes

double *ZC_readDoubleData(char *srcFilePath, int *nbEle);

4.2.2 ZC_readFloatData in bytes

float *ZC_readFloatData(char *srcFilePath, int *nbEle); (in bytes)

4.2.3 ZC_writeFloatData_inBytes

void ZC_writeFloatData_inBytes(float *data, int nbEle, char* tgtFilePath);

4.2.4 ZC_writeDoubleData_inBytes

void ZC_writeDoubleData_inBytes(double *data, int nbEle, char* tgtFilePath);

4.2.5 ZC_writeData in text

void ZC_writeData(void *data, int dataType, int nbEle, char *tgtFilePath);

4.3 Data property analysis

4.3.1 ZC_genProperties

ZC_DataProperty* ZC_genProperties(char* varName, int dataType, void *oriData, int r5, int r4, int r3, int r2, int r1);

4.3.2 ZC_printDataProperty

void ZC_printDataProperty(ZC_DataProperty* property);

4.3.3 ZC_writeDataProperty

void ZC_writeDataProperty(ZC_DataProperty* property, char* tgtWorkspaceDir);

4.3.4 ZC_loadDataProperty

ZC_DataProperty* ZC_loadDataProperty(char* propResultFile);

4.4 Data Comparison

4.4.1 ZC_compareData

ZC_CompareData* ZC_compareData(char* varName, int dataType, void *oriData, void *decData, int r5, int r4, int r3, int r2, int r1);

4.4.2 ZC_printCompareResult

void ZC_printCompareResult(ZC_CompareData* compareResult);

4.4.3 ZC_writeCompareResult

void ZC_writeCompareResult(ZC_CompareData* compareResult, char* solution, char* varName, char* tgtWorkspaceDir);

4.4.4 ZC_loadCompareResult

ZC_CompareData* ZC_loadCompareResult(char* cmpResultFile);

4.5 Setting monitoring calls in compressor codes

4.5.1 ZC_startCmpr

ZC_DataProperty* ZC_startCmpr(char* varName, int dataType, void *oriData, int r5, int r4, int r3, int r2, int r1);

4.5.2 ZC_startCmpr_withDataAnalysis

ZC_DataProperty* ZC_startCmpr_withDataAnalysis(char* varName, int dataType, void *oriData, int r5, int r4, int r3, int r2, int r1);

This function includes the data analysis. That is, it will analyze the data and output the analysis results into the dataProperty/ directory. The analysis result includes auto-correlation of the data, spectrum result (based on FFT), distribution of the data, entropy, etc.

Note: data analysis is costly, so we suggest to call ZC_startCmpr instead of ZC_startCmpr_withDataAnalysis in most cases.

4.5.2 ZC_endCmpr

ZC_CompareData* ZC_endCmpr(ZC_DataProperty* dataProperty, int cmprSize);

4.5.3 ZC_startDec

void ZC_startDec();

4.5.4 ZC_endDec

void ZC_endDec(ZC_CompareData* compareResult, char* solution, void *decData);

4.6 Calling R script from Z-checker

4.6.1 ZC_executeCmd_RfloatVector

int ZC_executeCmd_RfloatVector(char* cmd, int* count, float** data);

4.6.2 ZC_executeCmd_RdoubleVector

int ZC_executeCmd_RdoubleVector(char* cmd, int* count, double** data);

4.6.3 ZC_executeCmd_RfloatMatrix

int ZC_executeCmd_RfloatMatrix(char* cmd, int* m, int* n, float** data);

4.6.4 ZC_executeCmd_RdoubleMatrix

int ZC_executeCmd_RdoubleMatrix(char* cmd, int* m, int* n, double** data);

4.7 Plotting Z-checker analysis data

4.7.1 ZC_plotCompressionRatio

void ZC_plotCompressionRatio();

4.7.2 ZC_ plotHistogramResults

void ZC_plotHistogramResults(int cmpCount, char** compressorCases);

4.7.3 ZC_plotComparisonCases

void ZC_plotComparisonCases();

4.7.4 ZC_plotAutoCorr_CompressError

void ZC_plotAutoCorr_CompressError();

4.7.5 ZC_plotAutoCorr_DataProperty

void ZC_plotAutoCorr_DataProperty();

4.7.6 ZC_plotFFTAmplitude_OriginalData

void ZC_plotFFTAmplitude_OriginalData();

4.7.7 ZC_plotFFTAmplitude_DecompressData

void ZC_plotFFTAmplitude_DecompressData();

4.7.8 ZC_ plotErrDistribtion

void ZC_plotErrDistribtion();

4.8 Plot generic data by Gnuplot

4.8.1 genGnuplotScript_linespoints

char** genGnuplotScript_linespoints(char* dataFileName, char* extension, int fontSize, int columns, char* xlabel, char* ylabel);

4.8.2 genGnuplotScript_histogram

char** genGnuplotScript_histogram(char* dataFileName, char* extension, int fontSize, int columns, char* xlabel, char* ylabel, long maxYValue);

4.8.3 genGnuplotScript_lines

char** genGnuplotScript_lines(char* dataFileName, char* extension, int fontSize, int columns, char* xlabel, char* ylabel);

4.8.4 genGnuplotScript_fillsteps

char** genGnuplotScript_fillsteps(char* dataFileName, char* extension, int fontSize, int columns, char* xlabel, char* ylabel);

4.9 Generating analysis report

TBD

5 Configuration file

You can switch on/off the metrics to check in the configuration file (zc.config) based on your demand. Please see the comments in the sz.config file for details.
In particular, there are two modes for running z-checker, “probe” mode and “analysis” mode. The former indicates the online checking by running the compressor, and the latter is to collect the compression results based on the previous probe-mode runs.

The configuration file actually is not a must when running a compressor with the z-checker as a probe/monitor. If configuration file is not used in the probe-mode running, all metrics will be switched on by default.

Note that if the z-checker is running in the "analysis" mode, please do remember to switch the checkingStatus parameter to "analysis" from "probe".

6. Version history

The latest version (**version 0.1.0**) is the recommended one.

Version	**New features**
Zc 0.1.0	Prototype of Z-checker. It's able to perform the data analysis for the original data set and compression quality analysis based on specific data compressors.

7. Q&A and Trouble shooting

TBD

<END>